البرمجة والخوارزميات

(الرسم والبرمجة بلغة) C++

وقل رب زدني علما
جامعة دمشق
Damascus University

منشورات جامعة دمشق

كلية العلوم

البرمجة والخوارزميات

(الرسم والبرمجة بلغة C++)

الدكتور

خالد الخنيفس

أستاذ في قسم الرياضيات

جامعة دمشق

١٤٣٢ - ١٤٣٣ هـ

٢٠١١ - ٢٠١٢ م


وقل رب زدني علما
جامعة دمشق
Damascus University

فهرس المحتويات

مقدمة

لقد تزايدت أهمية لغات البرمجة منذ النصف الثاني من القرن العشرين، حيث اعتبر تطوير عملية البرمجة غرضية التوجه أمراً ضرورياً وخاصة في مجال النظم التطبيقية من أجل رفع مستوى الاستخدامات الحاسوبية الى درجة عالية من الفعالية والإتقان.

إن لغة البرمجة C++ هي لغة البرمجة الفعالة. فعندما تتعقد المسائل، يلجأ المبرمجون إلى لغة البرمجة C++. علماً أنّ هناك العديد من لغات البرمجة الأخرى، لكنها تفتقر إلى القوة والشمولية المتوفرة في لغة البرمجة C++ . لغة البرمجة فيجوال بيسك (Visual Basic) مفيدة لإنشاء برامج غير كبيرة نسبياً وغير متطلبة لسرعة التنفيذ، ولغة التجميع (Assembly) جيدة لكتابة برامج تشغيل الأجهزة والتحكم بها، ولغة البرمجة الجافا تمكن تصميم مواقع على الشبكة العالمية (الانترنت) لـ (World Wide Web). إن لغة الجافا مستقاة من لغة البرمجة C++. لكن لإنشاء برنامج رئيسي مستقل، فانّ لغة البرمجة C++ أشهر لغة تتمتع بالقوة والمرونة لإنتاج أسرع البرامج وأفضلها أداءً. لذلك، تحتل لغة البرمجة C++ موقعاً رئيساً في تعليم البرمجة أكاديمياً، وأي مبرمج يحتاج إلى تعلُّم لغة البرمجة C++ ، لأن برامجها سهلة الفهم والقراءة. سوف نشرح المفاهيم البرمجية في لغة البرمجة C++ ، وسنستخدم البرامج و الأشكال لتوضيح المفاهيم.

وأمل أن يكون هذا الكتاب حافزاً للطالب على زيادة الاهتمام في مجال لغات البرمجة غرضية التوجه، والارتقاء بمستواه، مما يؤدّي إلى النهوض باستخدام النظم البرمجية إلى المستوى المطلوب.

والله من وراء القصد

المؤلف

دمشق 2007/2/8

وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل الأول

العمل في بيئة لغة البرمجة C++ المرئية

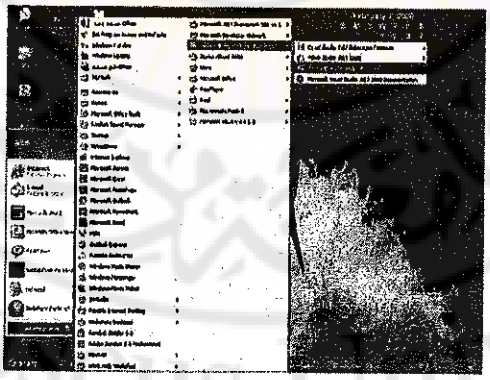
1- مقدمة:

إنَّ جميع البرامج الواردة تمَّ اختبارها والتأكد من صحة سير عملها وذلك ضمن بيئة Microsoft Visual Studio.Net 2003 وكذلك ضمن بيئة Microsoft Visual C++ 6.0. لذلك سنبين في هذا خطوات كتابة وإنشاء برنامج بسيط.

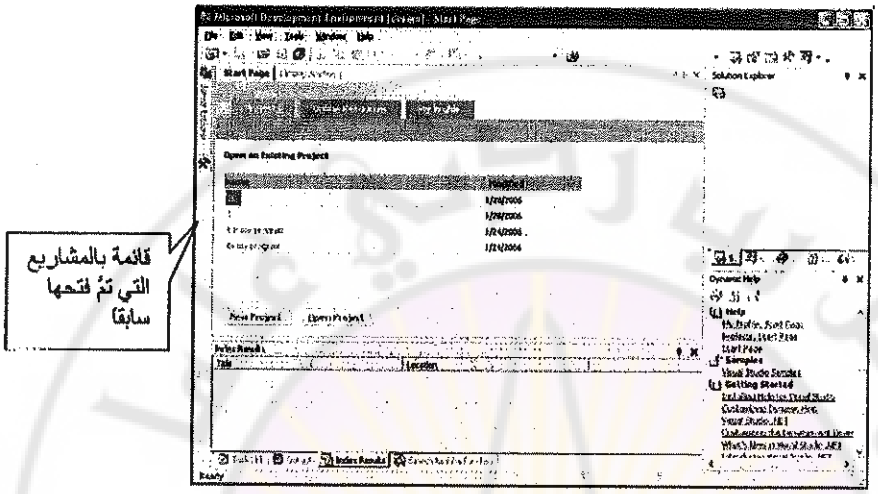
2- كيفية كتابة برنامج ضمن بيئة Microsoft Visual Studio.Net 2003

سنقوم بكتابة البرنامج الذي يظهر الجملة its my first project.

- كخطوة أولى يجب التأكد من تنصيب البرنامج Microsoft Visual Studio.Net 2003 على الجهاز.
- بالضغط على أيقونة البرنامج أو كما في الشكل (1-1) بالضغط على البرنامج من قائمة البرامج تظهر لنا الشاشة المبينة في الشكل (1-2)

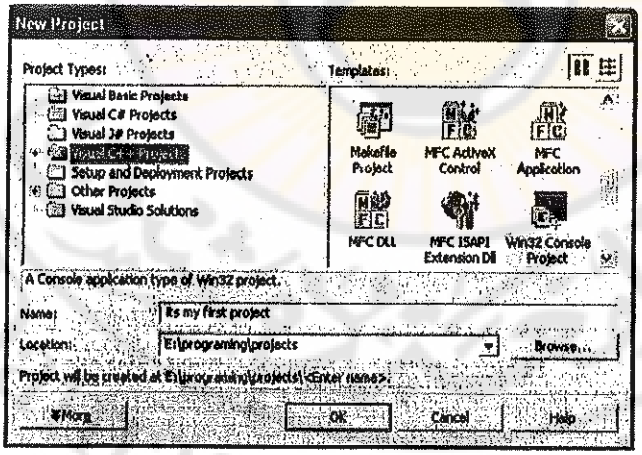


الشكل (1-1)



الشكل (1-2)

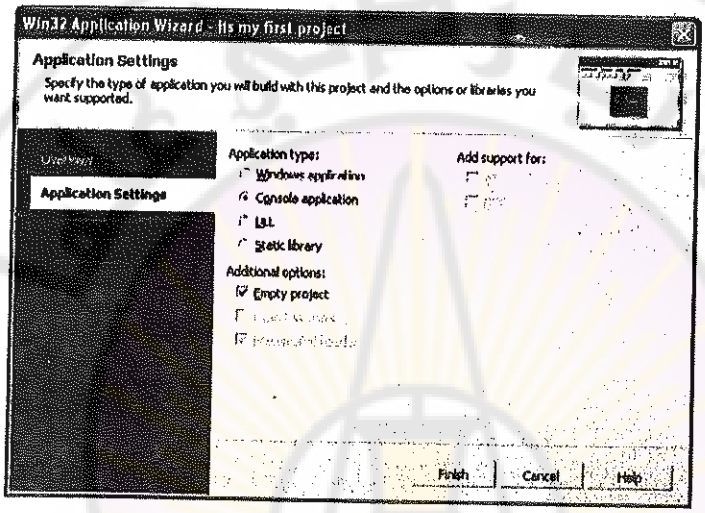
من القائمة File ضمن الشكل (1-2) نضغط على new project فتظهر لنا الشاشة الموضحة بالشكل (1-3)



الشكل (1-3)

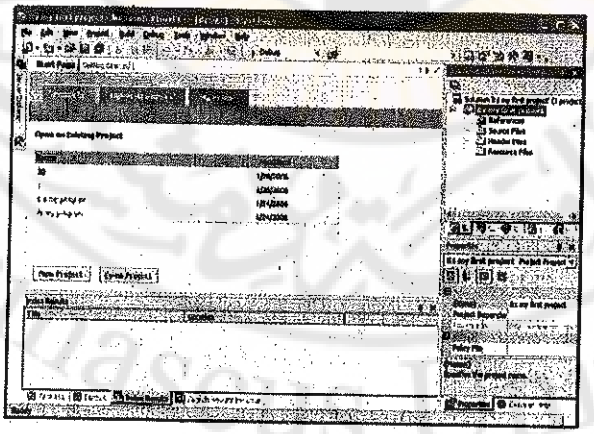
☜ نختار نوع المشروع الذي نريد إنشاءه على أنه Visual C++ Projects ثـم نختار نموذج (Template) المشروع على أنه Win32 Console Project. ثم نختار اسماً مناسباً للمشروع ثم نحدد المكان الذي سيخزن فيه المشروع.

☜ بالضغط على الزر oK ننتقل إلى الشاشة في الشكل (1-4)، نضـع إشـارة صح بجانب Empty Project، ثم نضغط على الزر Finish.



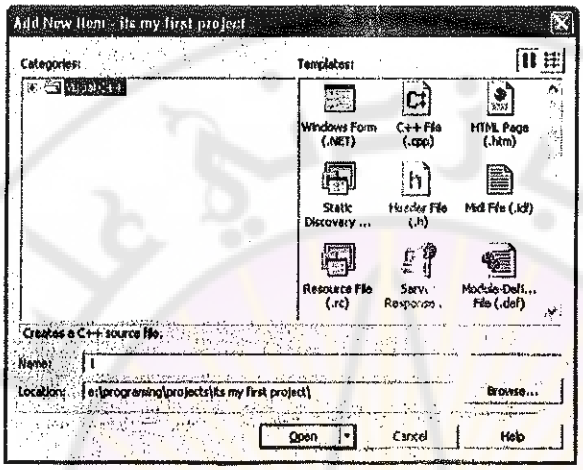
الشكل (1-4)

☜ بعد الضغط على الزر Finish تظهر على شاشة الشكل (1-5)



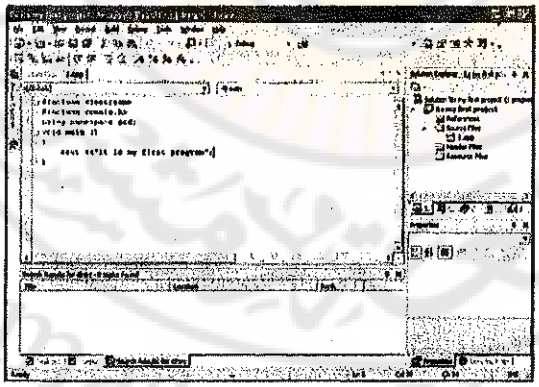
الشكل (1-5)

☜ نضغط بالزر إليميني فوق Source File من القائمة Add نختار Add New Item فتظهر الشاشة المبينة في الشكل (1-6).



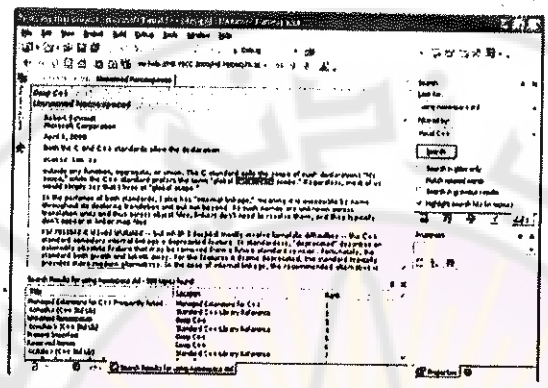
الشكل (1-6)

☜ نختار نوع الملف على أنه C++ File (.cpp) ونحدد له الاسم والموقع الذي سنحفظ فيه الملف. ثم نضغط على الزر Open. فنحصل على شاشة الشكل (1-7). نقوم بكتابة البرنامج وهنا يجب ملاحظة النقاط التالية:



الشكل (1-7)

يجب إضافة السطر using namespace std. لمعرفة تفاصيل عن سبب وضع هذه الجملة يمكن مراجعة ملفات الـ Help، كمثال الملف الموضح في الشكل (1-8).



الشكل (1-8)

1. تؤمن بعض النسخ غير المدعومة من قبل Microsoft الدالة clrscr والتي تقوم بمسح الشاشة في تطبيقات الـ Dos. في الحقيقة لا توجد أية Win32 Application Programming Interface (API) أو دالة تقوم بهذه المهمة.

توجد طريقتان للقيام بهذه المهمة (مسح الشاشة) لتطبيقات Win32 Console

الطريقة الأولى:

استخدام التعليمة system("cls");.

الطريقة الثانية:

كتابة الدالة برمجياً كما يوضح الـ code التالي:

```
/* Standard error macro for reporting API errors */
 #define PERR(bSuccess, api){if(!(bSuccess)) printf("%s:Error %d
from %s \
   on line %d\n", __FILE__, GetLastError(), api, __LINE__);}
```

```
void cls( HANDLE hConsole )
{
  COORD coordScreen = { 0, 0 };    /* here's where we'll home the
                        cursor */
  BOOL bSuccess;
  DWORD cCharsWritten;
  CONSOLE_SCREEN_BUFFER_INFO csbi; /* to get buffer info
*/
  DWORD dwConSize;                 /* number of character cells in
                        the current buffer */

  /* get the number of character cells in the current buffer */

  bSuccess = GetConsoleScreenBufferInfo( hConsole, &csbi );
  PERR( bSuccess, "GetConsoleScreenBufferInfo" );
  dwConSize = csbi.dwSize.X * csbi.dwSize.Y;

  /* fill the entire screen with blanks */

  bSuccess = FillConsoleOutputCharacter( hConsole, (TCHAR) ' ',
    dwConSize, coordScreen, &cCharsWritten );
  PERR( bSuccess, "FillConsoleOutputCharacter" );

  /* get the current text attribute */

  bSuccess = GetConsoleScreenBufferInfo( hConsole, &csbi );
  PERR( bSuccess, "ConsoleScreenBufferInfo" );

  /* now set the buffer's attributes accordingly */

  bSuccess = FillConsoleOutputAttribute( hConsole,
csbi.wAttributes,
    dwConSize, coordScreen, &cCharsWritten );
  PERR( bSuccess, "FillConsoleOutputAttribute" );

  /* put the cursor at (0, 0) */

  bSuccess = SetConsoleCursorPosition( hConsole, coordScreen );
  PERR( bSuccess, "SetConsoleCursorPosition" );
```

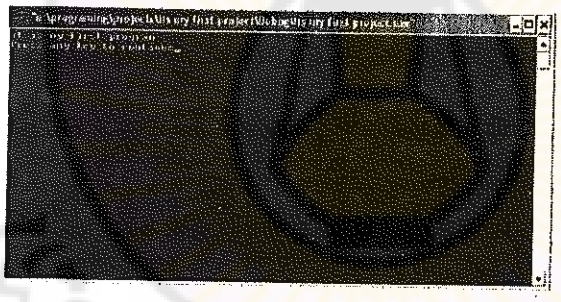
```
    return;
}
```

- بعد أن قمنا بكتابة البرنامج. يمكننا من القائمة Debug أن نختار start، أو يمكن أن نضغط F5. فنلاحظ ظهور الشاشة السوداء واختفاءها مباشرة دون أن تسنح الفرصة لمشاهدة الخرج.

- أما إذا رغبنا في مشاهدة الخرج نقوم بالضغط على Start Without Debugging من القائمة Debug أو نضغط ctrl+F5. فتظهر الشاشة الموضحة في الشكل (1-9).

ملاحظة:

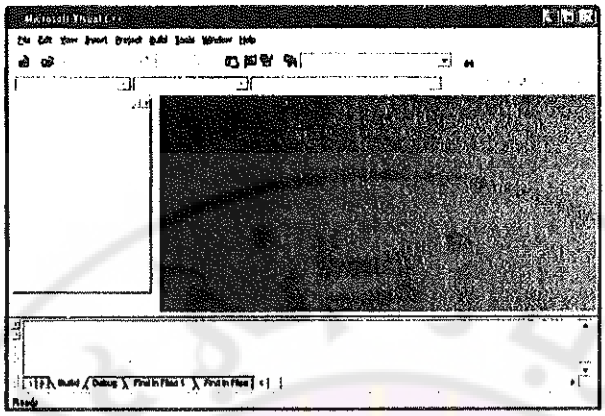
إذا أردنا إضافة برنامج موجود مسبقاً في ملف إلى المشروع الجديد الذي قمنا بإنشائه، نقوم بالضغط على Add Existing Item والذهاب إلى المكان المخزن فيه هذا الملف



الشكل (1-9)

3- كيفية كتابة برنامج ضمن بيئة Microsoft Visual C++ 6.0

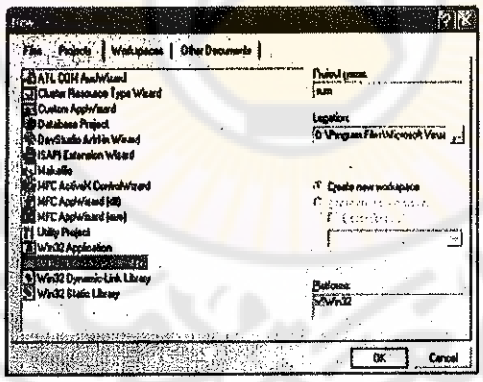
سنقوم فيما يلي بتوضيح خطوات إنشاء برنامج يقوم بجمع عددين صحيحين



الشكل (1-10)

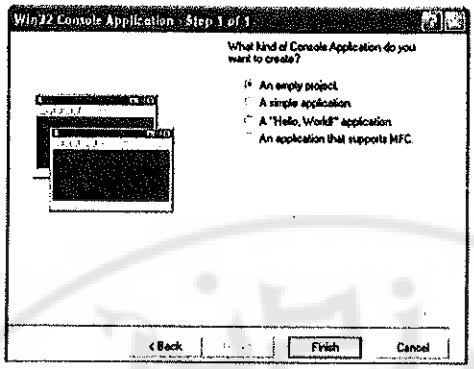
يبين الشكل (1-10) الشاشة التي ستظهر لدى الضغط على أيقونة البرنامج Microsoft Visual C++ 6.0

☜ من القائمة File نضغط على new فتظهر لنا شاشة الشكل (1-11) نختار من قائمة المشاريع نوع المشروع على أنه Win32 Console Application ونحدد أسمه وموقعه ثم نضغط على الزر oK. فتظهر شاشة الشكل (1-12)

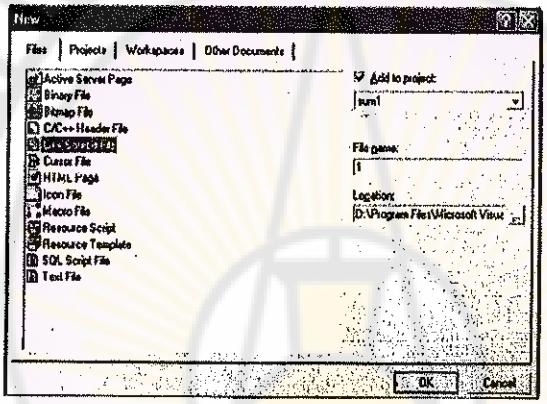


الشكل (1-11)

☜ نختار المشروع على أنه فارغ An empty project. ثم نضغط على Finish، ثم نعود ونختار new من القائمة file، فتظهر الشاشة الموضحة في الشكل (1-13).

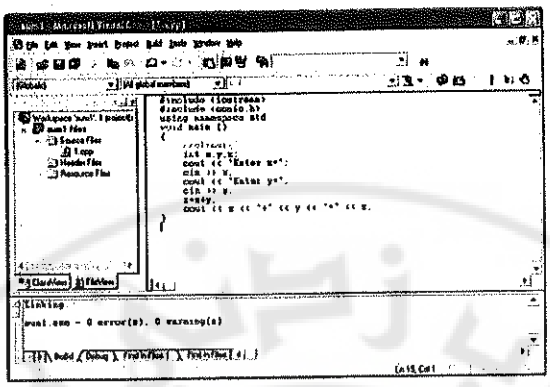


الشكل (1-12)

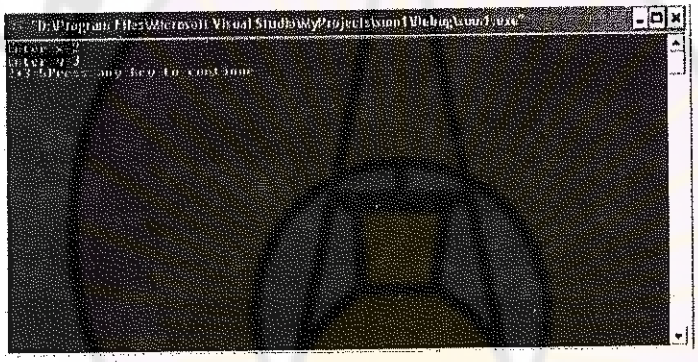


الشكل (1-13)

- نختار نوع الملف على أنه C++ Source File. ثم نضغط على الزر oK.
- يمكن ملاحظة أن الملف قد أضيف إلى مجلد Source File، نقوم بكتابة الـــ code ونضغط على ctrl+F5 أو على Execute filename.exe من القائمـــة Build فتظهر لنا الشاشة الموضحة في الشكل (1-15).



الشكل (1-14)



الشكل (1-15)

الفصل الثاني

مقدمة في الخوارزمية

1- تعريف الخوارزمية

الخوارزمية هي عبارة عن مجموعة من الخطوات المنطقية المرتبة وفق قواعد ما لحل مسألة محددة.

سنميز عدة حالات تمكننا من فهم وحل المسائل المطروحة وحلّها.

1. كتابة الخوازمية لغوياً:

وفي هذه الحالة تصاغ المسألة وحلها إن أمكن لغوياً باستخدام مفردات سهلة الفهم والاستيعاب. (المفردات المستخدمة هي المفردات المستخدمة في النظام)

2. صياغة الخوارزمية رياضياً:

إن المسألة وحلها يُصاغان بشكل رياضي يمكن من فهم المسألة وإمكانية برمجتها حاسوبياً. وهذه الصياغة تُمكنّ من إعداد برنامج حاسوبي إذا كان الأمر يتطلب ذلك.

3. المخططات التدفقية (المخططات الإلية):

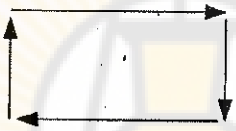
ترتيب المخططات التدفقية خطوات حل المسألة وألياتها وفق خطوات الخوارزمية وذلك لتسهيل عملية الحل وفهم المسألة وبرمجتها. وقد استخدمت رموز متفق عليها دولياً لتمثيل هذه المخططات التدفقية. وفيما يلي نبين الرموز المستخدمة في بناء مخططات تدفقية.

1) البداية والنهاية

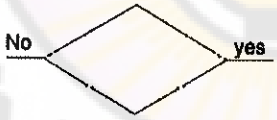
2) الإدخال والإخراج

3) المعالجة

4) التدفق



5) القرار أو الموافقة



وتستخدم هذه الأدوات في صياغة وبناء المخطط التدفقي للمسألة بحيث يمكن برمجتها بأية لغة برمجية مطلوبة.

ملاحظة:

توجد عدة أنواع للمخططات التدفقية، مخططات تدفق سير الوثائق في نظام ما ولها أدواتها الخاصة.

وفيما يلي نبين بعض الأمثلة التي تُمكّن من فهم كيفية بناء هذه الأشكال المختلفة للخوارزميات.

مثال:

اكتب خوارزمية إيجاد حل معادلة من الدرجة الثانية بشكلها العام.

$$ax^2 + bx + c = 0$$

2- صياغة المسألة لغوياً

a) المجهول هو x.

b) a, b, c أعداد حقيقة معلومة.

c) إيجاد مميز المعادلة وبيان فيما إذا كان ميز هذه المعادلة موجباً أم سالباً أو معدوماً.

d) إيجاد الناتج (قيمة x) وفق ما يلي:

- إذا كان المميز موجباً يوجد جذران حقيقيان.
- إذا كان المميز معدوماً يوجد جذر مضاعف حقيقي.
- إذا كان المميز سالباً يوجد جذران عقديان.

3- صياغة المسألة رياضياً:

الخطوة الأولى:

الدخل a, b, c أعداد حقيقية

الخطوة الثانية:

أوجد $\Delta = b^2 - 4ac$

الخطوة الثالثة:

إيجاد الجذور x_1, x_2

إذا كان $\Delta < 0$ اذهب إلى الخطوة الرابعة

وإلا

$$x_1 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$x_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

اذهب إلى الخطوة الخامسة

الخطوة الرابعة

أوجد

$$x_1 = \frac{-b}{2a} + i\frac{\sqrt{|\Delta|}}{2a}$$

$$x_2 = \frac{-b}{2a} - i\frac{\sqrt{|\Delta|}}{2a}$$

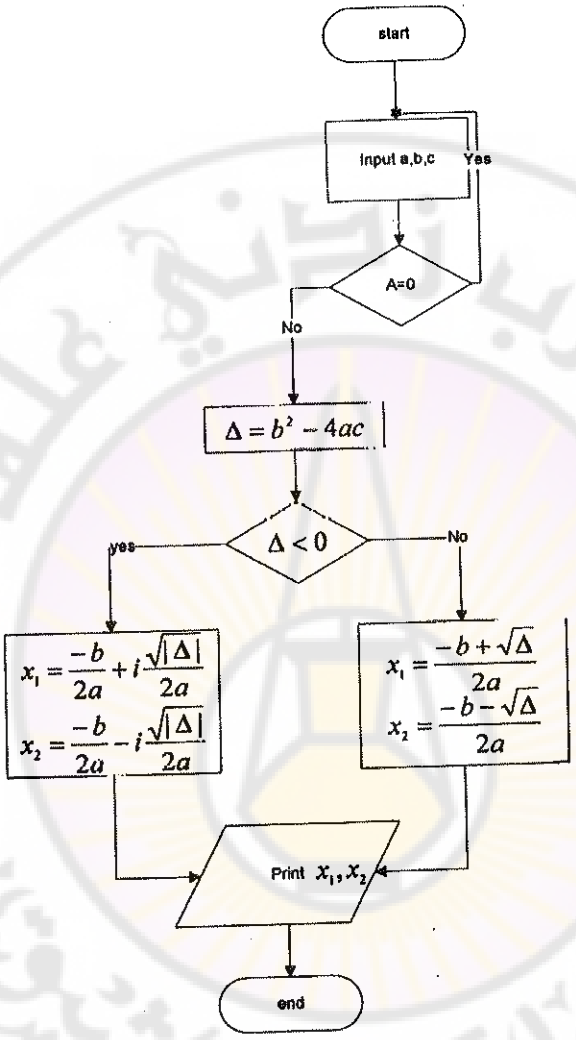
الخطوة الخامسة

اطبع x_1, x_2

الخطوة السادسة

النهاية

4- المخطط التدفقي للمسألة:




وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل الثالث

مكونات لغة البرمجة C++

1- مقدمة

سوف نعرض لغة البرمجة C++ بأسلوب بسيط لكي يتمكن المهتمّ من فهم أسلوب البرمجة غرضية التوجه. لقد تمّ تطوير لغات البرمجة غرضية التوجه OOP (Object Oriented Programming) بسبب القيود وأساليب البرمجة التي تفرضها لغات البرمجة التقليدية على المبرجين. وفيما يلي نعرض القيود التي كانت تفرضها لغات البرمجة التقليدية.

إن لغة البرمجة باسكال و لغة البرمجة C و لغة البرمجة بيسك و لغة البرمجة فورتران وغيرها من لغات البرمجة التقليدية هي لغات برمجة إجرائية Procedural. أي أن كل عبارة برمجية في اللغة تنفذ مهمة إدخال، أو عملية حسابية، أو عملية منطقية، أو إخراج. إن البرنامج المكتوب بلغة برمجة إجرائية هو عبارة عن لائحة من التعليمات يقوم الحاسوب بتنفيذها.

وعندما تصبح البرامج كبيرة فإنه من الصعب فهم التعليمات ومتابعة المهام التي تنفذها. ومن هنا يصعب فهم البرنامج إلا إذا كانت مجزئة إلى إجرائيات ودوال. لهذا السبب تمّ اعتماد أسلوب الإجرائيات والدوال كطريقة لجعل البرامج أسهل للقراءة والفهم (إن مفهوم الدالة مستخدم في لغة البرمجة C و في لغة البرمجة C++). يقسم البرنامج إلى دوال تملك كل دالة مهمة برمجية معرفة بشكل دقيق وواضح ومحدد.

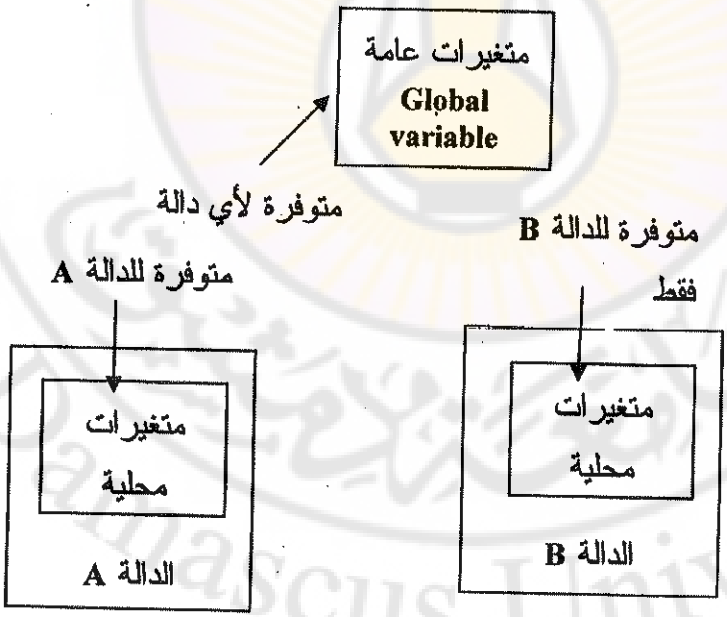
الوحدة هي عبارة عن عدد من الدوال تمّ تجميعها في ما يسمى Unit. إن تجزئة البرنامج إلى دوال ووحدات تعتبر أساس البرمجة البنيوية structured programming، ولقد أثر ذلك في أسلوب كتابة البرنامج وتصميمه.

تهتمّ اللغة البرمجة الإجرائية بتنفيذ المهام والتعليمات وتتمّ القراءة من لوحة المفاتيح أو من ملف ويقوم المترجم من التحقق من صحة البرنامج والتأكد من وجود أخطاء أو تنفيذ مهمة برمجية محددة. وتؤدي تجزئة البرنامج إلى دوال وإجرائيات إلى سهولة متابعة تنفيذ البرنامج وفهمه.

إنّ البيانات ومعالجتها هي سبب وجود البرامج. رغم ذلك تُعطى البيانات أهمية ثانوية في تنظيم اللغات البرمجة الإجرائية.

إن لغات البرمجة ، لغة البرمجة باسكال و لغة البرمجة C، تدعم متغيرات محلية، وهي متغيرات معرفة في الدوال والإجرائيات. لكن المتغيرات المحلية غير مفيدة للبيانات المهمة المشتركة ويتمّ تداولها من قبل عدة دوال مختلفة وإجرائيات. يبين الشكل (3-1) العلاقة بين المتغيرات العامة والمحلية.

تعرف المتغيرات العامة خارج أي دالة لكي يصبح بالإمكان الوصول إليها من كل الدوال والإجرائيات.

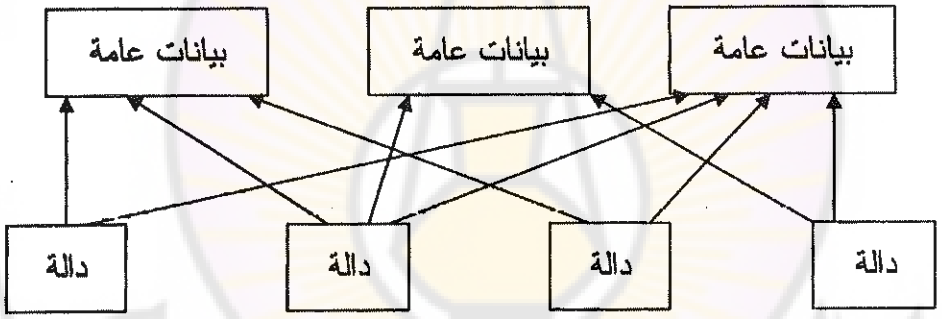


الشكل (3-1) المتغيرات العامة والمحلية

إن طريقة تخزين البيانات تصبح مهمة بسبب الوصول إليها من قبل عدة دوال. لذالك لا يمكن تغيير ترتيب البيانات من دون تغيير كل الدوال التي تستعملها. وإذا أضفنا عناصر بيانات جديدة ، سنحتاج إلى تعديل كل الدوال التي نستعملها لكي تتمكن هذه الدوال من استعمال هذه العناصر الجديدة. سيكون من الصعب تحديد كل تلك الدوال والأصعب تعديلها كلها بطريقة متشابهة. يبين الشكل (3-2) العلاقة بين الدوال والبيانات في البرامج المكتوبة بلغةالبرمجة الإجرائية.

نخفي البيانات عن كل الدوال ما عدا المهمة منها. سيؤدي هذا إلى حماية البيانات من العبث ونستطيع صيانة البيانات بسهولة.

إن تصميم البرامج الإجرائية صعب. والمشكلة هي أن مكوناتها الرئيسة ــ الدوال وبنى البيانات ــ لا تقلّد العالم الحقيقي.



الشكل (3-2) المنهج الإجرائي

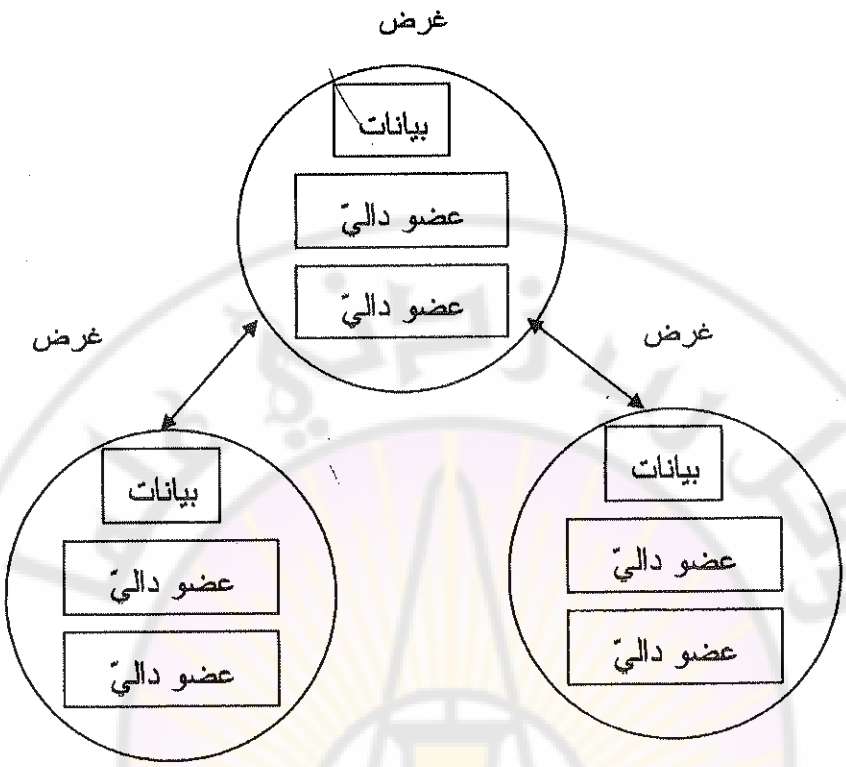
ومن مشاكل لغات البرمجة التقليدية صعوبة إنشاء أنواع بيانات جديدة. تتضمن لغات البرمجة عادة عدة أنواع من المتغيرات، متغيرات الأعداد صحيحة **(integer)** ومتغيرات الأعداد الحقيقية (العائمة) **(float)** ومتغيرات المحارف **(char)**،..... الخ. فإذا أردنا نوع متغيرات تتعامل مع الأعداد المركبة أو الإحداثيات.. الخ. مع كميات لا تتعامل معها أنواع المتغيرات المألوفة. فإن القدرة على إنشاء أنواع متغيرات خاصة جديدة تسمى المتغيرات التوسعية **extensibility** كونها تمكن من توسيع

إمكانية اللغة البرمجية. إنّ لغات البرمجة التقليدية غير قابلة للتوسيع عادة. كما أن كتابة البرامج التقليدية وصيانتها معقدة أكثر من كتابة وصيانة البرامج المكتوبة بلغات البرمجة غرضية التوجه.

الفكرة الأساسية في اللغات البرمجة غرضية التوجه هي دمج المتغيرات والدوال التي تعمل على تلك المتغيرات في غرض واحد يسمى **(object)**.

إن دوال المعرفة في الصف، نسميها في لغة البرمجة C++ أعضاء دالية **member functions** هي الوسيلة الوحيدة للوصول إلى البيانات. لقراءة عنصر بيانات مخزّن في غرض ما، نستدعي عضواً دالياً تابعاً لذلك الصف. سيقوم العضو الدالي بقراءة العنصر ويعيد القيمة. وهذه الدوال لا تمكن من الوصول إلى البيانات بشكل مباشر. فالبيانات مخفية، فهي محمية من والعبث والتعديلات الخطأ. إن البيانات ودوالها في لغة البرمجة C++ مغلّفة **encapsulated** في صف واحد.

لتعديل البيانات المخزنة في الغرض، نحدد الدوال التي نتعامل معها (الأعضاء الدالية) في الغرض ثم نقوم بالتعديل المطلوب. لا تستطيع أي دالة أخرى الوصول إلى البيانات في هذا الغرض. وبذلك يمكن كتابة البزامج وصيانتها بشكل أسهل.



الشكل (3-3) طريقة البرمجة غرضية التوجه

2- التنظيم في لغات البرمجة غرضية التوجة

إن البرمجة غرضية التوجه لا تهتمّ بتفاصيل عمل البرنامج. بل تتعامل مع التنظيم الإجمالي للبرنامج. إن معظم العبارات في لغة البرمجة C++ مشابهة للعبارات المستعملة في لغات البرمجة الإجرائية والكثير منها مطابق لعبارات اللغة C. قد يكون عضو دالي بأكمله في برنامج في لغة البرمجة C++ شبيه تماماً لدالة إجرائية في لغة البرمجة C. فإذا نظرنا إلى السياق البرمجي نتمكن من تحديد ما إذا كانت العبارة أو الدالة جزءاً من برنامج C إجرائي أو برنامج في لغة البرمجة C++ غرضي التوجه.

3- الكائنات الفعلية

لحل مشكلة برمجية في لغة غرضية التوجه نبحث عن كيفية تجزئتها إلى أغراض. إن التفكير بالأغراض بدلاً من الدوال يظهر مدى سهولة تصميم البرامج. ينتج ذلك عن المطابقة القوية بين الأغراض في المفهوم البرمجي والأغراض في الحياة. فيما يلي نعرض بعض الأنواع النموذجية للأغراض:

* العناصر في بيئة مستخدم الكمبيوتر:
  - النوافذ.
  - القوائم.
  - الكائنات الرسومية (الخطوط والمستطيلات والدوائر).
  - الفأرة ولوحة المفاتيح ومحركات الأقراص والطابعة.
* بنيات تخزين البيانات:
  - مصفوفات مخصصة.
  - مكدسات.
  - قوائم مرتبطة.
  - أشجار ثنائية.
* غرض بشري:
  - موظفون.
  - طلاب.
  - زبائن.
  - مندوبون.
* مجموعة من البيانات:
  - جرد.

- ملف معلومات.
- قاموس.

4- أسماء المتغيرات والدوال

يتمّ استعمال محرف التسطير السفلي (_) للفصل بين الكلمات في أسماء المتغيرات في لغة البرمجة C++، أو كتابة الكلمات ملتصقة ببعضها بعضاً مع تكبير أول حرف من كل كلمة. الأقواس ( ) بعد الاسم تشير إلى دالة. تسمح معظم المحارف في لغة البرمجة C++ بإنشاء أسماء (أسماء المتغيّرات وأسماء الدوال، وغيرها) طويلة قدر ما تشاء، لكن يجب أن تكون الأحرف الـ 32 الأولى ذات معنى للمترجم. من المسموح استعمال الأحرف الكبيرة والصغيرة وحرف التسطير السفلي والأعداد من 0 إلى 9، لاتبدأ أسماء المتغيرات والدوال برقم.

5- الصفوف

الأغراض في لغات البرمجة غرضية التوجه هي مثيلات من الصفوف **instances of classes**. كل لغات البرمجة تقريباً تتضمن أنواع متغيرات. فنوع المتغيرات **int**، الذي يعني **integer** (متغيرات الأعداد صحيحة)، معرّف بشكل مسبق في لغة البرمجة C++. نعرف قدر ما نشاء من المتغيرات ذات النوع الأعداد الصحيحة:

```
int x;
int count;
int y,z;
```

وبالطريقة نفسها يمكن تعريف الكثير من الأغراض التابعة للصف نفسه، تلعب الصف دور قالب. وهي تحدد ما هي المتغيرات والبيانات والدوال التي سيتمّ شملها في أغراض تلك الصف. إن تعريف الصف لا يؤدي إلى إنشاء أي غرض، وتعريف

المتغير (مثلاً وجود الكلمة المحجوزة **int**) لا يؤدي إلى إنشاء أي متغير من هذا النوع.

6- الوراثة

تؤدي فكرة الصفوف إلى فكرة الوراثة **inheritance**. حيث تجزء الصف إلى صفوف فرعية. المبدأ في هذا النوع من التجزئة هو أن كل صف فرعي يملك خصائص مشتركة مع الصف الذي يرثه. ويملك كل صف فرعي خصائص خاصة به.

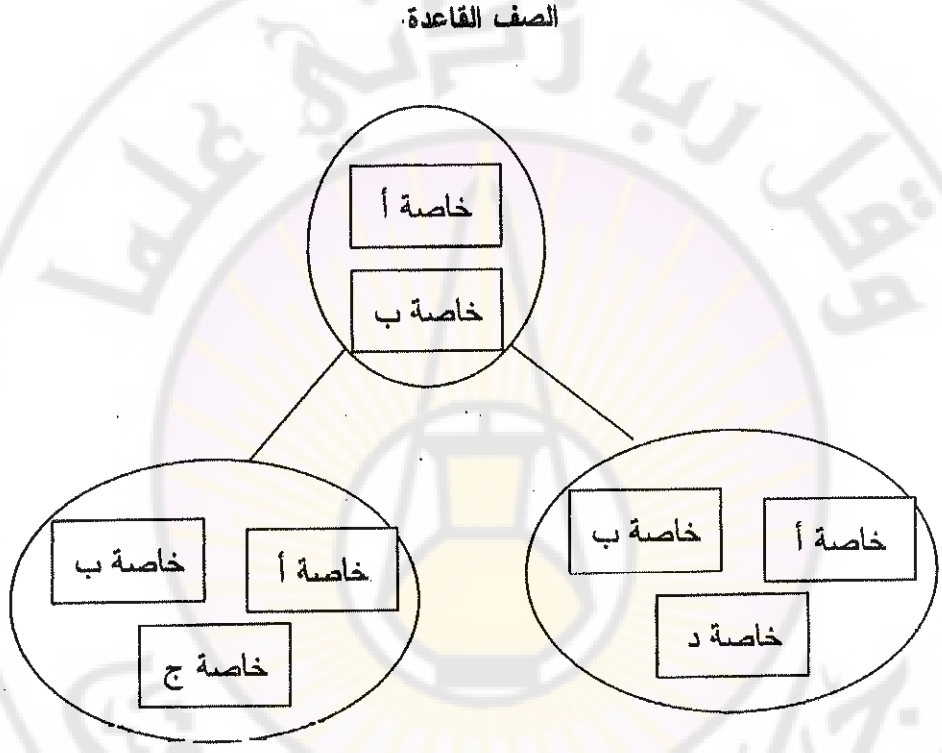
ويبين الشكل (3-4) مفهوم الوراثة. في هذا الشكل أن الميزتين أ و ب، اللتين تشكلان جزءاً من الصف الأساسية، مشتركتان بين كل الأعضاء الوارثة، لكن لكل صف وارثة لها خصائص خاصة بها.

يمكن استعمال صف كأساس لصف فرعي واحدة أو أكثر. في لغة البرمجة C++، يسمى الصف الأصلي القاعدة **base class**؛ ويمكن تعريف صفوف أخرى تتشارك خصائصها، لكن تضيف خصائصها الذاتية أيضاً. إنها تسمى الصفوف المشتقة **derived classes**.

إذا وجدنا ثلاثة أقسام مختلفة في برنامج إجرائي تقوم بالعمل نفسه ، نستخرج العناصر المشتركة من هذه الأقسام الثلاثة ونضعها في دالة واحدة. وتستطيع الأقسام الثلاثة في البرنامج استدعاء الدالة لتنفيذ الأعمال المشتركة. كما يمكنها تنفيذ أعمالها الخاصة أيضاً. بشكل مماثل، يحتوي الصف القاعدة على عناصر مشتركة بين مجموعة من الصفوف المشتقة. ومثلما تفعل الدوال في البرنامج الإجرائي، تقصّر الوراثة البرنامج غرضي التوجه وتوضح العلاقة بين عناصره.

إن مفهوم الوراثة تمكن المبرمج من أخذ صف موجودة، ومن دون تغييرها، يضيف ميزات وقدرات إليها. يتمّ هذا خلال اشتقاق صف جديد من الصف الموجود.

سيرث الصف الجديدة كل قدرات الصف القديم، لكن يمكنها أيضاً أن تتضمن ميـزات جديدة خاصة به.



الشكل (3-4) مفهوم الوراثة

7- العلاقة بين لغة البرمجة C++ و لغة البرمجة C:

إن لغة البرمجة C++ مشتقة من لغة البرمجة C. إن كل عبارة صحيحة في لغة البرمجة C هي أيضاً عبارة صحيحة في لغـة البرمجـة C++، أن العكـس لـيس

صحيحاً. إن أهم العناصر التي تمت إضافتها إلى لغة البرمجة **C** لإنشاء لغة البرمجة **C++** لها علاقة بالصفوف والأغراض والبرمجة غرضية التوجه. لكن لغة البرمجة **C++** تتضمن الكثير من الميزات الأخرى أيضاً، بما في ذلك أسلوباً محسّناً لعمليات الدخل/الخرج **(I/O)** وطريقة جديدة لكتابة التعليقات. تضيف إلى لغة البرمجة **C** القدرة على تطبيق مبدأ البرمجة غرضية التوجه. كما أنها تضيف مجموعة جديدة من الميزات. وتعتبر لغة البرمجة **C++** من لغات البرمجة الكائنية Object Programming Language. وتعتمد الكائنات اعتماداً مباشرة على الصفوف Classes. وعلى البنى Structures.

8- التركيب النحوي لمواصفات الصف:

تتألف مواصفات الصف من الكلمة الأساسية **class** واسم الصف وقوس حاصر فتح وقوس حاصر إغلاق وفاصلة منقوطة:

```
class Study
{
.
.
<other program lines>
.
};
```

الأقواس الحاصرة هي محددات **delimiters** تحيط جسم مواصفات الصف لكي نتمكن (وكذلك المترجم في لغة البرمجة **C++**) من تحديد بداية الصف ونهايتها. وهي تفعل الشيء نفسه الذي تفعله الكلمات الأساسية **Begin** و **End** في لغة البرمجة باسكال. والأقواس الحاصرة هي المحددات القياسية في لغة البرمجة **C++**. تُنهي الفاصلة المنقوطة المواصفات بأكملها. يتمّ استعمال الفواصل المنقوطة لإنهاء عبارات البرنامج، وتعريفات المتغيرات والبيانات، ومواصفات الصفوف إذ يجب كتابة قوس حاصر غلق وفاصلة منقوطة معاً.

9- تعريف المتغيّرات:

تعريف متغيرين، على أنهما عددان صحيحان (النوع int)، إنه أحد أنواع المتغيرات (البيانات) المتوفرة في لغة البرمجة C++.

```
int studx
int study;
```

إنّ هذه التعريفات لا تعطي المتغيرات أي قيمة، وإنما تعطيها فقط اسماً وتحجز مساحة معينة في الذاكرة، رغم أنه لا يتمّ تخصيص مساحة الذاكرة إلا بعد إنشاء المتغير.

10- الدوال:

هناك ثلاث دوال في مواصفات الصف، حيث يتمّ إبلاغ المترجم في لغة البرمجة C++ إنّ هذه الأسماء هي لدوال بكتابة أقواس بعد الاسم:

```
viod student( )
{
<statements>
}
```

إنّ الكلمة الأساسية التي تسبق اسم الدالة تشير إلى نوع البيانات التي ستعيدها تلك الدالة. فإذا كانت تعيد أي قيمة، يتمّ استعمال الكلمة المحجوزة void لتعريف دالة تسمى دالة عقيمة. فإذا كانت تعيد عدداً صحيحاً عندئذ نكتب:

```
int sum1( )
{
<statements>
}
```

إضافة بارامترات parameters إلى الدالة من خلال وضع قيم أو أسماء متغيرات ضمن الأقواس. تنقل الوسطاء القيم إلى الدالة، كما يلي:

```
void funct (int x , int y )
```

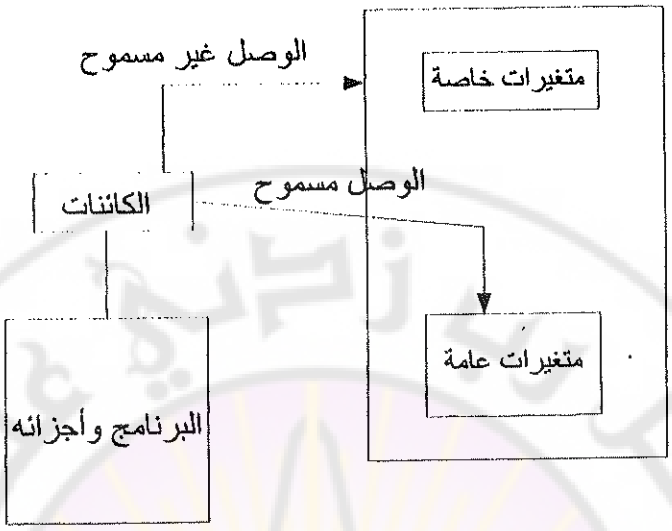
```
{
<statements>
}
```

يتمّ حصر جسم الدالة بأقواس حاصرة.

11- المتغيرات العامة والمتغيرات الخاصة:

إنّ المتغيرات العامة (**public**) والمتغيرات الخاصة (**private**) تمكن العبارات التي تقع خارج الكائن الوصول إلى بعض أجزائه، في حين تبقى إمكانية الوصول إلى الأجزاء الأخرى من ضمن الكائن نفسه، كما هو مبين في الشكل (3-5).

نستعمل الكلمتين الأساسيتين **public** و**private** ، تليهما نقطتان، للإشارة إلى هذه الأنواع من المتغيرات والدوال. في حالتنا هذه كل المتغيرات (البيانات) خاصة وكل الأعضاء الدالية عامة. إنها الحالة الاعتيادية: لإخفاء البيانات عن العالم الخارجي لكي تكون محمية من العبث نعرف المتغيرات (البيانات) كمتغيرات خاصة، لكن يجب أن تكون الأعضاء الدالية عامة لكي تتمكن الأجزاء الأخرى من البرنامج استدعاءها من أجل تنفيذ مهام برمجية ويمكن الوصول للبيانات، من خلال الأعضاء الدالية.



الشكل (3-5) الوصول للمتغيرات العامة والمتغيرات الخاصة

12- أنواع المتغيرات (البيانات) الأساسية في لغة البرمجة C++:

يتألف الصف من جزئين رئيسيين: متغيرات البيانات والأعضاء الدالية. فيما يلي نبين أنواع المتغيرات (البيانات) الأساسية في لغة البرمجة C++. ثم نبين الأعضاء الدالية بشكل مفصل.

تمكن لغة البرمجة C++ من تعريف أي نوع من أنواع متغيرات بيانات، لكن الأنواع مسبقة التعريف توفر العناء في أغلب الحالات البرمجية.

هناك سبعة أنواع من متغيرات البيانات في لغة البرمجة C++، واحد منها يمثل المحارف، وثلاثة تمثل الأعداد الصحيحة وثلاثة تمثل الأعداد الحقيقية (الأعداد العائمة). يلخص الجدول (3-1) أنواع متغيرات البيانات:

النوع	يُستعمل لتخزين	أمثلة عن القيم المخزنة
char	محرف	**'?', '3', '$', 'B', 'a'**
short	أعداد صحيحة قصيرة	7، 30.000، -222
int	أعداد صحيحة عادية الحجم (مثل **short** أو **long**)	
long	أعداد صحيحة طويلة	1000.000.000، -123.456.789
float	أعداد حقيقية قصيرة	3.7، 199.99، -16.2، 0.000125
double	أعداد حقيقية مزدوجة	0.048512934، -7.553.393.95.47
long double	أعداد حقيقية ضخمة	345.666، 012، 789، 456، 123، 9

الجدول (3-1) أنواع متغيرات البيانات في لغة البرمجة C++:

- المحارف:

يتمّ تخزين المحرف في متغيرات من النوع **char**. العبارة:

```
char ch;
```

ينشئ المترجم مساحة في الذاكرة لمحرف ويسميها **ch**، لتخزين محرف ما في المتغير، استعمل عبارة كما يلي:

```
ch = 'a';
```

الأحرف الثابتة، كـ **'a'** أو **'B'** أو **'&'** أو **'4'**، تكون محصورة بعلامات اقتباس فردية. يمكن استعمال المتغيرات ذات النوع **char** أيضاً من أجل تخزين أرقام صحيحة بدلاً من أحرف. مثلاً:

```
ch = 44;
```

لكن مجال القيم الرقمية التي يمكن تخزينها في النوع **char** تقع ضمن المجال [-128,127]. إن المتغير ذا النوع **char** يحتل **1 Bytes**، أو **8 Bits**، في الذاكرة.

13- معامل التعيين:

يؤدي معامل المساواة (=) إلى تعيين القيمة الموجودة على إليمين للمتغير الموجود على إليسار؛ تسمى علامة المساواة معامل تعيين **assignment operator**.

يتمّ تخزين كل المحارف كأرقام، ويستعمل جدول الآسكي **ASCII** لترجمة المحارف إلى أرقام. لذا الحرف **'A'** هو الرقم 65، و**'B'** هو 66، إلخ. وذلك حسب جدول الآسكي**ASCII** .

14- دوال التحكم:

يتمّ تمثيل عدة أحرف خاصة بحرف تسبقه شرطة مائلة (اشارة التقسيم). يسمى هذا الحرف دالة تحكم **escape sequence** لأن الشرطة المائلة تجعل تفسير الحرف الذي يلي "يهرب" من جدول الآسكي **ASCII** ويشير إلى شيء مختلف. يبين الجدول (3-2) بعض دوال التحكم:

دالة التحكم	الحرف المقصود
'\n'	سطر جديد. يجبر المؤشر على الانتقال إلى بداية السطر التالي
'\t'	حرف الجدولة **Tab**
'\b'	حرف التراجع **Backspace**
'\r'	حرف الإرجاع. يجبر المؤشر على الانتقال إلى بداية هذا السطر. يتمّ توليده أيضاً بضغط **Enter**.

الجدول (3-2) دوال التحكم (الهروب)

15- متغيرات الأعداد الصحيحة:

هناك ثلاثة أنواع من متغيرات الأعداد الصحيحة في لغة البرمجة C++: **short** (قصير) و**int** (عدد صحيح) و**long** (طويل). إنها متشابهة فيما بينها لكنها تحتل مساحات مختلفة في الذاكرة ويمكنها معالجة أرقام تقع في مجالات مختلفة، كما هو مبين في الجدول (3-3). لقد وضعنا النوع **char** رغم أنه يتمّ استعماله في أغلب الأحيان للمحارف – لأنه يمكن استعماله لتخزين أرقام صحيحة صغيرة أيضاً.

إن أنظمة التشغيل القديمة، كـ دوس وويندوز 3.1، هي أنظمة **Bits 16**. أما الأنظمة الحديثة، كويندوز 95، 2003 و**OS/2** وويندوز **NT**، فهي أنظمة **Bits 32**.

كما أن نظام إليونيكس نظام **Bits32**.

مثال:

تعريف بعض المتغيرات ذات النوع عدد صحيح واسناد قيماً لها:

```
int x;
long y;
```

اسناد قيم لهذه المتغيرات:

```
x=321
y=1506823L;
```

يتمّ استعمال الحرف **L** لتحديد ثابت من النوع **long**، أي ثابت لن يتسع في عدد صحيح (في الأنظمة **Bytes 16**).

النوع	الحجم	المجال
char	(Bits8) Bytes 1	[-128 , 127]
short	(Bits16) Bytes 2	[-32767 , 32768]
int	مثل short في الأنظمة التي تعمل بـ 16 Bits، ومثل long في الأنظمة 32 Bits	
long	(Bits 32) Bytes 4	[-2147483648 , 2147483647]

الجدول (3-3) أنواع متغيرات الأعداد الصحيحة في لغة البرمجة C++:

16- متغيرات الأعداد الصحيحة التي ليس لها إشارة:

إن كل أنواع متغيرات الأعداد الصحيحة لها إصدارات من دون إشارة **unsigned**. لا تستطيع المتغيرات التي ليس لها إشارة تخزين أرقام سالبة، لكن حجم مجال قيمها الموجبة يساوي ضعف حجم مجال التي لها إشارة. يبين الجدول (3-4) متغيرات الأعداد الصحيحة التي ليس لها إشارة.

تكون الأعداد الصحيحة الاعتيادية، من دون الميزة **unsigned**، لها إشارة بشكل افتراضي.

النوع	الحجم	المجال
unsigned char	(Bits8) Bytes 1	[0 , 255]
unsigned short	(Bits16) Bytes 2	[0 , 65535]
Unsigned int أو Unsigned	مثل unsigned short في الأنظمة 16 Bits مثل Unsigned long في الأنظمة 32 Bits	
Unsigned long	(Bits 32) Bytes 4	[0 , 4294967295]

الجدول (3-4) أعداد صحيحة ليس لها إشارة:

17- متغيرات (الاعداد الحقيقية) الأرقام العائمة:

يتمّ تمثيل الأرقام العائمة عادة برقم صحيح على إليسار مع نقطة عشرية أو كسر على إليمين. بدلاً من النقطة العشرية، يمكن استعمال الآسي **exponent**. لذا القيمة 124.65 في الشكل العادية هي **1.2465e2** في الشكل الآسي، حيث يشير الرقم الذي يلي الحرف e إلى كم مرة يجب نقل النقطة العشرية إلى إليمين لاسترجاع القيمة إلى الشكل العادي. غالباً ما يستعمل الشكل الآسي لعرض الأرقام الطويلة في الشكل العشري. لذا **9,876,000,000,000,000,000** في الشكل العادي هي في الشكل الآسي كما يلي : **9.876e18**

هناك ثلاثة أنواع من متغيرات الأرقام العائمة في أنظمة التشغيل ، كما هو مبين في الجدول (3-5). (بعض الأنظمة لا تملك النوع **long double**).

اسم النوع	الحجم	المجال	الدقة
float	**(Bits 32) Bytes 4**	**[10e-38 , 10e38]**	5 أعداد
double	**(Bits 64) Bytes 8**	**[10e-308 , 10e308]**	15 عدداً
long double	**(Bits 80) Bytes 10**	**[10e-4932 , 10e4932]**	19 عدداً

الجدول (3-5) الأرقام العائمة

إن أشهر نوع من متغيرات الأرقام العائمة هو النوع **double**، الذي يستعمل في معظم الدوال المكتبية الرياضية في لغة البرمجة C++. يتطلب النوع **float** ذاكرة أقل من النوع **double** وقد يسرّع إنجاز العمليات الحسابية. ويستعمل النوع **long double** في معالج (processor) الأرقام الكبيرة وهو من معالجات إنتل (Intel).

مثال:

تعريف واستعمال متغيرات من أنواع الأرقام العائمة المختلفة.

```
float pi-float;
double pi-double;
long double pi-long-double;
```

لقد عرفنا ثلاثة أنوع من التغيرات وذلك حسب رغبة المبرمج.

18- المسافات الفارغة:

تمكن لغة البرمجة C++ من وضع مسافات فارغة إضافية في أسطر البرنامج لتسهيل القراءة والكتابة.

```
int x       =12;
float y     =3.14;
```

يمكن وضع قدر ما نشاء من المسافات وعلامات الجدولة والسطور الجديدة في البرنامج. تؤلف هذه الأحرف المسافة البيضاء (الفارغة) **white space**، ويتجاهلها المترجم في لغة البرمجة C++. يستعمل المبرمجون المسافة الفارغة لكي يكون البرنامج أسهل للقراءة والفهم.

19- التعليقات:

يمكن وضع تعليقات في البرنامج لتسهيل فهم البرنامج وسهولة صيانته وتصحيحه. وذلك من خلال وضع إشارة مزدوجة (//) قبل التعليقات.

مثال:

نستعمل تعليقاً في كامل السطر أوتعليقات تلي كل عبارة:

```
// My first program.
```

أي نص يلي الرمز // إلى نهاية السطر يتجاهله المترجم.

هناك نوع آخر من التعليقات يُتيح كتابة تعليقات على عدة أسطر.

مثال:

```
/*
I am student.
I write my program.
*/
```

يبدأ التعليق بالرمز */ وينهيه الرمز /*. فبذلك نتمكن من كتابة التعليق على عدة أسطر قبل الإنهاء بالرمز /*.

20- دالة الدخل / الخرج:

ـ دالة الخرج:

عبارة إظهار النص على الشاشة:

```
cout<< "The text on the screen";
```

الدالة **cout**، والمعامل >>، المسمى معامل إخراج (الوضع) **put to operator**، يمكن من إظهار سلسلة محارف، أو محتويات المتغيرات. إن الدالة **cout** والمعامل >> معرفان في مكتبة الدفق الياسية في لغة البرمجة C++، حيث يجب إدراج ملف الترويسة من أجل تعريفها.

- ثوابت متسلسلة محارف:

إن النص **"This text will appear on the screen"** يسمى ثابت متسلسلة محارف **string constant**. ثابت متسلسلة محارف تحيطه علامات اقتباس مزدوجة. وتنتهي العبارة بأكملها بفاصلة منقوطة.

يمكن من إخرج الثوابت الرقمية بالطريقة نفسها:

```
Cout << 27;
```

يظهر على الشاشة الرقم 27، بينما العبارة:

```
Cout << 123. 45;
```

فتعرض الرقم 123.45

تمكن من إخراج قيم المتغيرات والثوابت. الشيفرة التالية مثلاً تعرض 2.7:

```
floate x-float=2.7;
cout<<x-float;
```

ويمكن عرض أكثر من قيمة واحدة في عبارة **cout** ، باستعمال معاملات

إخراج:

```
cout<<" The value is"<<x-float;
```

تؤدي هذه الشيفرة إلى عرض الخرج التالي:

The value is 2.7.

- تنسيق الخرج:

يمكن التحكم بطريقة تنسيق الخرج. هناك عدة أساليب مختلفة في عمليات الدخل / الخرج في لغة البرمجة **C++**.

إن دالة الخرج **cout** أو معامل الخرج لا تنشئ أسطراً جديدة.

مثال:

```
cout<<342;
cout<<12<<45;
```

حيث تنتج الخرج

3421245

حيث يلتصق كل الخرج ببعضه بعضاً.

ويمكن أن نميز طريقتين للتحكم بالخرج:

- طرق التحكم بالخرج

أ- دوال التحكم:

من الطرق السهلة لتنسيق البيانات إدراج دالة تحكم في ثابت متسلسلة المحارف. نستعمل دالة التحكم **n** لبدء سطر جديد:

```
cout<<"\n Enter your name:";
cout<<"\n Enter your age:";
```

نحصل على الخرج:

Enter your name
Enter your age:

الدالة \n الموجود قبل الكلمة **First** تُجبر ثابت متسلسلة المحارف الثاني على الظهور على سطرجديد. الدالة \n الموجودة قبل الكلمة **Now** تضمن بدء السطر الأول على سطر جديد أيضاً.

يمكن من استعمال دالة التحطم \t إنتقال المشيرة الضوئية /8/ محارف فارغة (جدولة). الكما يلي:

```
cout<<"\n Ahmad \t Ali \t Ziad";
```

تمكن من إخراج الأسماء في أعمدة

Ahmad Ali Ziad

ب – دالة التحكم (المُناور) endl:

لبدء أسطر جديدة في لغة البرمجة C++. يمكن إدراج كائن مُناور **manipulator** في دفق الخرج، كأنه عنصر بيانات. يمكن استعمال الدالة **endl** لتنسيق الخرج.إن الدالة **endl** هي اختصار لـ **end line** أي إنهاء السطر. عند إدراج الدالة في عبارة **cout**، تُجبر هذا الدالة المؤشر على الانتقال إلى بداية السطر التالي، تماماً كما تفعل الدالة \n.

مثال:

```
cout<<endl;
cout<<"Enter your name:"<<endl;
cout<<" Enter your age:";
```

تضمن العبارة الأولى البدأ على سطر جديد، الدالة **endl** الموجود في نهاية العبارة الثانية تجبر النص الذي يبدء بالكلمة **First** أن يظهر على سطر جديد.

نحصل على الخرج:

Enter your name
Enter your age:

– دالة الدخل من لوحة المفاتيح:

إن الدالة **cin** ومعامل الإدخال (الحصول) **get from** (>>) تمكن من اسناد القيم

الموجودة على يساره ويخزنها في المتغير الموجود على يمينه. عند تنفيذ هذه العبارة، ينتظر البرنامج أن يكتب المستخدم رقماً ويضغط **Enter**.

مثال:

نبين إدخال رقم من قبل المستخدم من لوحة المفاتيح، وتخزينه في متغير يدعى **age**:

يمكن عرض رسالة قبل انتظار الدخل من المستخدم:

```
int age;
cout<<"Enter your age:";
cin>>age;
```

ويكون الخرج وفق التفاعل التالي:

```
Enter your age: 20
```

حيث يكتب المستخدم 20.

تمكن من استعمال معامل الدخل عدة مرات في العبارة نفسها:

```
int x-int;
float y-float;
cin>>x-int>>y-float;
```

بضغط المفتاح **Enter** أو مفتاح المسافة أو **Tab** بعد كل قيمة قبل أن نكتب القيمة التالية.

21- دفق الدخل / الخرج:

إن أساليب الدخل والخرج التي عرضناها تسمى دفق دخل / خرج **stream I/O**. الدفق هو مصطلح عام لدفق من البيانات **flow of data**. لكي نستعمل دفق الدخل / الخرج نحتاج إلى الملف **iostream.h** من التعريفات في البرنامج. ، يدعى هذا الملف ملف ترويسة **(header)** وهذا الملف موجود ضمن المجلد (الشمل) **(include)**.

22- دخل / خرج لغة البرمجة C:

دالة الدخل / الخرج في لغة البرمجة C، **printf()** و**scanf()** وغيرها من دوال المكتبات المماثلة (المعرّفة في الملف **stdio.h**). يمكن استخدام هذه الدوال في لغة البرمجة C++.

23- لأعضاء الدالية:

يستدعي البرنامج الأعضاء الدالية لغرض ما لإبلاغ ذلك الغرض ما عليه فعله.

```
void student-data( )
{ int age;
cout<<"Enter your age:";
cin>>age; }
```

فيكون الخرج التالي:

```
Enter your age: 20
```

24- المعاملات الحسابية:

تتضمن لغة البرمجة C++ المعاملات الحسابية الأربعة بالإضافة إلي معامل باقي القسمة. كما هو مبين في الجدول (3-6).

المعامل	وظيفته
+	جمع
-	طرح
*	ضرب
/ و div	قسمة
%	الباقي

الجدول (3-6) المعاملات الحسابية

المعاملات الحسابية الأربعة الأولى تنجز العمليات الحسابية. أما معامل باقي القسمة % (المسمى أيضاً المُعامِل **modulus**) فيتمّ استعماله لاحتساب باقي قسمة عدد صحيح على عدد صحيح آخر. مثلا التعبير 3% 20 يساوي 2، لأن 20 مقسومة على 3 هي 6 ويبقى 2. تستعمل المعاملات الحسابية بشكل مشابه لاستعمالها في لغات البرمجة الإجرائية الأخرى. وهي تسمى معاملات ثنائية **binary operators** لأنها تعمل على كميتين. فبذلك يمكن أن نكتب التعابير الرياضية بالطريقة التي نريدها.

مثال:

```
C = (f-32)* 5 / 9;
```

لتحول درجة الحرارة من درجة مئوية إلى درجة فهرنهايت. نستعمل الأقواس لكي يتمّ تنفيذ الطرح أولاً، بالرغم من أولويته المتدنية (يشير المصطلح أولوية **precedence** إلى ترتب تنفيذ المعاملات. المعاملان * و / لهما أولوية أعلى من أولوية + و -).

25- معامل التزايد والتناقص:

تمكن لغة البرمجة C++ من استخدام معاملين خاصين ينفذان زيادة 1 إلى المتغير أو طرح 1 من المتغير. يضيف معامل التزايد **increment** 1 إلى المتغير ويطرح معامل التناقص **decrement** 1 من المتغير. تمكن دوال لغة البرمجة C++ من تعريف معاملات الزيادة ومعاملات النقصان وفق ما يلي:

- عامل الزيادة :

ويرمز له بالرمز ++ وباستخدام هذا المعامل فإن المترجم يزيد قيمة المتغير بمقدار (1) ونميز حالتين:

- الزيادة على المتغير قبل التنفيذ

```
int i;
++i;
```

- الزيادة على المتغير بعد التنفيذ

```
int i;
i++;
```

في هذه الحالة يتمّ تنفيذ العملية المخصصة ثم يقوم بالزيادة

ويمكن أيضاً زيادة قيمة المتغير بالقيمة التي نراها مناسبة

```
int i;
i=i+5;
```

أو `i += 5;`

- معامل النقصان:

ويرمز له -- تمكن لغة الـ C++ من إنقاص قيمة المتغير وفق إحدى الطريقتين

- إنقاص قبل التنفيذ

```
int I;
--I;
```

- إنقاص بعد التنفيذ

```
int j;
j--;
```

ملاحظة:

تمكن لغة البرمجة C++ من تنفيذ العمليات الحسابية على المتغير بأحدى الطريقتين.

```
int k;
k=k+3;      k+=3
k=k-5       k-=5
k=k*2       k*=2
k=k/4       k/=4
```

مثال:

نبين استعمال معامل التناقص:

```
int x=30;
int y=59;
--x;
--y;
```

يتألف معامل التناقص من علامتي طرح: --. إذا كان المتغير يساوي 30 والمتغير 59، فتصبح قيمتهما 29 و58 بعد تنفيذ هذه الدالة.

بشكل مماثل، يؤدي معامل التزايد إلى زيادة قيمة المتغير بـ 1. يسمى معاملا التزايد والتناقص معاملان أحاديان **unary operators** لأنهما يعملان على متغير واحد فقط، ولهما أولوية أعلى من أولوية المعاملات الحسابية.

مثال:

في التعبير تتمّ زيادة قيمة المتغير قبل تنفيذ عملية الجمع.

```
int x=30;
int y=59;
++x;
++y;
```

26- تحديد مواصفات الصف:

يتكون الصف من الكلمة الأساسية **class** والقوس الحاصرة { } وجزئين: الجزء الأول يتضمن الأعضاء البيانية والجزء الثاني الأعضاء الدالية.

نعرض المواصفات الكاملة للصف:

```
class Study
{
private:
    <variables>;
public:
    <function>;
```

```
void student-data( )
{
int age;
cout<<"Enter your age:";
cin>>age;
} };
```

27- إنشاء الكائنات:

إن الهدف الأساسي من مواصفات الصف هو استعمالها كأساس لإنشاء الكائنات.

- إنشاء الكائنات من مواصفات الصف:

تنشأ الكائنات باستعمال التركيب النحوي نفسه الذي استعمل لإنشاء متغير من النوع أساسي كـ **int** أو **float**. فالكائنات في لغة البرمجة C++ نتعامل معها بشكل مشابه للتعامل مع المتغيرات، كما تتمّ معاملة الصفوف كأنواع بيانات.

مثال:

فيما يلي ننشئ كائناً يدعى **obj1** دالة للصف**Study**:

```
Study obj1;
```

يجد برنامج مواصفات الصف **Study** ، ثم يحتسب حجم الكائن ويخصص مساحة من الذاكرة. ثم يعطي مساحة الذاكرة تلك اسماً:**obj1** . هذا تماماً ما سيفعله البرنامج مع متغير من نوع أساسي. لكن **obj1** معقد أكثر كونه يتضمن عدة أجزاء من البيانات وعدة أعضاء دالية.

يمكن إنشاء أكثر من كائن:

```
Study obj1;
Study obj2;
```

يمكن استعمال عبارة واحدة لإنشاء عدة كائنات من النوع نفسه.

```
Study obj1,obj2;
```

أن تحديد مواصفات صف قد تبدو صعبة أو غير مألوفة، إلا أن إنشاء الكائنات

على أساس تلك المواصفات بسيط.

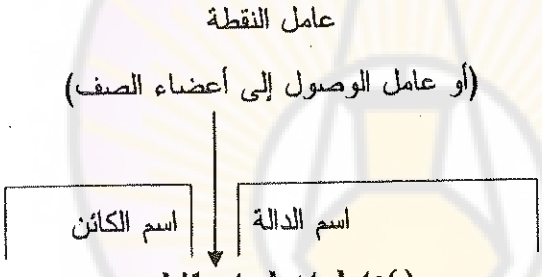
28- النفاذ إلى الصف:

بعد إنشاء الكائن يمكن النفاذ إلى الأعضاء الدالية، أي الدوال المحددة في مواصفات الصف. ومن أجل النفاذ إلى الصف نحتاج إلى تركيب نحوي مؤلف من قسمين: اسم الكائن واسم العضو الدالي تفصل بينهما نقطة.

فيما يلي نبين كيفية استخدام الكائن **obj1** ليخرج بياناته بواسطة الدالة **:student-data( )**

```
obj1. student-data( ) ;
```

يربط اسم الكائن واسم الدالة بواسطة رمز نقطة (.)، وهو يسمى معامل النقطة **dot**. يبين الشكل (3-6) كيف نستخدم هذا المعامل.



الشكل (3-6) التركيب النحوي لاستخدام الكائن

عند تنفيذ هذه العبارة، سيخرج الكائن **obj1** بياناته.

مهما تكن القيم المخزنة في متغيرات البيانات. ويمكن استخدام كائن آخر، **obj2** مثلاً. سيعرض عندها ذلك الكائن بياناته وهي على الأرجح ليست نفسها المخزنة في الكائن **obj1**.


وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل الرابع

كتابة برامج غرضية التوجه

(البرمجة بلغة البرمجة C++)

1- مقدمة:

سنعرض برنامج يمكن تنفيذه، حيث نستخدم التعليمات الأساسية في لغة البرمجة C++. ثم، باستعمال الحلقات والقرارات، وهي عبارة التحكم الأساسية في لغة البرمجة C++ تمكن من فعل شيء أكثر من مرة واحدة.

- برنامج 1

فيما يلي نعرض البرنامج **Mat-Prog**

```
// Mat-Prog.ccp
# include <iostream.h>
class test-oper
{
 private:
   int x;
   int y;
 pubic:
 void display( )
  {
     cout<<"\n x ="<<x;
     cout<<"\n y ="<<y;
  }
 void increm-data( )
  {
    ++x;
    ++y;
```

```
    }
  void initdata( )
   {
 cout<<"Enter the value x = ";
 cin>>x;
 cout<<"Enter the value y = ";
 cin>>y;
    }
};
void main ( )
{
 test-oper obj1;
 test-oper obj2;
cout<<"\n Initialize Data for obj1";
obj1. initdata( );
cout<<"\n Initialize Data for obj2";
obj2. initdata( );
cout<<"\n Increment 2 to obj1";
obj1. increm-data( );
obj1. increm-data( );
cout<<"\n Increment to obj2";
obj2. increm-data( );
cout<<endl;
cout<<"\n Supplies on obj1";
obj1. display( );
cout<<"\n Supplies on obj2";
obj2. display( );
}
```

2- ملف الترويسة iostream.h:

يستخدم ملف الترويسة iostream.h لدفق الدخل / الخرج، حيث خزن في هذا الملف موصفات الصفوف المختلفة التي تمّ اشتقاق الكائنين cin و cout والمعاملين >>

و<< والدالة **endl** ، بالإضافة إلى التعريفات المختلفة الأخرى، يتمّ تزويد ملف **iostream.h** مع المترجم. إنه ملف نصي، كالملفات **cpp**. التي نكتبها.

لإدراج الملف **iostream.h** في الملف المصدر، نضع سطراً من النص مُرشد ما قبل المعالج **preprocessor directive** في شيفرة البرنامج. ونكتب المرشد وفق مايلي:

#include <iostream.h>

يُدرج هذا المرشد كل النص الموجود في الملف **iostream.h** في الملف المصدر أي يعتبره المترجم مُدرجاً في الملف المصدر، إن ملف المصدر لايتغير.

3- المُرشدات ما قبل المعالج:

المرشدات ما قبل المعالج هي تعليمات للمعالج يترجمها المترجم إلى لغة آلة يستطيع فهمها. تبدأ المرشدات ما قبل المعالج بالعلامة #. عندما يجد المترجم المرشد ما قبل المعالج **include #** ، يبحث عن الملف **iostream.h**. تخزين هذه ملفات في الدليل الفرعي الخاص ، يسمى **include**. يوجد هذا الدليل الفرعي في الدليل العام للمترجم الذي نستخدمه. يعرف المترجم مكان وجود هذا الدليل؛ فإذا تغير مكان هذا الدليل نحدد للمترجم مسار مكان وجود هذا الدليل. بعد أن يجد المترجم الملف، يقوم المترجم بإدراج ذلك النص في الملف المصدر محل المرشد **include #**.

إن محتويات الملف **iostream.h** غير مفهومة ، إلا إن **iostream.h** هو ملف نصي، مكتوب بمحارف جدول الآسكي **ASCII** .

نستخدم تنسيقاً محدداً لتحديد اسم الملف المرشد **include #**. إن استعمال التنسيق المناسب يسرع عملية البحث عن الملف. نستخدم أقواس زاوية لحصر اسم الملف:

 # include <filename.ext>

يؤدي هذا إلى جعل عملية البحث عن الملف تبدأ في الدليل **/include/..** القياسي. لكن إذا استخدمنا علامات اقتباس:

 # include "filename.ext"

فإن المترجم يبداء عملية البحث من الدليل حيث يتمّ تخزين الملفات المصدر الخاصة بالبرنامج.

4- ملفات ترويسة أخرى:

هناك الكثير من ملفات الترويسات. بعضها يُستخدم مع مكتبة دوال لغة البرمجة C. لاستخدام دوال رياضية مثلاً **sin()** و**cos()**،نحتاج إلى شمل ملف تروسية يسمى **math.h**؛ ومن أجل استخدام سلاسل المحارف، نحتاج إلى شمل ملف تروسية يسمى **string.h**؛ ومن أجل إجراء بعض أنواع عمليات تحويل البيانات، نحتاج إلى شمل ملف تروسية يسمى **stdlib.h**. قد تتطلب مختلف دوال الدخل/الخرج شمل ملف تروسية **conio.h** أو شمل ملف تروسية **stdio.h**.

نحتاج إلى ملفات ترويسة أخرى لأنواع مواصفات الصفوف الأخرى. قد يتضمن المترجم مكتبة صفوف حاوية ، كالمصفوفة أو المكدس أو صف الانتظار أو اللائحة المرتبطة. تحتوي مكتبة الصفوف الحاوية على صفوف تصوغ بنيات البيانات هذه، ومن أجل استخدام واحدة، نحتاج إلى شمل ملف ترويسة خاص بتلك الحاوية، كـ **arrays.h** أو **stacks.h** أو **queus.h**.

5- دالة مسح الشاشة:

تمكن الدالة **clrscr( )** من مسح الشاشة التنفيذية ونقل المؤشرة الضوئية إلى الزاوية اليسارية العليا.ولكي نستطيع تنفيذ هذه الدالة نحتاج إلى الترويسة التالية:

ترويسة التحكم بالمخرجات

```
#include<conio.h>
```

ملاحظة:

لا توجد أهمية لترتيب الترويسات في لغة البرمجة C++ وفيما يلي نعرض بنية البرنامج بلغة البرمجة C++ بشكل عام.

```
#include <iostream.h>
```

```
#include <conio.h>
void main()
{
   clrscr();
   N
}
```

6- الدالة الرئيسة ()main:

تتضمن لغة البرمجة C++ دالة رئيسة تسمى **()main**. إنها ليست عضواً دالياً في صف ما، بل هي دالة مستقلة ينقل نظام التشغيل التحكم إليها. العبارة الأولى في **()main**، مهما كانت، هي العبارة الأولى التي سيتمّ تنفيذها.

بنية الدالة **()main** فارغة:

```
void main()
{
}
```

يجب أن يحتوي كل برنامج على دالة**()main**، وإلا سنحصل على رسائل خطأ من الرابط **(linker)** عندما تحاول ترجمة البرنامج وربطه. لقد أظهرت الدالة **()main** مع نوع إعادة عقيم **(void)**. يمكن من استخدام نوع إعادة عدد صحيح **int**

مثال:

اكتب برنامجاً بغلة C++ يمكن من طباعة العبارتين

```
This is my program.
My name is Ahmad.
```

وذلك في سطرين مختلفين متتالين

```
#include <iostream.h>
#include <conio.h>
void main ()
{
```

```
    clrscr();
    cout << "\n this is my program\n";
    cout << "My name is Ahmad";

}
```

7- تنظيم البرنامج:

يرتب ملف المصدر بحيث تأتي المرشدات ما قبل المعالج **# include** أولاً، تليها مواصفات الصفوف، ثم الدالة الرئيسة **main( )**. وتحتوي الدالة الرئيسة **main( )** على العبارات التي تعرّف الكائنات وتفاعل مع الكائنات والدوال.

الشكل (4-1) تنظيم الملف المصدر

الخرج في البرنامج 1 :

```
Initialize Data for obj1
Enter the value x =45
Initialize Data for obj2
Enter the value y = 33
Increment 2 to obj
Increment to obj2
```

Supplies on obj1
x = 47
y= 35
Supplies on obj2
x = 46
y= 34

7- المعاملات العلائقية:

تقارن المعاملات العلائقية قيمتين وتؤدي إلى نتيجة صح / خطأ وفيما إذا كانت المقارنة صح أو خطأ. وتتضمن لغة البرمجة C++ ست معاملات علائقية.

الرمز	المعنى	مثال
==	يساوي	a == b
=!	لا يساوي	a! = b
<	أصغر من	a < b
>	أكبر من	a > b
<=	أصغر من أو يساوي	a <= b
>=	أكبر من أو يساوي	a >= b

الجدول (4-1) المعاملات العلائقية

تمكن المعاملات المعرفة في لغة البرمجة C++ من مقارنة المحارف والأرقام، لأن المحارف لها قيم في جدول الآسكي ASCII رقمية متوافقة معها. لذا فالتعبير 'a' < 'b' والتعبير 'A' == 65 صحيحان، على عكس التعبير 'a' <= 'z' (لأن المحرف z له في الواقع قيمة آسكي ASCII أكبر من قيمة المحرف a).

8- الحلقات Loops:

تعتبر الحلقات من أدوات التحكم البرمجية الهامة في لغة البرمجة C++ وفيما يلي نعرض الحلقة **while** والحلقة **do...while** والحلقة **for**. تمكن الحلقات من تكرار مهمة برمجية محدده وذلك حسب تحقق الشرط الذي يحدده المبرمج.

تأخذ الحلقات قراراتها على أساس قيم يمكن أن تكون إما صح أو خطأ. القيمة 0 (صفر) في لغة البرمجة C++ تعتبر خطأ وأي قيمة أخرى تعتبر صحيحة. إن الثابت 0 هو خطأ بشكل افترضي، لكن الثابت 1 صح.

تختبر الحلقات العلاقة بين متغيرين، أو بين متغير وثابت،وتكون نتيجة الاختبار إما صح أو خطأ. مثلا، إذا كان المتغير **j** أكبر من 0 تنفذ الحلقة أي تكرار مقطع برمجي؛ أو إذا كان المتغير **ch** لا يساوي 'x' تنفذ الحلقة. إن معرفة حالة التساوي أو الأكبر من أو غيرها تتمّ بواسطة المعاملات العلائقية **(relational)** في لغة البرمجة C++.

أ - الحلقة while:

تمكن الحلقة **while** من تكرار مهمة برمجية إلى أن يتغيّر شرط ما. هذا الشرط هو شيء يمكن التعبير عنه بقيمة صح / خطأ. تتألف الحلقة **while** من الكلمة الأساسية **while** يليها تعبير شرط محصور بين قوسين. ويكون جسم الحلقة محصوراً بين أقواس حاصرة (لكن من دون فاصلة منقوطة)، تماماً كالدالة. إذا كان جسم الحلقة يتألف من عبارة واحدة فقط، لا نحتاج إلى الأقواس الحاصرة.

```
while (n<50)
  n= n*2
```

ستستمر هذه الحلقة في مضاعفة قيمة المتغير **n** إلى أن تعود هذه القيمة أصغر من (أي عندما تصبح أكبر من أو تساوي) 50؛ عندها تتوقف الحلقة.

مثال:

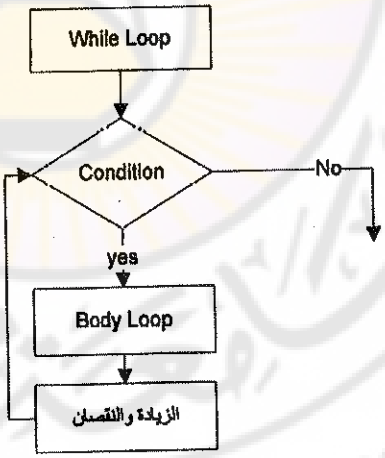
نعرض مثالاً يبين استخدام الحلقة **while** . تواصل الحلقة **while** الطلب من

المستخدم إدخال محرف. وستستمر الحلقة في التكرار إلى أن يكتب المستخدم الحرف q (اختصار quit إنهاء):

```
while (ch !='q')
  {
    cout<<"My program ";
    cout<<"Enter a character :";
    cin>>ch;
    }
```

إذا لم يكتب المستخدم الحرف q ستستمر الحلقة في التكرار. ويكون الخرج كما يلي:

```
My program
Enter a character :r
My program
Enter a character :h
My program
Enter a character :q
```



الشكل (4-2) المخطط التدفقي لحلقة while

- التركيب النحوي لحلقة while

نميز حالتين

- الحلقة تحوي عبارة واحدة

```
while (condition)
Statement;
```

- الحلقة تحوي أكثر من عبارة برمجية

```
while ( Condition)
{
   statement 1;
   statement 2;
   .
   .
}
```

ملاحظة:

يتمّ اختبار الشرط قبل تنفيذ جسم الحلقة. إذا كان الشرط غير محقق (خطأ) عند دخول الحلقة، فإن الحلقة لن تنفذ.

مثال:

اكتب برنامجاً بلغة C++ يمكن من إيجاد مجموعة حدود سلسلة منشور دالة الـ cos وذلك وفق ماكلوران. على أن يكون الحد الأخير في هذه السلسة قيمة أصغر من القيمة التالية **0.002**

الحل:

لنوجد أولاً منشور الدالة cos

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \Lambda\ (-1)^i \frac{x^{2i}}{(2i)!}$$

ولنوجد علاقة الحصول على كل الحدود

$$u_i = (-1)^i \frac{x^{2i}}{(2i)!}$$

$$u_{i+1} = (-1)^{i+1} \frac{x^{2i+2}}{(2i+2)!}$$

$$\frac{u_{i+1}}{u_i} = (-1)^{i+1} \frac{x^{2i+2}}{(2i+2)!} \times \frac{(2i)!}{(-1)^i x^{2i}}$$

$$u_{i+1} = -u_i \frac{x^2}{(2i+2)(2i+1)}$$

البرنامج:

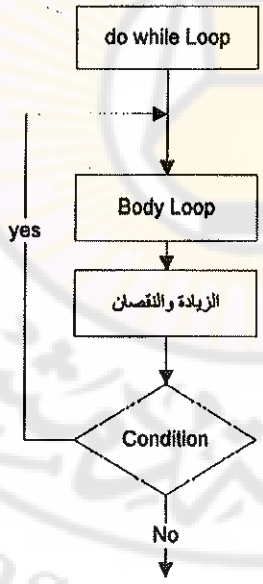
```
#include <iostream.h>
#include <conio.h>
void main ()
{
        clrscr();
        int x;
        cout << "enter x=";
        cin >> x;
        float s,u,e=0.002;
        int i=1;
        u=0.1;
        s=0.0;
        while(u>e)
        {
                u=-(u*x*x)/((2*i+2)*(2*i+1));
                s +=u;
                i+=1;

        }
        cout << "s="<<s;
        getch();
}
```

ب – حلقة do.....while:

تعمل الحلقة **do.....while** بشكل مشابه للحلقة **while** إلا إن هذه الحلقة تختبر الشرط بعد تنفيذ جسم الحلقة. هذا الأمر جيد عندما نريد تنفيذ الحلقة مرة واحدة على الأقل، بغض النظر عن تحق الشرط أوعدم تحقق الشرط (صح / خطأ)، الأولية ليس للشرط. تبدأ الحلقة **do** بالكلمة الأساسية **do** يليها جسم الحلقة بين أقواس حاصرة، ثم الكلمة الأساسية **while**، ثم تعبير الشرط بين قوسين، ثم فاصلة منقوطة. إن الحلقة **do** هي الحلقة الوحيدة التي تنتهي بفاصلة منقوطة. الفاصلة المنقوطة ضرورية لأن تعبير الشرط يلي جسم الحلقة، لذا فالأقواس الحاصرة لا تستطيع التصرف كمحدد للحلقة بأكملها.

مثال:

نبين فيما يلي حلقة **do.....while**، حيث تواصل هذه الحلقة جمع رقمين يدخلهما المستخدم. وعندما يُدخل الرقم 0 للرقم الأول تتوقف الحلقة.



الشكل (4–3) المخطط التدفقي للحلقة do.....while

- التركيب النحوي لحلقة do... while:

نميز حالتين

- الحلقة تحوي عبارة برمجية واحدة

```
do
statement
while (condition);
```

- الحلقة تحوي أكثر من عبارة برمجية

```
do
{
   statement 1;
   .
   .
   statement n;
} while (condition);
```

مثال:

اكتب برنامجاً بلغة الـ C++ يمكن مما يلي:

إدخال عددين صحيحين وإيجاد حاصل الجمع وحاصل القسمة علماً أن هذا البرنامج يكرر نفسه ما دام المستخدم يرغب بذلك.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
      clrscr();
      char ch;
      do
      {
            float x,y,z;
            float d;
            cout << "Enter x=";
```

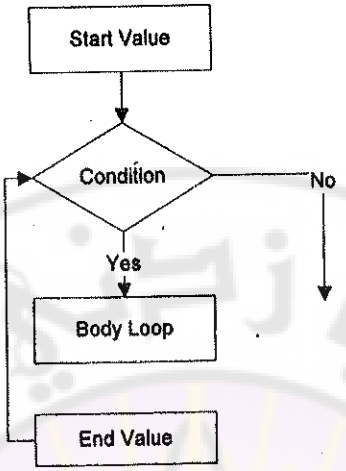
```
            cin >> x;
            cout << "Enter y=";
            cin >> y;
            z=x+y;
            cout << x << "+" << y << "=" << z;
            cout << "\n\n";
            cout << "The result of devsion";
            if(y == 0)
                  cout <<"worning";
            if(y != 0)
            {
                  d = x / y;
                  cout << x << "/" << y << "=" << d;
            }
            cout << "want again\n";
            cout << "yes (y) or No (n)\n";
            cin >> ch;
      } while(ch == 'y' || ch == 'Y');
      getch();
}
```

ج- الحلقة for:

لا نعرف عدد تكرارات في الحلقات while أوالحلقةdo... while . يظهر الشرط الذي يوقف الحلقة ضمن الحلقة. هذه ليست حال الحلقة for.

تحتوي الحلقة for الأقواس التي تلي الكلمة الأساسية for على ثلاثة تعابير مختلفة، تفصلها فاصلة منقوطة. تعمل هذه التعابير الثلاثة، على متغير يسمى متغير الحلقة. هذه التعابير هي:

- تعبير التمهيد، الذي يمهد قيمة متغير الحلقة.
- تعبير الشرط، الذي يختبر قيمة متغير الحلقة ليحدد ما إذا كان يجب تكرار الحلقة مرة أخرى أو إيقافها.
- تعبير التزايد، الذي يقوم عادة بزيادة (أو إنقاص) قيمة متغير الحلقة.



الشكل (4-4) المخطط التدفقي لحلقة for

- التركيب النحوي للحلقة for:

نميز حالتين :

- الحلقة تحتوي على عبارة برمجية واحدة فقط على الشكل التالي:

```
for(start value; condition; Increment or Decrement)
    Start ment;
```

- الحلقة تحوي على أكثر من عبارة برمجية على الشكل التالي:

```
for(start value; condition; Increment or Decrement)
{
      start ment 1;
      start ment 2;
      .
      .
}
```

مثال:

تحتوي الحلقة for على عدة عبارات في جسم الحلقة. كما الحال مع الحالات

الأخرى، يجب حصر العبارات المتعددة بأقواس حاصرة.

```
int j;
int total =0;
for (j=0; j<10; ++j)
{
total = total + j;
cout<< total << " ";
}
```

خرج هذه الحلقة تسلسل الأرقام

0 1 3 6 10 15 21 28 36 45

وتمكن حلقة **for** من إنقاص قيمة المتغير بـــ 1 كلما تكررت الحلقة:

```
for (j=10; j>0; --j)
cout<< j << " ";
```

خرج هذه الحلقة

10 9 8 7 6 5 4 3 2 1

أو تمكن من زياده أو إنقاص قيمة المتغير بكمية أخرى. هذه الشيفرة:

```
for (j=0; j<100; j=j+10)
cout<< j<< " ";
```

خرج هذه الحلقة

0 10 20 30 40 50 60 70 80 90

ملاحظة:

هناك درجة من المرونة بشأن ما تضعه في التعابير الثلاثة في الحلقة **for**. تمكن من استعمال عدة عبارات مثلاً في تعبير إسناد قيمة إبتدائية وتعبير الشرط.

مثال:

يمكن لحلقة **for** من استعمال عدة عبارات:

```
for (j=0; total=0; j<10; ++j; total = total + j )
  cout<< total << " ";
```

خرج هذه الحلقة

0 1 3 6 10 15 21 28 36 45

يمكن ضبط المتغير **total** عند القيمة 0 في تعبير الأسناد بدلاً من تنفيذ ذلك في جسم الحلقة، وتتمّ زيادته بـــ **j** في تعبير التزايد بدلاً من أن تتمّ زياده في جسم الحلقة. يتمّ فصل العبارات عن بعضها بعضاً في هذه التعابير بفواصل.

مثال:

اكتب برنامجاً بلغة الـــ C++ يمكن من تكرار عبارة **5 my name is ahmad** مرات.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
        clrscr();
        int i=0;
        for (i=0;i<5;i++)
        cout << "\n this is my program\n";
        getch();
}
```

يُعطى متغير الحلقة **i** في هذا المثال القيمة الأولية 0، ثم يتغير تقييم تعبير الاختبار. إذا كان التعبير صح (إذا كان **i** أصغر من 5) يتمّ تنفيذ جسم الحلقة ثم تعبير التزايد (تتمّ زيادة قيمة **i** بـــ 1). إذا كان تعبير الشرط خطأ، تتوقف الحلقة.

ملاحظة:

ضمن حلقة الـــ for لا يلعب معامل الزيادة قبل أو بعد المتغير

مثال:

اكتب برنامجاً بلغة الـــ C++ يمكن من طباعة جملتين متتاليتين على سطرين مختلفين

My name is Ahmad و this is my program

وذلك خمس مرات

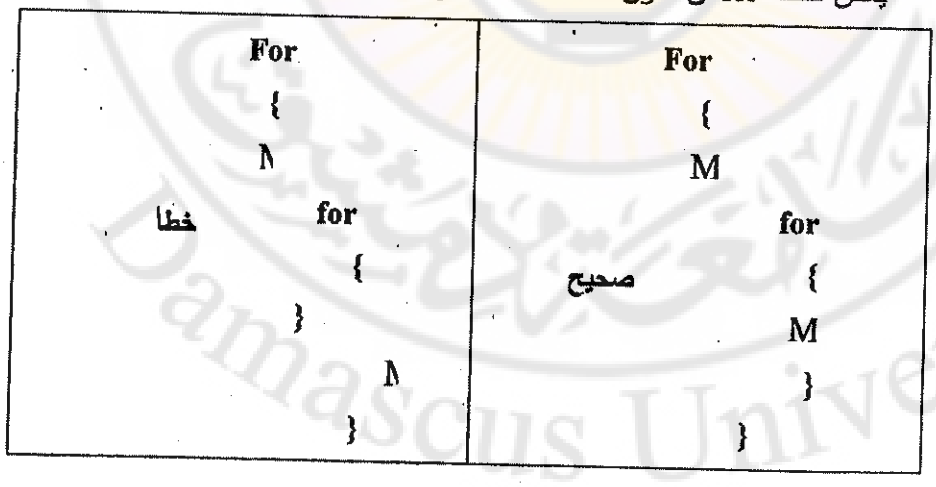
```
#include <iostream.h>
#include <conio.h>
void main ()
{
        clrscr();
        int i;
        for (i=1;i<6;i++)
        {
                cout << "\n this is my program\n";
                cout << "My name is Ahmad";
        }
        getch();
}
```

9- الحلقات المتداخلة:

يمكن للحلقات أن تكون متداخلة. يمكن أن نضع أي نوع من الحلقات ضمن أي نوع آخر، ويمكن مداخلة الحلقات في حلقات متداخلة في حلقات أخرى، الخ.

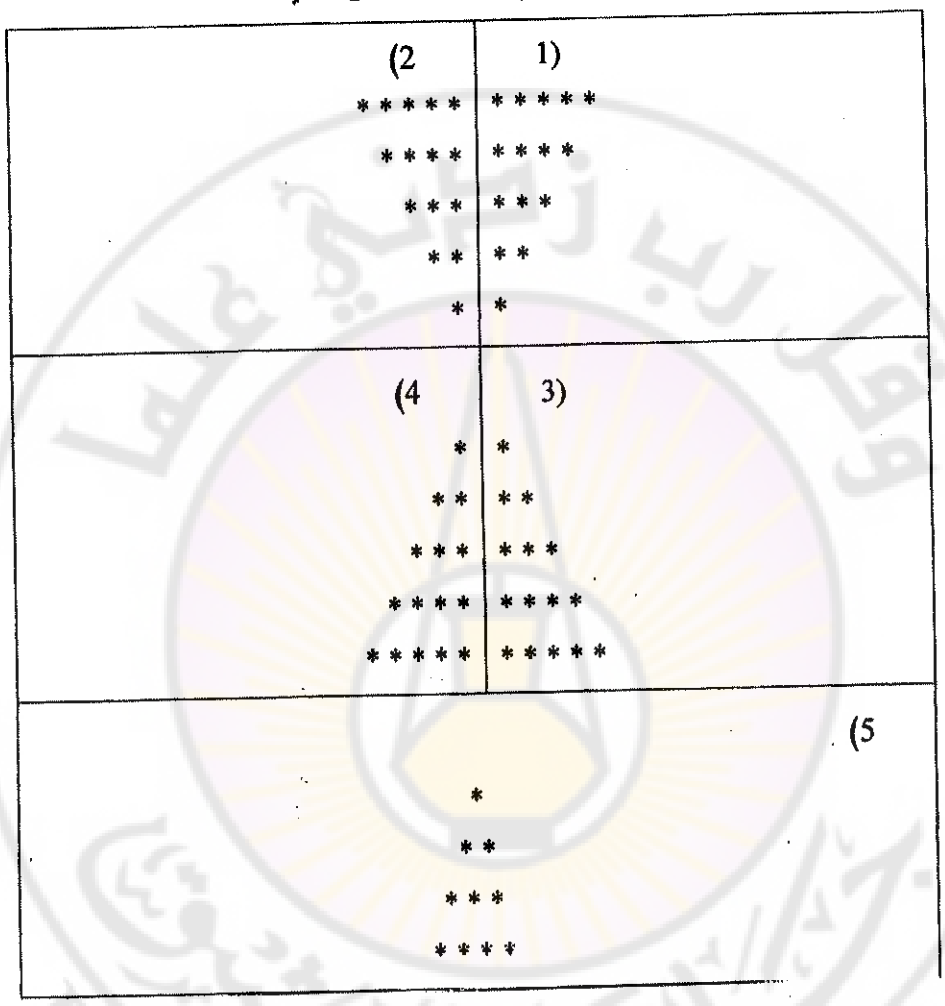
ملاحظة:

يمكن لحلقة for أن تكون حلقات متداخلة ولا يمكن أن حلقات متقاطعة.



مثال:

اكتب برنامجاً بلغة C++ يمكن من طباعة الأشكال التالية



الحل:

سنكتفي بحل الطلب الأول

```
#include <iostream.h>
#include <conio.h>
void main ()
{
        clrscr();
```

```
        for (int j=5; j >0; j--)
        {
                for(int i=1; i<j ; i++)
                {
                        cout << "*";
                }
                cout << "\n";
        }
        getch();
}
```

10- المعاملات المنطقية:

تمكن المعاملات المنطقية (logical operators) من العمل على القيم صح / خطأ. تتضمن لغة البرمجة C++ ثلاث معاملات منطقية، اثنان منها تدمج قيمتين صح / خطأ، والثالثة يعكس القيمة صح / خطأ.

المعامل المنطقي	معناه	مثال
&&	AND (و)	x>0 && x<10
LI	OR (أو)	x==3 ; l l x<1
!	NOT (نفي)	!x

الجدول (4-2) المعاملات المنطقية

يكون التعبير AND صحيحاً فقط إذا كان التعبيران الموجودان على جانبي المعامل && صح. غالباً ما يتمّ استعمال المعامل AND لتحديد فيما إذا كان متغير ما يقع ضمن مجال محدد.

مثال:

```
x>=0 && x<=10
```

تكون العبارة صحيحة إذا كان المتغير x يقع في المجال 1 إلى 9.

مثال:

```
ch>='a' && ch<='z'
```

تكون العبارة صحيحة إذا كان المتغير **ch** حرفاً صغيراً.

يؤدي المعامل **OR (||)** إلى صح إذا كان أحد التعبيرين أو كلاهما صحيحاً.

مثال:

```
j==0 || ch=='q'
```

تكون العبارة صحيحة إذا كان **j** يساوي **0** أو **ch** يساوي **'q'**، أو إذا كان كلاهما صحيحاً.

يستخدم المعامل **OR** لمعرفة فيما إذا كانت قيمة ما تقع خارج مجال محدد.

مثال:

```
x< 75 || x>100
```

تكون العبارة صحيحة إذا كان المتغير **x** يقع خارج المجال **75** إلى **100**.

استخدام الممعامل **NOT (!)** يُبطل تأثير المتغير الذي يليه. لذا، التعبير **!x** صح إذا كان المتغير **x** خطأ، ويكون خطأ إلى كان **x** صحيحاً. نستخدم معامل النفي عندما نريد أن تستمر حلقة **while** بالتنفيذ حتى يتحقق الشرط ، بدلاً من أن تستمر حلقة **while** بالتنفيذ إلى أن يصبح الشرط غير محقق.

مثال:

```
while (!x)
{
<statements>;
}
```

ستتكرر الحلقة إلى أن يصبح المتغير صحيحاً (غير صفر).

يمكن كتابة التعابير المنطقية في عدة طرق. التعبير

```
x< 75 || x>100
```

يمكن كتابته كالتالي

!(x< 75 && x>100)

لأن عدم وجود قيمة المتغير x في المجال يعني وجود قيمة المتغير خارج المجال.

11- الأولوية Precedence:

يتمّ تنفيذ عمليات الضرب والقسمة في التعابير الرياضية قبل عمليات الجمع والطرح. في التعبير التالي.

مثال:

2 * 3 + 3 * 4

يتمّ ضرب 2 و3 (النتيجة 6) ثم يتمّ ضرب 3 و4 (النتيجة 12)، بعدها يتمّ جمع هاتين النتيجتين مما يؤدي إلى القيمة 18. يتمّ تنفيذ عمليات الضرب قبل الجمع لأن العامل * له أولوية أعلى من أولوية العامل +.

x< 75 || x>100

المعامل	أنواع المعاملات	الأولوية
* / %	الضرب - القسمة - باقي القسمة	أعلى
+ -	جمع - الطرح	
> < => =< == =!	علائقية (معاملات المقارنة -معامل النفي)	
&& ||	منطقية (or , and)	
=	تعيين	أدنى

الجدول (4-3) ترتيب الأولويات:

نتأكد أن القيمة صح / خطأ للتعبيرين x <75 و x >100 من خلال تقييمهما أولاً، قبل تقييم المعامل OR. لأن المعاملات العلائقية لها أولوية أعلى من المعاملات المنطقية.

كيف يتمّ تقييم التعبير التالي؟

n + 2 < x + 3

نريد مقارنة تعبيرين حسابيين: **n+3** و**x+3**. بما أن المعاملات الحسابية لها أولوية أعلى من المعاملات العلائقية يتمّ تنفيذ المعاملات الحسابية أولاً ثم ينفذ معامل المقارنة. يبين الجدول (4-3) ترتيب الأولويات. أن معامل التعيين، =، له الأولوية الأدنى. أي يتمّ تنفيذه بعد تنفيذ كل المعاملات الأخرى.

12- العبارة if:

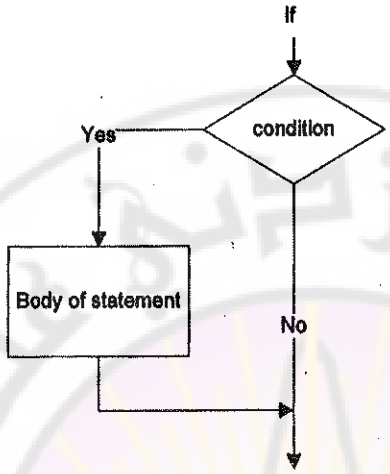
تمكن العبارة **if** من اتخاذ القرار في لغة البرمجة **C++**. تتألف العبارة **if** من الكلمة الأساسية **if** يليها تعبير اختبار بين أقواس. ويتألف جسم القرار ، إما من عبارة واحدة أو من عدة عبارات تحيطها أقواس حاصرة.

مثال:

```
If (x==0)
Cout<< "Division by zero is illega";
```

إذا كان المتغير المسمى **x** يساوي **0**، فإن هذه العبارة تسبب ظهور العبارة الموجودة ضمن القوسين. وإذا لم يكن المتغير صفراً، لا يخرج شيء.

- المخطط التدفقي لعبارة if



الشكل (4-5) المخطط التدفقي للعبارة if .

- التركيب النحوي ل لعبارة if

نميز حالتين

- الشرط يحوي عبارة واحدة فقط

```
if (condition)
Statement;
```

- الشرط يحوي أكثر من عبارة برمجية

```
if ( Condition)
{
        statement 1;
        statement 2;
        N
}
```

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال عدد صحيح ويختبر فيما إذا كان

زوجي فإذا كان زوجياً يطبع عبارة even.

الحل:

```
#include <iostream.h>
#include <conio.h>
void main ()
{
        clrser();
        int x;
        cout << "Enter x=";
        cin >> x;
        if(x%2 == 0)
                cout << x << " is even";
        getch(); }
```

ملاحظة:

الشرط if إذا كان المقصود به تحديد حالة true فلا داعي للإشارة إلى ذلك أي يمكننا أن نكتب.

```
if(x%2)
```

13- عبارة if المتداخلة:

تمكن لغة البرمجة C++ من استخدام if الشرطية المتداخلة علماً أن المترجم يستطيع تمييز (64) شرط متداخل على أكثر. وهذا ما يسمى بـ nested if.

- التركيب النحوي:

نميز حالتين:

- إذا كانت تحوي عبارة واحدة

```
if (condition)
        if (condition)
                if (condition)
                        if (condition)
                                statement;
```

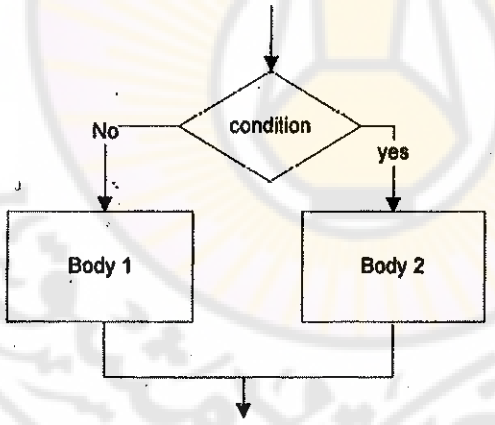
- إذا كانت تحوي أكثر من عبارة

```
if (condition)
        if (condition)
                if (condition)
                {
                        statement 1;
                        statement 2;  }
```

14- العبارة if..else:

في العبارة **if** البسيطة، إذا كان الشرط صح يتمّ تنفيذ المهمة البرمجية، لكن إذا لم يكن الشرط صح لن ينفذ شيء. لتنفيذ مهمة برمجية في الحالتين: مهمة برمجية إذا كان الشرط صح ومهمة برمجية آخرى إذا كان الشرط خطأ. لتحقيق ذلك، نستعمل العبارة **if..else**.

المخطط التدفقي لعبارة if ... else



الشكل (4-6) المخطط التدفقي لعبارة if ... else

- التركيب النحوي if ... else

كما الحال مع عبارات في لغة البرمجة **C++** الأخرى، يمكن أن يتألف جسم **if** أو جسم **else** من عدة عبارات تحيطها أقواس حاصرة.

- نميز عدة حالات:

- العبارة if تحوي عبارة برمجية واحدة والعبارة else تحوي عبارة برمجية واحدة

```
if (condition)
        statement ;
else
        statement ;
```

- العبارة if تحوي أكثر من عبارة والعبارة else تحوي عبارة برمجية واحدة

```
if (condition)
{
        statement 1;
        statement 2;
        .
        .
}
else
        statement ;
```

- العبارة if تحوي عبارة برمجة واحدة و else تحوي أكثر من عبارة برمجية

```
if (condition)
        statement;
else
{
        statement 1;
        statement 2;
        .
        .
}
```

- العبارة if والعبارة else تحوي أكثر من عبارة برمجية

```
if (condittion)
{
        statement 1;
```

```
        statement 2;
        }
}
else (condition)
{
        statement 1;
        statement 2;
        }
}
```

مثال:

اكتب برنامجاً يمكن من إدخال عددين صحيحين وإيجاد الأكبر منهما.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
        //clrscr();
        int n,m;
        cout << "Enter n=";
        cin >> n;
        cout << "Enter m=";
        cin >> m;
        if( n > m)
                cout << "the creater number is:"<<n;
        else
                cout << "the creater number is:"<<m;
        getch();
}
```

ملاحظة:

يمكن للفاصلة المنقوطة أن تسبق عبارة else

- تعبير الاختبار:

تعبير الاختبار هو عبارة if أو if..else يمكن أن تكون معقدة كالتعابير في الحلقات.

مثال:

العبارة **if..else** التالية تقدّم اليوم من 28 شباط إلى 1 آذار أو إلى 29 شباط، وفقاً لما إذا كانت السنة كبيسة أم لا.

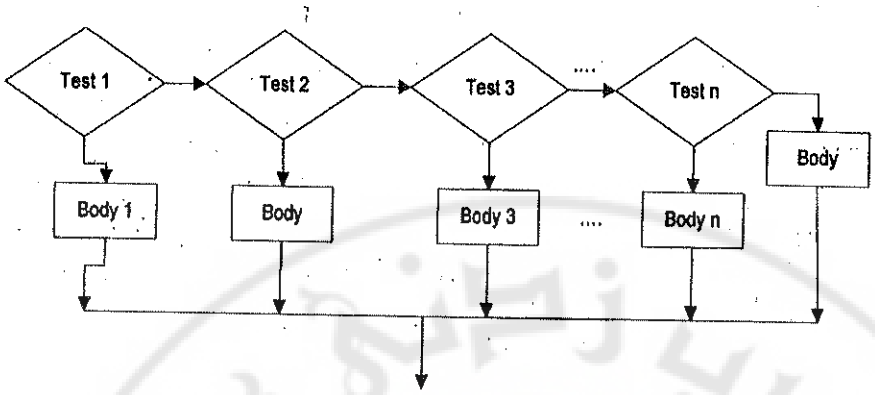
```
// if it is feb 28 and it is a leab year
if (day ==28 && year%4==0 && year%100!==0)
 day ==29;          // then the next day is the 29th
else
 {
  day = 1;          // next day is March 1st
  month = 3;
  }
```

تحدث السنوات الكبيسة عندما تكون السنة قابلة للقسمة على 4 (1996 مثلاً هي سنة كبيسة) لكن غير قابلة للقسمة على 100 (إذا 1900 ليست سنة كبيسة، رغم أنه يمكن قسمتها على 4). يتمّ استعمال عامل الباق (%) لمعرفة ما إذا كان يمكن قسمة السنة على 4 (وعلى 100) من دون باقي أم لا. تضمن العوامل **AND** أن 29 شباط يظهر فقط عندما تكون كل الشروط صحيحة في الوقت نفسه.

- العبارات if..else المتداخلة:

تمكن من وضع العبارات **if..else** ضمن بعضها بعضاً. نموذجياً، تنتهي العبارات المتداخلة في الجسم **else** وليس في الجسم **if**.

المخطط التدفقي العبارات if..else المتداخلة:



الشكل (4-7) المخطط التدفقي العبارات if..else المتداخلة

- التركيب النحوي لهذه العبارة:

```
if (condition)
        statement;
 else if (condition)
        statement;
 else if (condition)
        statement;
    .
    .
    .
 else if (condition)
        statement;
```

ملاحظة:

يمكن للمترجم أن يتعامل مع (64) شرطاً متداخلاً

ملاحظة:

عند استعمال العبارات **if..else** المتداخلة. فنسيان أي **if** لأي جزء **else** هو مصدر للخطأ.

15- التحكم بالحلقات:

إن العبارتين **break** (قطع) و**continue** (متابعة)، يتمّ استعمالهما بفعالية في الحلقات لاتخاذ القرارات. كما أن العبارة **break** هي ميزة مهمة في العبارة **switch**.

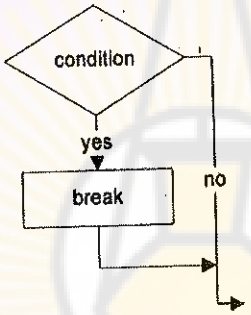
تعمل الحلقات عادة بشكل جيد مع التركيب النحوي المباشر. لكن نحتاج في بعض الحالات إلى تنفيذ مهام برمجية محددة. تزود العبارتان **break** و**continue** هذه المرونة المطلوبة.

- **العبارة break:**

تمكن العبارة **break** من الخروج من الحلقة فوراً، كما هو مبين في الشكل (4-8).

غالباً ما يتمّ استعمال هذه العبارة لمعالجة الحالات غير المتوقعة أو غير القياسية التي تظهر في الحلقة.

- **المخطط التدفقي العبارة break:**



الشكل (4-8) المخطط التدفقي العبارة break:

فلو وضعنا في البرنامج السابق بدل تعليمة **continue** تعليمة break فإن المترجم لن ينفذ شيئاً.

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال عدد صحيح موجب وبيان فيما إذا كان العدد أولياً أم لا.

الحل:

نضبط المتغير **flag** عند 1 إذا كان عدداً صحيحاً يدعى **n** رقماً أولياً **(flag)** أو عند 0 إذا لم يكن رقماً أولياً. لمعرفة ما إذا كان **n** رقماً أولياً أم لا، نستعمل أسلوباً مباشراً لمحاولة قسمته على كل الأرقام وصولاً إلى **n-1**. إذا انقسم **n** على أحد هذه الأرقام بشكل صحيح (من دون باق)، فإنه لا يكون أولياً.

نقسم على كل الأرقام وصولاً إلى **n-1**، لذا استعملت حلقة **for** بتعابير مناسبة. لكن إذا انقسم الرقم **n** على أحد القيم **j** بشكل صحيح، لا داعي لإكمال الحلقة. عندما يجد البرنامج الرقم الأول الذي يقسم **n** بشكل صحيح، يجب أن يضبط المتغير **flag** عند **0** ويخرج من الحلقة فوراً. تتيح العبارة **break** الخروج من الحلقة في أي وقت.

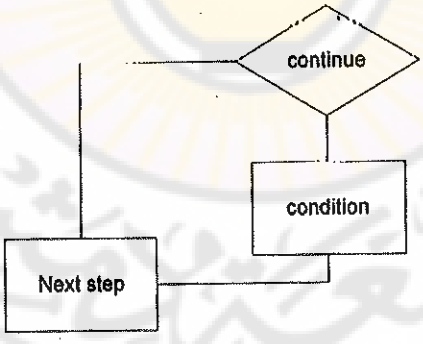
```
#include <iostream.h>
#include <conio.h>
void main ()
{
        clrscr();
        int n;
        cout << "Enter n=";
        cin >> n;
        while(n < 0)
        {
                cout << "enter again n=";
                cin >> n;
                int flag = 1;
                if(n == 0 || n == 1)
                        flag = 0;
                else if(n ==2)
                        flag = 1;
                else
                {
                        for (int i = 2; i < n ; i++)
                        {
                                if (n % i = 0)
                                {
                                        flag = 0;
                                        break;
```

```
            }
        }
        if (flag == 0)
            cout << n << "is not prim";
        else
            cout << n << "is prim";
        getch();
    }
}
}
```

- العبارة continue:

العبارة **continue** مشابهة للعبارة **break** وتستخدم في الحالة غير المتوقعة في الحلقة. لكنها تعيد التنفيذ إلى أعلى الحلقة ــ مما يؤدي إلى متابعة الحلقة ــ بدلاً من الخروج منها. يبين الشكل (4-9) المخطط التدفقي لعبارة **continue** . في حين أن العبارة **break** تؤدي إلى الخروج من الحلقة، تؤدي العبارة **continue** إلى "عرقلة" بقية الحلقة مع استمرار عمل هذه الأخيرة. أي أنها تجعل التنفيذ ينتقل إلى أعلى الحلقة.

- المخطط التدفقي لعبارة **continue**



الشكل (4-9) المخطط التدفقي لعبارة **continue**

مثال:

اكتب برنامجاً بلغة C++ يمكن من إيجاد مجموعة الأعداد التي تقبل القسمة على الأعداد الفردية 11,7,5,3 والمحصورة بين 1 و 15000.

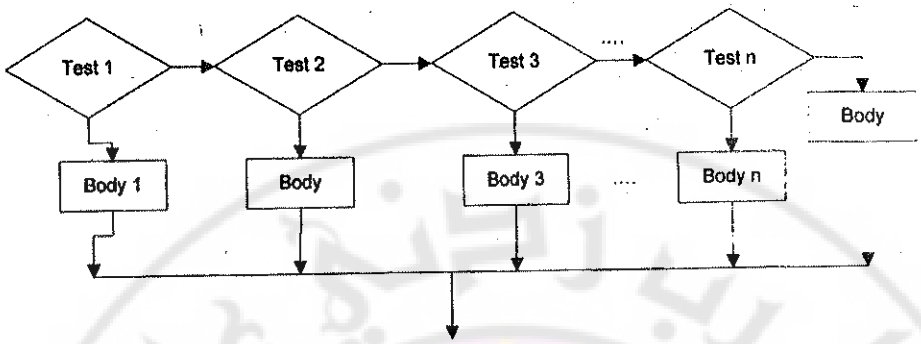
الحل

```
#include <iostream.h>
#include <conio.h>
void main ()
{
        //clrscr();
        int n,m;
        for (int i=1; i<15000;i++)
        {
                if(i%3 != 0)
                        continue;
                if(i%5 != 0)
                        continue;
                if(i%7 != 0)
                        continue;
                if(i%11 != 0)
                        continue;
                cout << " "<<i;
        }
        getch();
}
```

- العبارة switch:

تختبر عبارة **switch** المتغير وتوجه البرنامج إلى أقسام مختلفة من البرنامج، وفقاً لقيمة ذلك المتغير. تتألف العبارة **switch** من الكلمة الأساسية **switch** يليها اسم متغير بين قوسين ثم جسمها بين أقواس حاصرة. يتضمن الجسم عدداً من الحالات، وهي أسماء تليها نقطتان. تتألف هذه الحالات من الكلمة الأساسية **case** ثم ثابت ثم نقطتين. عندما تكون قيمة متغير العبارة **switch** مساوية للثابت المذكور في أحدى الحالات **case** ، ينتقل التنفيذ إلى العبارات التي تلي ذلك الحالة.

- المخطط التدفقي لعبارة switch



الشكل (4-10) المخطط التدفقي لعبارة switch

- التركيب النحوي لعبارة switch:

```
switch (variable)
{
        case (value 1):
        {
                statement 1;
                ٨
                statement n;
        }
        break;
    case (value 2):
        {
                statement 1;
                ٨
                statement n;
        }
        break;
        ٨
        case (value n):
        {
                statement 1;
                ٨
                statement n;
        }
```

```
        break;
    default;
        {
            statement ;
            statement;
        }
}
```

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال دراجات امتحان طالب بحيث يكون ما يلي محققاً.

إذا كانت درجة الطالب أقل من 50 يطبعF.

إذا كانت درجة الطالب محصورة بين 50 و 59 يطبع D

إذا كانت درجة الطالب محصورة بين 60 و 69 يطبع D^+

إذا كانت درجة الطالب محصورة بين 70 و 79 يطبع C

إذا كانت درجة الطالب محصورة بين 80 و 89 يطبع B

إذا كانت درجة الطالب محصورة بين 90 و 99 يطبع A

إذا كانت درجة الطالب 100 يطبع A^+

وذلك باستخدام العبارة switch

الحل:

```
#include <iostream>
#include <conio.h>
using namespace std;
void main ()
{
    //clrscr();
    int n;
    cout << "enter n=";
    cin >> n;
```

```
while (n < 0 || n > 100)
{
        cout << "enter agin n=";
        cin >> n;
}
switch(n /10)
{
case 10:
        cout << "A+";
        break;
case 9:
        cout << "A";
        break;
case 8:
        cout << "B";
        break;
case 7:
        cout << "C";
        break;
case 6:
        cout << "D+";
        break;
case 5:
        cout << "D";
        break;
default:
        cout << "F";
}
}
```

إن المتغير أو التعبير المستعمل لتحديد الحالة التي يجب الانتقال إليها، يجب أن تكون عدداً صحيحاً أو محرفاً أو يجب أن يؤدي تقييمه إلى عدد صحيح أو محرف. ويجب أن تكون القيم التي تلي الحالة – case أو يجب أن يؤدي تقييمها إلى – ثوابت أعداد صحيحة أو محارف. أي أنه يمكن من استعمال أسماء متغيرات أو تعابير، كـ alpha و j+20 و ch+'0'، طالما كانت alpha و j و ch لها قيم مناسبة مسبقاً. بعد الانتقال إلى الحالة، يتمّ تنفيذ العبارات التي تليه واحدة تلو الأخرى. سيتمّ تنفيذ العبارة

cout في المثال. إذا لم نضع عبارة **break**، سيتابع التنفيذ إلى العبارة **cout** التالية. لا تحصر الحالة قطع المهمة البرمجية، إنها مجرد نقاط دخول. تؤدي العبارة **break** إلى منع تنفيذ بقية العبارة **switch**. إذا لم تتحقق أي حالة، ينتقل التنفيذ إلى الحالة الافتراضية **default** (أو، إذا لم يكن هناك الحالة الافتراضية ينتقل التنفيذ إلى أسفل العبارة **switch**).

16- المعامل المشروط (الشرط المزدوج):

العامل المشروط هو المعامل الوحيد في لغة البرمجة C++ الذي يعمل على ثلاث قيم. وهو يتألف من رمزين: علامة استفهام ونقطتين. أولاً يأتي تعبير الاختبار (مع أقواس اختيارية) ثم علامة الاستفهام ثم قيمتان تفصلهما نقطتان. إذا كان تعبير الاختبار صحيحاً، يُعطى التعبير بأكمله القيمة الموجودة قبل النقطتين (المتغير n). وإذا كان تعبير الاختبار خطأ، يعطى التعبير بأكمله القيمة التي تلي النقطتين (m).

مثال:

اكتب برنامجاً يمكن من إدخال عددين صحيحين وإيجاد الأكبر منهما.

يمكن أن نكتب الشرط بالشكل التالي:

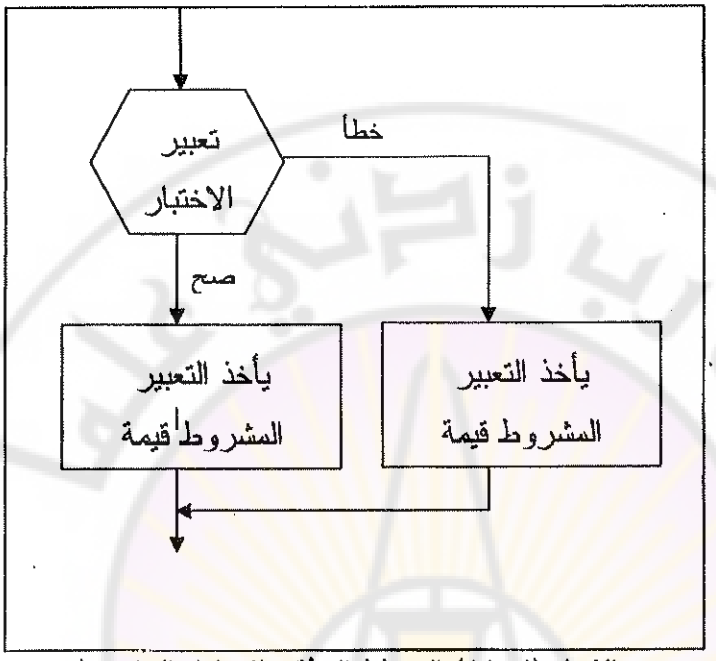
```
z=(n>m)? n:m;
```

أي يمكننا كتابة البرنامج السابق بالشكل:

```
#include <iostream.h>
#include <conio.h>
void main ()
{
        //clrscr();
        int n,m,z;
        cout << "Enter n=";
        cin >> n;
        cout << "Enter m=";
        cin >> m;
        z=(n>m)? n:m;
        cout << "the creater number is:"<<z;
```

```
getch();
```

- المخطط التدفقي للمعامل المشروط



الشكل (4-11) المخطط التدفقي للمعامل المشروط

17- أنواع البيانات الجديدة (الكائنات):

إن تمثيل أنواع بيانات جديدة هو استعمال رئيسي للغات البرمجة غرضية التوجه.

أما أنوع البيانات الأخرى التي قد تريد تمثيلها باستعمال الصفوف هي:

- التواريخ، التي لها قيم منفصلة للسنة والشهر واليوم.
- الكسور، التي لها قيم منفصلة للبسط والمقام.
- النقاط على سطح المستوي الديكارتي (Cartesian)، التي لها إحداثيات X و Y منفصلة.
- الأرقام المركبة (العقدية) في الرياضيات، التي لها مكوّن حقيقي وآخر وهمي.

لاستخدام المعامل + من أجل جمع قيمتين من نوع بيانات معرّف من قبل

المستخدم، يتطلب تعريف قاعدة تنفيذ الجمع.

مثال:

تمثيل الأوقات برمجياً:

```
// time.ccp
# include <iostream.h>
    class time1
    {
      private:
            int hours;       //0 --23
            int minutes;     // 0 --59
      public:
            void set( )
              {
                char ch
                cout<<"Enter time  (format  23:59 ):";
                 cin>> hours>>ch>> minutes;
                }
            void display ( )
               {
                 cout<< hours<<":"<< minutes;
                 }
    };
void main ( )
    {
       time1  obj1,obj2;                    //create  two  variables
(objects)
        cout<<"For obj1,";
        obj1.set( );
        cout<<"For obj2,";
        obj2.set( );
        cout<<"\n obj1 =";
```

```
        obj1.display( );          // display obj1
        cout<<"\n obj2 =";
        obj2.display( );          // display obj2
    }
```

بحديد الصف **time1** عضوي بيانات، الساعة **hour** والدقائق **minutes**، وعضوين داليين **set()** و**display()**، اللذين يحصلان على قيمة وقت من المستخدم ويعرضان قيمة. يمكن أن يكون خرج البرنامج.

```
For obj1, Enter time (format 23:59 ): 11:21
For obj2, Enter time (format 23:59 ): 18:45
obj1 = 11:21
obj2 = 18:45
```

يدخل المستخدم وقتين ويعرضهما البرنامج.

- **معامل الإسناد = :**

تمكن لغة البرمجة C++ من إسناد قيمة أحد الكائنات إلى كائن آخر.

مثال:

```
obj1 = obj1;
```

ثم تعرض قيمة المتغير **obj2**.

```
void main ( )
   {
        time1 obj1,obj2;              //create two variables
(objects)
        cout<<"For obj1,";
        obj1.set( );
        obj1 = obj1;          // obj2 equal obj1
        cout<<"\n obj2 =";
        obj2.display( );          // display obj2
    }
```

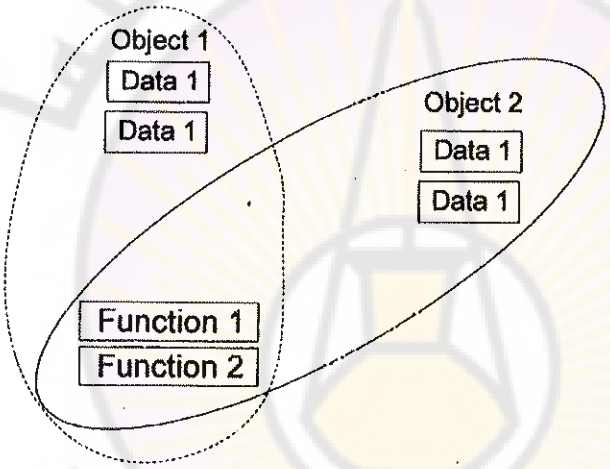
مهما كان عدد متغيرات البيانات الموجودة في الكائن الأول، سيتمُّ نسخها إلى الكائن الثاني أثناء تنفيذ عبارة الإسناد.

ملاحظة:

توجد صورة واحدة من كل عضو دالي في الذاكرة، ومهما كان عدد الكائنات المنشأة من تلك الصف. لذلك فإن الكائنات تتشارك بالأعضاء الدالية. بينما البيانات المخزنة في هذه الكائنات يمكن أن تكون قيمها مختلفة. عندما نستدعي العضو الدالي **display( )** بوساطة الكائن **obj2**:

```
obj2.display( );
```

سيتمّ استدعاء الدالة **display()**، لكنها ستعمل على البيانات المخزنة في **obj2**.



الشكل (4-12) الكائنات والبيانات والدوال والذاكرة

18- الدوال:

أ- دوال مسبقة التعريف لتنسيق المخرجات:

تمكن لغة البرمجة C++ من استخدام أدوات متعددة لتنسيق المخرجات على الشاشة. لقد عرضنا استخدام الدالة **endl** وهي دالة لا تأخذ أي وسيط.

setw( ):

تأخذ الدالة setw() وسيطة واحدة، وهي عرض ــ بالمحارف ــ الحقل التالي الذي سيتمّ عرضه في الخرج. يتمّ عادة ضبط عرض الحقل تلقائياً عند كمية الأعداد التي سيتمّ عرضها. تتمّ تعبئة المساحة غير المستعملة بحرف تعبئة، والفراغ هو (' ') بشكل افتراضي. الدالة setw() هي دالة تأخذ متغيراً واحداً من نوع صحيح ويمكن هذه الدالة من تحديد نقطة البداية في الكتابة أي أنها تنقل المشيرة الضوئية للنقطة التي يرغب بها المستخدم ولتطبيق هذه الدالة نحتاج إلى الترويسة التالية:

```
#include <iomannip.h>
```

مثال:

أوجد مخرجات البرنامج التالي:

```
#include <iostream.h>
#include <conio.h>
#include <iomannip.h>
void main ()
{
clrscr();
int n=5;
cout << setw(6);
cout << n;
getch();
}
```

:setfill('0')

دالة ملء الفراغ cout fill ('character') وهي دالة تأخذ متغيراً واحداً محرفياً وتملأ الدالة الفراغ بهذا التغيير.

:cout width( )

دالة تحديد عدد خانات الإخراج ، تأخذ متغيراً من نوع صحيح تحدد عدد خانات الإخراج.

ملاحظة:

إذا كان عدد الخانات المحددة بهذه الدالة أقل من عدد الخانات اللازمة لإخراج

قيمة المتحول فإن المترجم يهمل هذه الدالة.

مثال:

اكتب برنامجاً بلغة C++ يمكن من إخراج المتغير x الذي قيمته (123) على 5 خانات وملء الفراغ بـــ #.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
        clrscr();
        int n,m;
        int i = 123;
        cout fill('#');
        cout width(5);
        cout <<i;
        getch();
}
```

مثال:

اكتب برنامجاً يقوم بطباعة الشكل وذلك باستخدام setw( )

```
    1
   2 2
  3 3 3
 4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
5 5 5 5 5
 4 4 4 4
  3 3 3
   2 2
    1
```

```
#include <iostream>
#include <conio.h>
#include <iomanip.h>
void main ()
{
clrscr();
int i,j;
int counter = 20;
    for (int i=1; i < 10; i++)
    {
    cout << setw(counter);
    for (int j=1; j < i+1; j++)
    cout << i << " ";
    cout << endl;
    }
        for (int k=8; k >= 1; k--)
        {
        cout << setw(counter);
        for (int j=1; j <= k; j++)
        cout << k << " ";
        cout << "\n";
        }
getch();
}
```

مثال:

اكتب برنامجاً يمكن من إدخال عدد صحيح موجب وإنجاز ما يلي:

إذا كان العدد يقبل القسمة على (4) نوجد ناتج قسمته على (2) ونختبر فيما إذا كان الناتج عدداً فردياً أم عدداً زوجياً عدا ذلك يطبع باقي القسمة.

```
#include <iostream>
#include <conio.h>
void main ()
{
        clrscr();
        int n,m;
        cout << "enter n=";
        cin >> n;
```

```
    while (n < 0)
    {
            cout << "enter agin n=";
            cin >> n;
    }
    switch (n % 4)
    {
    case 0:
            {
                    m = n / 4;
                    cout << m;
                    switch (m % 2)
                    {
                    case 0:
                            cout << m << "is even";
                            break;
                    case 1:
                            cout << m << "is odd";
                            break;
                    }
            }
            break;
    case 1:
            cout << "remainder is 1";
            break;
    case 2:
            cout << "remainder is 2";
            break;
    default:
            cout << "remainder is 3";
    }
    getch();
}
```

- مكتبات الدوال (الدوال مسبقة التعريف):

إن مكتبات دوال لغة البرمجة C لا تزال تشكل ناحية أساسية في البرمجة باللغة البرمجية C++. هناك العديد من المكتبات للدخل / الخرج، وتحويل البيانات، ومعالجة سلاسل المحارف، والتحكم بالأدلة والملفات، وتخصيص الذاكرة، والرياضيات،

وغيرها. كل دالة تتطلب شمل ملف ترويسة ما قبل أن يصبح بالإمكان استدعاؤها.

أولاً. الدوال الرياضية Mathmatical function:

وهي دوال تمكن من تنفيذ مهام رياضية محددة تمّ فيها الأخذ بعين الاعتبار أن تكون هذه الدوال متوافقة مع المفاهيم الرياضية التي تمّ التعرف إليها فيما سبق ولاستخدام أي دالة رياضية مسبقة التعريف نحتاج إلى الترويسة التالية:

#include <math.h>

بعض الدوال الرياضية:

1) دالة القيمة المطلقة

Abs(x) $\Leftrightarrow |x|$

2) دالة الجذر التربيعي

sqrt(x) $\Leftrightarrow \sqrt{x}$

3) دالة أكبر قيمة صحيحة تلي العدد x

cil(x) $\Leftrightarrow [x]$

4) دالة أصغر قيمة صحيحة تسبق المتغير x

floor(x) $\Leftrightarrow \lfloor x \rfloor$

مثال:

$$x = 3.5 \Rightarrow cil(x) = 4 \quad floor(x) = 3$$
$$x = 1.2 \Rightarrow cil(x) = -1 \quad floor(x) = -2$$

5) دالة القوة

pow(x,y) $\Leftrightarrow x^y$

6) الدالة e^x

exp(x)

مثال:

$$2.7 = e^1 \Leftrightarrow pow(1) = 2.7$$

7) الدوال المثلثية:

$\sin(x)$	$\arcsin(y)$
$\cos(x)$	$\arccos(y)$
$\tan(x)$	$\arctan(y)$
$\cot an(x)$	$arc\cot an(y)$

حيث x بالراديان و $y \in \Re$.

8) دالة اللوغارتيم

Ln(y)

ثانياً. دوال سلاسل الحروف

هي دوال تمكن من إدخال سلاسل حروف ومعالجة هذه السلاسل حسب الحاجة البرمجية ولكي نتمكن من إدخال سلاسل الخروف نحتاج إلى الترويسة التالية:

```
#include <string.h>
```

لتكن s سلسلة محارف، ولنعرف دوال سلاسل الحروف.

1) الدالة **strlen(s)**

تمكن هذه الدالة من حساب طول سلسلة المحارف s أي عدد الأحرف الموجودة في هذه السلسلة إضافة إلى الفراغات حيث يعتبر الفراغ محرفاً.

ملاحظة:

تعيد هذه الدالة قيمة صحيحة مثال ;"s = "Ahmad abou Anas

عندها ;x= strlen(s) تساوي الـ 15.

2) الدالة **strcopy**

مثلاً strcopy(s1,s2)

تمكن هذه الدالة من نسخ سلسلة المحارف s2 إلى سلسلة المحارف s1 مع الأخذ بعين الاعتبار أنه إذا لم تكن s1 فارغة فعند إجراء عملية النسخ تحذف المحارف المقابلة لسلسلة المحارف الجديدة.

3) الدالة **strcmp(s1,s2)**

تمكن هذه الدالة من مقارنة سلسلة المحارف s1 مع سلسلة المحارف s2 وفي حال عدم وجود تطابق تعيد رسالة عدم التطابق والقيمة المعادة من هذه الدالة هي قيمة بولیانية مع الأخذ بعين الاعتبار أنه يمكن تحديد الفوارغ الموجودة بين السلسلتين.

4) الدالة **strcat(s1,s2)**

تمكن هذه الدالة من إلحاق محتويات سلسلة المحارف s2 بسلسلة المحارف s1.

```
s1 = "Ahmad";
s2 = "abuo anas";
s=strcat (s1,s2);
cout << s;
s = "ahmad abou anas";
```

5) دالة **toupper()** :

```
char ch = 'a';
toupper(ch);
```

تقوم هذه الدالة بتحويل جميع محارف السلسلة إلى الحجم الكبير.

6) الدالة tolower()

تقوم هذه الدالة بتحويل جميع محارف السلسلة إلى الحجم الصغير.

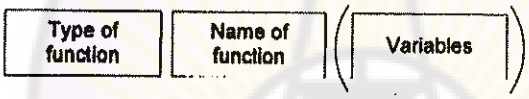
7) الدالة towasii

تمكن هذه الدالة من إظهار المحرف من جدول ASCII.

ثالثاً: الدوال البرمجية

هي دوال يكتبها المبرمج لتنفيذ مهام برمجية و تستطيع هذه الدوال استدعاء دالة أخرى ويمكن للدالة استدعاء نفسها (التعاودية).

- التركيب النحوي للدوال:



{

Body of function

}

الشكل (4-13) التركيب النحوي للدالة

وتوجد طريقتان لإنشاء هذه الدوال.

ذكر جسم الدالة بعد ترويسة البرنامج ونذكر الدالة التي نرغب باستخدامها قبل استخدامها

المخطط التدفقي لهذه الحالة

Head program

Function

Body program

Main function

1. ذكر الدوال بعد جسم البرنامج على أن يصرح عن هذه الدوال بعد الترويسات مباشرة مع الأخذ بعين الاعتبار أنه لا يوجد أهمية لترتيب الدوال.

- المخطط التدفقي لهذه الحالة

Head program

Prototype of function

Body program

Function

الطريقة الثانية هي الأكثر شيوعاً.

19- وسطاء الدوال:

في معظم الحالات تزود بمعلومات عند استدعائها. يتمّ ذلك بوساطة وسطاء الدالة، وهي قيم يتمّ تمريرها إلى الدالة عند استدعائها.

20- كتابة العضو الدالي:

عند كتابة عضو دالي نكتب أولاً نوع الدالة ثم اسم الدالة ونضع بين قوسين وسطاء، نضع بين الأقواس في سطر تعريف الدالة: نوع البيانات واسم متغير.

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال عددين صحيحين وإيجاد مجموع هذين العددين باستخدام مفهوم الدوال.

الحل:

```
# include <iostream>
#include <conio.h>

void input(int &n)
{
```

```
        cout << "enter element =";
        cin >> n;
}
int sum (int n, int m)
{
        int z;
        z=m+n;
        return z;
}

void main()
{
        clrscr();
        int x,y,r;
        input (x);
        input (y);
        r = sum(x,y);
        cout << x << "+" << y << "=" << r;
        getch();

 }
```

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال عدد صحيح موجب وبيان فيما إذا كان هذا العدد أولياً كذلك بيان فيما إذا كان هذا العدد تاماً وذلك باستخدام الدوال.

```
# include <iostream>
#include <conio.h>
        void input(int &x)
                {
                cout << "enter element =";
                cin >> x;
                while (x < 0)
                {
                cout << "input elemet=";
                cin >> x;
                }
                }
        int prim (int y)
```

```
{
int t=1;
if  (y == 0 || y == 1)
t = 0;
else if (y == 2)
t = 1;
else
{
for (int i=2 ; i < y ; i++)
{
if (y % i == 0)
{
t = 0;
break;
}
}
return t;
}
}
int comp1 (int z)
{
int s=1,l=0;
for (int i=1; i < z; i++)
{
if (z % i == 0)
l+=i;
if (l != z)
s=0;
return s;
}
}

void main()
{
clrscr();
int n,m,r;
input (n);
m = prim (n);
        if (m == 0)
```

```
        {
        cout << n << "is not prim";
        }
        else
        cout << n << "is prim";
        cout << endl;
        r = comp1(n);
        if(r = 0)
        cout << n << "is not complet number";
        else
        cout << n << "in complet number";
getch();
}
```

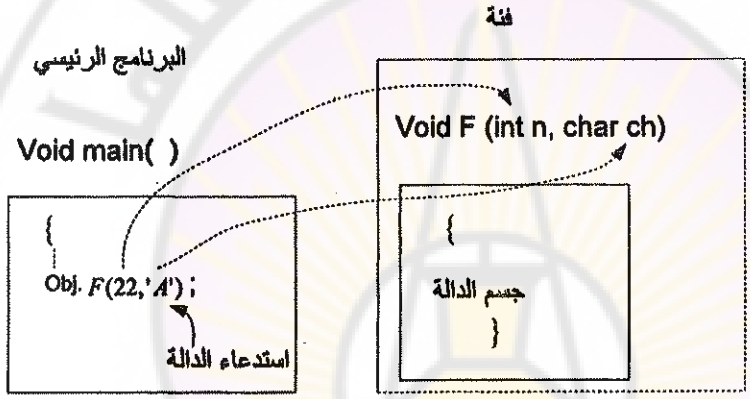
مثال:

اكتب البرنامج السابق باستخدام الطريقة الثانية أي بالتصريح عن الدوال في ترويسة البرنامج وكتابة الدوال بعد الدالة الرئيسية.

```
# include <iostream>
#include <conio.h>
void input (int &);
int prim(int);
int comp1(int);
void main()
{
clrscr();
int n,m,r;
input (n);
m = prim (n);
      if (m == 0)
      {
      cout << n << "is not prim";
      }
      else
      cout << n << "is prim";
      cout << endl;
      r = comp1(n);
      if(r = 0)
```

```
        cout << n << "is not complet number";
        else
        cout << n << "in complet number";
 getch();
 }
 void input(int &x)
         {
         cout << "enter element =";
         cin >> x;
         while (x < 0)
         {
         cout << "input elemet=";
         cin >> x;
         }
         }
int prim (int y)
             {
             int t=1;
             if  (y == 0 || y == 1)
             t = 0;
             else if (y == 2)
             t = 1;
             else
             {
             for (int i=2 ; i < y ; i++)
             {
             if (y % i == 0)
             {
             t = 0;
             break;
             }
             }
             return t;
             }
             }
int comp1 (int z)
             {
             int s=1,l=0;
             for (int i=1; i < z; i++)
             {
```

```
if (z % i == 0)
l+=i;
if (l != z)
s=0;
return s;
}
}
```



الشكل (4-14) استدعاء دالة مع وسيطاء

في السطرتضمن سطر تعريف الدالة **void addhours (int h)** نوع الوسيط **int** واسم الوسيط **h** ويمكن الوصول إليه من أي مكان في الدالة. إن الوسيط **h** معرف ضمن الدالة فقط؛ وغير معرف في الأجزاء الأخرى من البرنامج.

مثال:

اكتب برنامجاً يمكن المستخدم من إدخال الوقت ويتضمن البرنامج تعريف العمليات الحسابية على المتغيرات الجديدة.

```
// time.ccp
# include <iostream.h>
#include <conio.h>
#include <iomannip.h>
```

```
class time1
{
  private:
        int hours;        //0 --23
        int minutes;    // 0 --59
  public:
        void set( )
          {
            char ch
            cout<<"Enter time  (format  23:59 ):";
            cin>> hours>>ch>> minutes;
            }
        void display ( )
            {
              cout<< hours<<":"
                  <<setfill('0)
                   <<setw(2)<< minutes;
              }
        void  add (int  h)
        {
        hours = hours + h;
        if (hours > 23)
        hours = hours- 24;

        }
  };
void main ( )
  {
     time1 obj1;          //create two variables (objects)
     int diffhours;
     char choice;
     do
      {
```

```
        cout<<"For obj1,";
        obj1.set( );
          cout<<"Enter hours  to add:";
           cin>> diffhours;
        obj1. addhours (diffhours )
     cout<<"\n obj1 =";
     obj1.display( );            // display obj1
          cout<<"\n Do another (y/n)?";
           cin>> choice;
      } while (choice !='n')
}
```

يمكن أن يكون خرج البرنامج.

```
For obj1, Enter time  (format  23:59 ): 10:45
Enter hours  to add: 3
obj1 = 13:45
Do another (y/n)? n
```

يكتب المستخدم قيمة time1 وقيمة ساعات فيجمعها البرنامج. يستمر الأمر على هذا الحال إلى أن يكتب المستخدم n لإنهاء الحلقة.

ملاحظة:

يمكن أن تكون الوسيطاء في الدالة من أي نوع، كما يمكن أن يكون هناك أي عدد من الوسيطاء في الدالة.

```
class  para
    {
       void func(int ivar , float fvar , char ch)
       {
       :
       }
       ..
    };
    main( )
```

```
{
    para obj;
    int x= 16;
    float y = 6.0534;
    char ch1='a';
    :
    obj.func(x, y, ch1);
    :
}
```

إن العضو الدالي **funs()** المعرف في الصف **para** يتطلب ثلاث وسيطاء من ثلاثة أنواع مختلفة. ثم أُنشئت في الدالة الرئيسة **main()** كائناً من الصف **para** واستدعيت الدالة **func()** التابعة له، مع قيم مناسبة للوسيطاء.

21- التمرير بالقيمة(value passing by).:

تعني أن الدالة تنشئ متغيراً جديداً (وتحجز له حيز في الذاكرة وباسم وحجم مناسبين) لتخزين قيمة كل وسيط تمّ تمريره إليها. يبين المثال السابق ثلاثة متغيرات في **mian()** هي **x** و**y** و**ch1**. عند استدعاء الدالة، يتمّ إنشاء ثلاثة متغيرات جديدة، هي **ivar** و**fvar** و**ch**، ويتمّ نسخ القيم إليها. يمكن للدالة أن تعدّل المتغيرات التي تنشئها، دون أن يؤثر ذلك المتغيرات في **main()**.

22- التمرير بالمرجع (passing by reference)

يمكن للدالة أن تعمل على المتغيرات الأصلية التابعة لـ **main()** وذلك من خلال استخدام المتغيرات المرجعية.

23- جمع قيم المتغيرات الجديدة (الكائنات):

إن العضو الدالي **add()** الدالة للصف **time1** يأخذ قيمتي أوقات كوسيطين، ويجمعهما ويضعهما في الكائن الذي استدعاه. بمعنى، إذا تمّ استدعاء **add()** وفق مايلي:

Obj3.add(obj1, obj2);

نجمع الكائن **obj1** مع الكائن **obj2** وتوضع النتيجة في الكائن **obj3**.

يمكن من استعمال وسيطاء من أنواع كائنات تمّ تعريفها من قبل المبرمج. ويمكن معاملة المتغيرات ذات النوع المعرف من قبل المستخدم إلى حد بعيد كالأنواع الأساسية في كل نواحي البرمجة في لغة البرمجة C++.

تجمع الدالة **add()** الدقائق من قيمتي **time1** ثم تجمع الساعات. إذا تخطى مجموع الدقائق القيمة 59، تضاف ساعة واحدة إلى المتغير **hours** ويطرح 60 من المتغير **minutes**. وإذا تخطى مجموع الساعات القيمة 23، يطرح 24 من **hours**. إن الكائنات التي تمّ تعرفها لها طابع عددي.

مثال:

```
// time.ccp
# include <iostream.h>
#include <conio.h>
#include <iomanip.h>
        class time1
        {
          private:
              int hours;        //0 --23
              int minutes;     // 0 --59
          public:
              void set( )
                {
                  char ch
                  cout<<"Enter time  (format  23:59 ):";
                  cin>> hours>>ch>> minutes;
                  }
              void display ( )
                  {
                    cout<< hours<<":"
                        <<setfill('0)
```

```
                    <<setw(2)<< minutes;
                }
            void  add (time1 obj1 ,  time1 obj2)
            {
            minutes = obj1. minutes + obj2. minutes ;
            hours = obj1.hours + obj2.hours;
            if (minutes > 59)
            {
             minutes = minutes  - 60 ;
             hours = hours + 1;
             }
            if (hours > 23)
            hours = hours- 24;
            }
    };
void main ( )
    {
        time1 obj1, obj2, obj3;  //create two variables (objects)
        char choice;
        do
         {
            cout<<"For obj1,";
            obj1.set( );
            cout<<"For obj2,";
            obj2.set( );
            obj3.add(obj1, obj2)
        cout<<"\n obj3 =";
        obj3.display( );              // display obj1
              cout<<"\n Do another (y/n)?";
               cin>> choice;
         } while (choice !='n')
```

24- الوصول إلى البيانات الخاصة:

لا تستطيع الدالة الرئيسة **main()** الوصول إلى متغيرات البيانات الخاصة مباشرة. فبما أنها خاصة، لا يمكن الوصول إليها إلا بوساطة الأعضاء الدالية التابعة للصف. لكن الأعضاء الدالية تستطيع الوصول إلى البيانات الخاصة التابعة ليس فقط للكائن الذي استدعاها بل التابعة لأي كائن أيضاً، شرط أن يكون ذلك الكائن تابعاً للصف نفسه. تكون بيانات الكائن خاصة بالنسبة للعالم الخارجي، لكن ليس بالنسبة للكائنات الأخرى التابعة للصف نفسه.

25- الارتقاء بالمتغيرات:

إن الارتقاء بالمتغيرات ، تلعب دوراً مهماً في برامج لغة البرمجة **C++**. تمكن لغة البرمجة **C++** من تحول الأنواع الأساسية من نوع إلى آخر تلقائياً.

- التركيب النحوي:

```
variable 1  = (Type) variable 2;
```

مثال:

```
int x; float y;
y = (float) x;
```

أي ارتقى نوع المتغير x من نوع صحيح إلى نوع حقيقي من النمط float.

```
# include <iostream>
#include <conio.h>
#include <math.h>
void main()
{
        clrscr();
        int x,y,m;
        float z;
        cout << "enter x=";
        cin >> x;
        y = abs(x);
```

```
        cout << "the abslute value is \n";
        cout << y;
        cout << endl;
        m = pow(x,5);
        cout << "The power of "<< x << "of 5 is " << endl;
        cout << m;
        cout << endl;
        cout <<"The square root of" << x << "is \n";
        if(x<0)
                cout << x<<" is negative";
        else
        {
                z = sqrt((float) x);
                cout << z;
        }
        getch();
}
```

يقوم المترجم في لغة البرمجة C++ بتحويل القيمة العددية الصحيحة إلى القيمة العائمة. تتمّ هكذا تحويلات بشكل تلقائي .إن روتينات التحويل مبنية بشكل داخلي في المترجم، وهو يعرف كيف يستعملها. لاتوجد روتينات ضمن المترجم لمعالجة التحول بين الأنواع التي تمّ تعريفها من قبل المبرمج ، لذلك يجب تعريف دوال للقيام بهذه التحويلات.

26- المتغيرات المرجعية

لقد سبق وأن عرفنا المتغيرات العامة global variable، وهي متغيرات يمكن استخدامها في أي مكان أو دالة خارج البرنامج وتعرف هذه المتغيرات مباشرة في تروبسة البرنامج وهذه المتغيرات تحجز مكاناً في الذاكرة ولا تنتهي من الذاكرة إلا إذا تمّ إغلاق البرنامج.

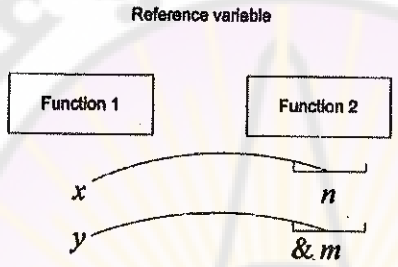
المتغيرات المحلية (الموضعية) **Local variable** وهي متغيرات موضعية تنشأ عن استخدامها الدالة التي عرفت فيها أي أنها تحجز عند ذلك مكاناً في الذاكرة وعند الانتهاء من تنفيذ هذه الدالة تدمر هذه المتغيرات.

المتغيرات الموضعية تعرف ضمن الدوال عند الحاجة إليها.

أما المتغيرات المرجعية فهي متغيرات تكون مرتبطة مع المتغير الأصلي وذلك من خلال استدعائه دالة معينة لتنفيذ معالجة على متغير ما.

إن المتغيرات التي تستقبل هذا المتغير في الدالة الجديدة هي المتغيرات المرجعية وأي تغيير يطرأ على هذه المتغيرات ينعكس على المتغير الأصلي.

- المخطط التدفقي للمتغيرات المرجعية



الشكل (4-15) المخطط التدفقي للمتغيرات المرجعية

مثلاً لو كانت قيمة x=5 و y=6 فإن القيمتين التي تأخذهما n=5 و &m=6.

ولكن عندما تتغير قيمة n وتأخذ قيماً مثل 4 3 2... لا تتغير قيمة x.

أما إذا تغيرت قيمة &m فإن ذلك التغيير يطرأ على y.

- التركيب النحوي للمتغيرات المرجعية:

Type of variable & variable name

ملاحظة:

لاسترجاع أكثر من قيمة نستخدم متغيرات مرجعية.

27- القيم المُعادة من الدوال:

إن الدوال من النوع العقيم **void**، لاتعيد أي قيمة. أما الدوال من أنواع أخرى

فإنها تعيد القيم حسب نوع الدالة ولكي تتمّ إعادة القيمة تحتاج الدالة لعبارة **return**.

- العبارة return:

لكي تعيد الدالة قيمة ما، عليك استعمال العبارة **return**. يمكن اتباع الكلمة الأساسية **return** بتعبير يؤدي تقييمه إلى القيمة المطلوب إعادتها.

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال عددين صحيحين وإيجاد مجموع هذين العددين باستخدام مفهوم الدوال.

الحل:

```
# include <iostream>
#include <conio.h>

void input(int &n)
{
        cout << "enter element =";
        cin >> n;
}
int sum (int n, int m)
{
        int z;
        z=m+n;
        return z;
}

void main()
{
        clrscr();
        int x,y,r;
        input (x);
        input (y);
        r = sum(x,y);
        cout << x << "+" << y << "=" << r;
        getch();
```

}

28- إنشاء متغيرات تلقائية:

يمكن تعريف واسناد قيمة للمتغير أثناء تعريفة. ويمكن احتساب تلك القيمة في العبارة نفسها.

```
Int x= y*60 +z;
```

إن تعريف المتغير واحتسابه في العبارة نفسها ليس أمراً مألوفاً في لغة البرمجة C، لكنه مألوف الاستعمال في لغة البرمجة C++.

عندما تعود الدالة إلى البرنامج الذي استدعاها، يتمّ تدمير المتغيرات التلقائية التي تمّ إنشاؤها في تلك الدالة، كـ x (لهذا السبب تمّ تسميتها متغيرات تلقائية؛ يتمّ إنشاؤها تلقائياً عند استدعاء الدالة ويتمّ تدميرها تلقائياً عند الانتهاء من الدالة). لقد تمّ في هذه الحالة إعادة قيمة x في الوقت المناسب ـ فحالما ينتهي تنفيذ العبارة **return** تعود الدالة ويتمّ تدمير x.

29- المكدس:

يتمّ تخزين المتغيرات التلقائية في قسم من ذاكرة الكمبيوتر يُعرف بـ المكدس **stack**. ينمو المكدس ويتقلص وفقاً لاستدعاء الدوال وعودتها. وللمكدس حجم أقصى، لذلك لا يمكن استعمال كميات ضخمة من البيانات التلقائية. عند إنشاء متغير تلقائي تكون قيمته "عشوائية"، على الأرجح ليست صفراً.

30- الإعادة بالقيمة:

إن عبارة **return** تعيد بالقيمة **by valuc**. أي أنه يتمّ إعادة القيمة الفعلية إلى البرنامج الذي استدعى الدالة.

31- الإعادة بالمرجع by reference:

يتلقى البرنامج الذي استدعى الدالة مرجعاً إلى المتغير الأصلي في الدالة.

32- الدوال التعاودية (reference function)

الدوال التعاودية هي الدوال التي تستدعي نفسها بنفسها ما دام شطر معين محقق أو غير محقق

مثال:

اكتب برنامجاً يمكن من إدخال عدد صحيح وإيجاد مضروب هذا العدد وذلك باستخدام الدالة التعاودية:

الحل:

```
# include <iostream>
#include <conio.h>

void input (int &);
long fact (int);
void main()
{
        //clrscr();
        int n;
        long m;
        input (n);
        m= fact(n);
        cout << n << "! = "<< m;
        getch();
}

void input(int &n)
{
        cout << "enter element =";
        cin >> n;
        while (n < 0)
        {
                cout << "input elemet=";
                cin >> n;
        }
}
```

```
long fact(int y)
{
        int z=y;
        if(y == 0 || y == 1)
                z=1;
        else
                z= z*fact(y-1);
        return z;
}
```

مثال:

اكتب البرنامج السابق بحيث تكون دالة إيجاد المضروب غير عودية.

```
# include <iostream>
#include <conio.h>

void input (int &);
long fact (int);
void main()
{
        clrscr();
        int n;
        long m;
        input (n);
        m= fact(n);
        cout << n << "! = "<< m;
        getch();
}

void input(int &n)
{
        cout << "enter element =";
        cin >> n;
        while (n < 0)
        {
                cout << "input elemet=";
                cin >> n;
        }
}
```

```
long fact(int y)
{
        int z=y;
        if(y == 0 || y == 1)
                z=1;
        else
                for (int i=y-1 ; i>=2; i--)
                {
                        z = z * i;
                }
        return z;
}
```

مثال:

اكتب برنامجاً يمكن من إدخال عددين صحيحين موجبين وأوجد حاصل ضرب هذين العددين باستخدام مفهوم التعاودية أي إيجاد قيمة الضرب باستخدام مفهوم الجمع لنذكر الحل أولاً بطريقة غير تعاودية:

الحل:

```
# include <iostream>
#include <conio.h>

void input (int &,int &);
int nuel(int,int);
void main()
{
        clrscr();
        int n,m;
        input (n,m);
        int t;
        t = nuel(n,m);
        cout << n << "* "<< m << "= " << t;
        getch();
}

void input(int &n,int &m)
{
```

```
        cout << "enter the first number =";
        cin >> n;
        while (n < 0)
        {
                cout << "enter the first number =";
                cin >> n;
        }
        cout << "enter the second number =";
        cin >> m;
        while (n < 0)
        {
                cout << "enter the second number =";
                cin >> m;
        }
}

int nuel (int s,int r)
{
        int a;
        if(s == 0 || r == 0)
        {
                a = 0;
        }
        else if (s == 1)
                a = r;
        else if (r == 1)
                a = s;
        else
        {
                a = 0;
                for (int i = 1; i <= s; i++)
                {
                        a += r;
                }
        }
        return a;
}
```

أما كتابة البرنامج باستخدام التعاودية فهو على الشاكلة:

```
# include <iostream>
#include <conio.h>

void input (int &,int &);
int nuel(int,int);
void main()
{
        clrscr();
        int n,m;
        input (n,m);
        int t;
        t = nuel(n,m);
        cout << n << "* "<< m << "= " << t;
        getch();
}

void input(int &n,int &m)
{
        cout << "enter the first number =";
        cin >> n;
        while (n < 0)
        {
                cout << "enter the first number =";
                cin >> n;
        }
        cout << "enter the second number =";
        cin >> m;
        while (n < 0)
        {
                cout << "enter the second number =";
                cin >> m;
        }
}

int nuel (int s,int r)
{
        int a;
        if(s == 0 || r == 0)
```

```
    {
        a = 0;
    }
    else if (s == 1)
        a = r;
    else if (r == 1)
        a = s;
    else
    {
        a = r + nuel(s-1,r);
    }
    return a;
}
```

تمارين

1. اكتب برنامجاً يقوم بطباعة الشكل وذلك باستخدام حلقة for

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```

2. اكتب برنامجاً بلغة C++ يمكن من رسم الشكل التالي. وذلك باستخدام حلقة for

```
1
3 3
5 5 5
7 7 7 7
9 9 9 9 9
7 7 7 7
5 5 5
3 3
1
```

3. اكتب بلغة C++ برنامجاً يمكن من إيجاد مجموعة من الأعداد الأولية المحصورة بين 1 و 100.

4. اكتب برنامجاً بلغة C++ يمكن من إدخال عدد صحيح وبيان فيما إذا كان هذا العدد تاماً أم لا.

5. اكتب برنامجاً بلغة C++ يمكن من إيجاد الأعداد التامة المحصورة بين 1 و 100.

6. اكتب برنامجاً بلغة C++ يمكن من إدخال عددين صحيحين موجبين (n,m) وإيجاد ما يلي.

   1. إيجاد مجموع قواسم العدد الأول (x).
   2. مجموع قواسم العدد الثاني (y).
   3. بين فيما إذا كان العددان x و y أولين فيما بينهما.
   4. حاصل جداء العدد الأول في الثاني. وإيجاد حاصل جداء القواسم وقارن بين النتيجتين.
   5. مجموع الأعداد الأولية بين n و m.
   6. مجموع الأعداد الأولية بين x و y.
   7. أوجد مجموع الأعداد الأولية بين الناتجين السابقين وأوجد أكبر هذين العددين (باستخدام التعاودية إن أمكن والدوال).

7. اكتب برنامجاً بلغة C++ يقوم بما يلي إدخال عدد صحيح موجب

   1. بيان فيما إذا كان العدد أولياً.
   2. بيان فيما إذا كان العدد تاماً.
   3. أوجد العدد الذي يسبق هذا العدد.

4. أوجد العدد الذي يلي هذا العدد. (وذلك باستخدام مفهوم الدوال (prototype) أي الطريقة الثانية.)

8. اكتب برنامجاً بلغة C++ يمكن من إدخال عددين صحيحين متمايزين مختلفين وإيجاد ما يلي:

1. مجموع الأعداد الأولية المحصورة بينهما.
2. مجموع الأعداد الفردية المحصورة بينهما.
3. مجموع الأعداد الزوجية المحصورة بينهما.
4. بيان فيما إذا كان هذين العددان أولين أم لا.

وذلك باستخدام مفهوم الدوال (prototype) علماً أن العددين موجبان.

9. اكتب برنامجاً يمكن من طباعة الشكل التالي

	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	9	12	15
4	4	8	12	16	20
5	5	10	15	20	25

10. اكتب برنامجاً بلغة C++ يمكن من إدخال عدد صحيح وإيجاد الجذر التربيعي وقيمة العدد بعد رفعه إلى القوة 5 والقيمة المطلقة له. (**ملاحظة:** نلاحظ عند البدء بالحل أن قيمة x صحيحة ولكن قيمة sqrt(x) قد تكون صحيحة أو حقيقية).

11. اكتب برنامجاً بلغة C++ يمكن من إدخال عدد صحيح موجب x وإيجاد جذر هذا العدد y وإيجاد cil(y) و floor.

12. اكتب برنامجاً بلغة C++ يمكن من إدخال عدد صحيح موجب x وإيجاد جذر هذا العدد y وإيجاد cil(y) و floor.

13. اكتب برنامجاً بلغة C++ يمكن من إدخال عدد صحيح بين 0 و 255 وإيجاد المحرف المقابل ويمكن لهذا البرنامج من إدخال محرف وإيجاد العدد المقابل له في النظام الثنائي والعشري والست عشري.

14. اكتب برنامجاً يمكن من إدخال عددين صحيحين موجبين وإيجاد حاصل ضرب العددين باستخدام مفهوم التعاودية أي إيجاد قيمة الضرب باستخدام مفهوم الجمع.

الفصل الخامس

المصفوفات والسلاسل

1- مقدمة:

تمكن المصفوفات من تخزين بيانات أساسية من النوع نفسه حسب الحاجة البرمجية، ويمكن أن تكون هذه البيانات مركبة. كما أن سلسلة النص، في لغة البرمجة C++ تتمّ معاملتها كمصفوفة من المحارف.

يمكن تجميع متغيرات من أنواع مختلفة في آلية تخزين أخرى تسمى بنية (structure)، وهناك نوع البيانات التعدادي (enumerated)، وهو نوع من الوسائل البسيطة لإنشاء أنواع بيانات خاصة.

2- أساسيات المصفوفات:

المصفوفة هي وسيلة لتجميع عدد من المتغيرات من النوع نفسه. يمكن الوصول إلى المتغيرات المخزنة في المصفوفة باستخدام الفهرس **index number**. وبالتالي يمكن التنقل في المصفوفة بسهولة، فيتمّ الوصول إلى جميع المتغيرات.

- تعريف المصفوفة:

يحجز المترجم مساحة تخزين لعدد معين من متغيرات البيانات من نوع معين. باسم المصفوفة.

- التركيب النحوي لتعريف المصفوفة أحادية البعد:

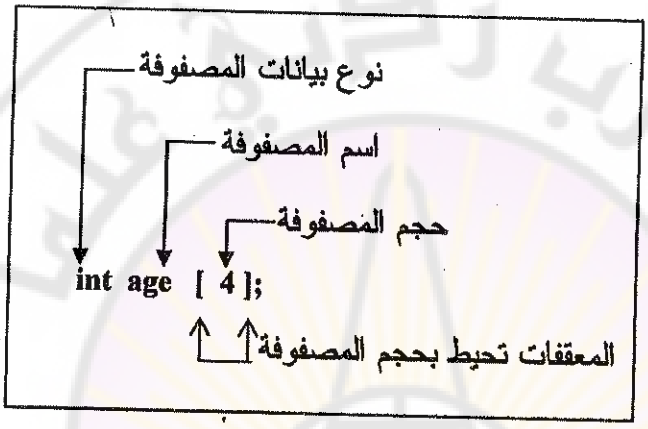
type name of array [number of element];

مثال:

تعريف مصفوفة تحتوي أربعة عناصر من نوع أعداد صحيحة. ولتكن المصفوفة **age**.

int age[4];

تحدد الكلمة **int** نوع البيانات التي سيتمّ تخزينها في المصفوفة، و**age** هو اسم المصفوفة و4 هو حجمها، أي العدد الأقصى للمتغيرات الصحيحة التي سيتمّ تخزينها في المصفوفة. والقوسان [ ] يحيطان بالحجم، الأقواس هي التي تبلغ المترجم بتعريف مصفوفة.



الشكل (5-1) التركيب النحوي لتعريف المصفوفة أحادية البعد

يمكن للمبرمج تعريف مصفوفة من أنواع من البيانات أنشأها المبرمج باستعمال الصفوف. يمكن أن يكون نوع المصفوفة أي نوع من الكائنات، سواء كانت تتصرف كنوع بيانات أم لا، لذلك يمكن إنشاء مصفوفة من الكائنات تمثل كائنات فعلية، وليست أنواع بيانات.

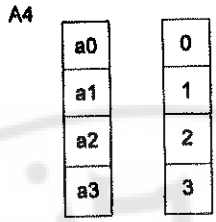
- عناصر المصفوفة:

إن كل متغير في المصفوفة يدعى عنصراً **element**. والعناصر مرقمة وفق فهرس. إن دليل العنصر الأول هو 0، والثاني هو 1، الخ. إذا كان حجم المصفوفة **n**، فإن دليل العنصر الأخير هو **n-1**.

مثال:

لتكن المصفوفة أحادية البعد

A[4]={23,44,3,98}



الشكل (2) دليل عناصر المصفوفة أحادية البعد

- الوصول إلى عناصر المصفوفة أحادية البعد:

يتمّ الوصول إلى عناصر المصفوفة باستخدام دليلها واسم المصفوفة واستخدام [ ].

لكن في سياق مختلف عما هي عليه في تعريف المصفوفة.

A[2] = 3;

مما سيؤدي إلى ضبط قيمة العنصر A[2]، والعبارة

Cout << A[3];

مما سيؤدي إلى عرض قيمة العنصر A[3]

إن العبارة

int A[4];

تعرّف مصفوفة من أربعة عناصر، لكن التعبير

A[2]

يشير إلى العنصر الثالث في المصفوفة.

القوة الفعالة للمصفوفات تأتي من أنها تمكن من استعمال متغير، بدلاً من ثابت.

```
int A[4];
.
for (i=0 ; i<4 ; i++)
cout <<A[i] <<endl;
```

الحلقة for تتنقل بين قيم j من 0 إلى 3، وهي قيم فهرس المصفوفة الضرورية للوصول إلى كل عنصر. إن التعبير i<4 يستعمل حجم المصفوفة كحد له.

إن الخرج يمكن أن يكون

```
23
44
3
98
```

وفقاً للقيم المخزنة في المصفوفة.

3- المصفوفة الثابتة أحادية البعد:

تعريف المصفوفة الثابتة ضمن البرنامج.

مثال:

```
A[4]={23,44,3,98}
```

تعريف لائحة القيم المصفوفة، ويتمّ الفصل بين القيم بفواصل، وتحيط الأقواس الحاصرة تلك اللائحة. يمكن عرض هذه القيم بواسطة حلقة for:

```
for (i=0 ; i<4 ; i++)
cout<<A[i] <<" ";
```

نحصل على الخرج التالي:

```
23 44 3 98
```

ملاحظة:

يمكن تعريف المصفوفة الثابتة دون ذكر حجم المصفوفة، فالمترجم يحتسب عدد القيم ويستعمل ذلك العدد كحجم للمصفوفة.

```
A[ ]={23,44,3,98}
```

ملاحظة:

إذا كان عدد عناصر المصفوفة الثابتة أكثر من حجم المصفوفة المحدد، يعترض المترجم. وإذا كانت أقل، سيملأ المترجم بقية العناصر بأصفار.

4- العمليات على المصفوفة أحادية البعد

يتمّ جمع المصفوفة أحادية البعد وفق المبدأ الجبري جمع كل عنصر مع مقابله أي يجب أن يكون للمصفوفتين البعد نفسه والنوع نفسه.

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال مصفوفتين A,B من البعد 1*5 يحوي كل منهما خمسة عناصر وإيجاد ما يلي.

A+B A-B A*B

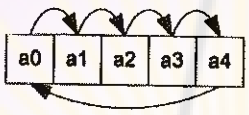
لإيجاد ناتج الضرب نقوم بضرب A بمنقول B.

```
# include <iostream>
#include <conio.h>

void main()
{
        //clrscr();
        int A[5],B[5],C[5],D[5];
        int m,i;
        for ( i =0 ; i < 5; i++)
        {
                cout << "enter element";
                cin >> A[i];
        }
        for (i = 0 ; i < 5; i++)
        {
                cout << "enter element";
                cin >> B[i];
        }
        m=0;
        for (i = 0; i < 5; i++)
        {
                D[i]= A[i]+B[i];
                C[i]= A[i]-B[i];
                m += A[i]*B[i];
        }
        for (i = 0; i < 5; i++)
```

```
            cout << "  "<<D[i] ;
        cout << endl;
        for (i = 0; i < 5; i++)
            cout << "  "<<C[i] ;
        cout << endl;
        cout <<  m;
        getch();
}
```

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال مصفوفة مكونة من 5 عناصر ويوجد ما يلي.



إزاحة العناصر بالشكل

$$a_0 \leftarrow a_4 \leftarrow a_3 \leftarrow a_2 \leftarrow a_1 \leftarrow a_0$$

الحل:

```
# include <iostream>
#include <conio.h>
void main()
{
        //clrscr();
        int m,t,i,A[5];
        for ( i =0 ; i < 5; i++)
        {
            cout << "enter element";
            cin >> A[i];
        }
        m = A[4];
        for (i=4; i >= 0; i--)
        {
            A[i]=A[i-1];
```

```
    }
    A[0]=m;
    for (i=0; i< 5; i++)
        cout << " "<<A[i];
    getch();
}
```

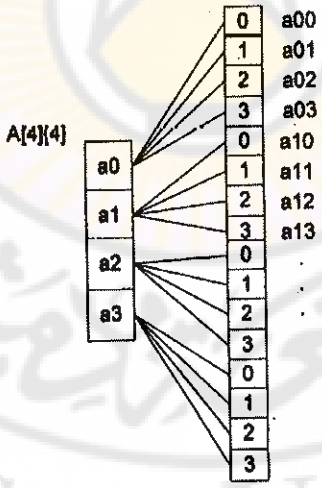
اكتب البرنامج السابق بحيث تكون هذه الإزاحات مدخلة من قبل المستخدم.

5- المصفوفات المتعددة الأبعاد:

تمكن لغة البرمجة C++ من تعريف مصفوفات متعددة الأبعاد، ويمكن أن يكون كل بعد بحجم مختلف. إن كل بعد محاط بقوسين.

- مصفوفات ثنائية البعد

المصفوفة ثنائية البعد هي عبارة عن مجموعة من البيانات (المتغيرات من نوع معين) المرتبة وفق اسطر وأعمدة. جميع عناصر المصفوفة عادة تكون من النوع نفسه. عناصر المصفوفة إما محارف أو أعداداً صحيحة، أعداداً حقيقية، أعداد منطقية، متغيرات من نوع يحددها ويعرفها المستخدم.



الشكل (5-2) تمثيل المصفوفة ثنائية البعد في الذاكرة

- التركيب النحوي لتعريف مصفوفة

name of array [num of row] [number of column];

يتمّ الوصول إلى عناصر المصفوفة باستخدام دليلين. لعرض العناصر الموجودة في هذه المصفوفة، نستخدم حلقتين **for** متداخلتين.

يمكن أن نعتبر المصفوفة ثنائية البعد كمصفوفة مصفوفات. المصفوفة A[4][3] هي مصفوفة من أربع مصفوفات فرعية، كل واحدة منها تحتوي على ثلاثة عناصر.

- المصفوفة الثابتة ثنائية البعد:

تعريف المصفوفة الثابتة ثنائية البعد بوساطة لائحة مكونة من عدة لوائح ، حيث يتمّ تعريف كل مصفوفة فرعية بلائحة أرقام خاصة بها، وتكون هذه اللوائح مفصولة عن بعضها بعضاً بوساطة فواصل.

مثال:

```
float B[4][3]={{2.6 , 0.3 , 11.3}, {11.0 , 21.3 , 33.4 } , {6.6 , 8.9 , 9.0}
                ,{1.1 , 2.2 , 3.1}}
```

نستخدم الحلقات **for** المتداخلة لعرض محتويات الصفوفة.

```
# include <iostream>
#include <conio.h>

void main()
{
      //clrscr();
float B[4][3]={{2.6 , 0.3 , 11.3}, {11.0 , 21.3 , 33.4 } ,
            {6.6 , 8.9 , 9.0} ,{1.1 , 2.2 ,  3.1}};
      for ( int i =0 ; i < 2; i++)
      {
            for (int j=0; j<2; j++)
                  cout << B[i][j];
      }
      getch();
}
```

سيكون الخرج:

```
2.6   0.3   11.3
11.0  21.3  33.4
6.6   8.9   9.0
1.1   2.2   3.1
```

- العمليات الحسابية المعرفة على المصفوفات ثنائية البعد:

تدخل عناصر المصفوفة ثنائية البعد بوساطة استخدام حلقات for المتداخلة ، ويمكن تنفيذ جميع العمليات المعرفة على المصفوفات باستخدام لغة البرمجة C++، كعملية جمع وطرح مصفوفتين وإيجاد منقول مصفوفة وإيجاد محدد مصفوفة وإيجاد حاصل ضرب مصفوفتين.

مثال:

int A[3][5];

اكتب برنامجاً يمكن من إدخال مصفوفة بعدها 2×2.

```
# include <iostream>
#include <conio.h>

void main()
{
        //clrscr();
        int A[2][2];
        for ( int i =0 ; i < 2; i++)
        {
                for (int j=0; j<2; j++)
                {
                        cout << "enter element ";
                        cout << "A"<<"["<<i<<"]["<<j<<"]=";
                        cin >> A[i][j];
                }
        }
        for ( int i =0 ; i < 2; i++)
        {
```

```
            for (int j=0; j<2; j++)
                cout << A[i][j];
        }
        getch();
    }
```

مثال:

اكتب برنامجاً بلغة C++ يمكن مما يلي

1. إدخال مصفوفتين 3×3.
2. إيجاد حاصل جمع المصفوفتين.
3. بيان فيما إذا كان ناتج مصفوفة متناظرة أم لا.
4. إيجاد حاصل ضرب المصفوفتين.
5. إيجاد مجموع عناصر القطر الثانوي في المصفوفة الناتجة عن حاصل الضرب.
6. إيجاد مجموع العناصر التي تقع فوق القطر الثانوي في المصفوفة الناتجة عن حاصل الضرب.
7. إيجاد مجموع العناصر التي تقع تحت القطر الرئيسي للمصفوفة الناتجة عن حاصل الضرب.
8. ترتيب عناصر القطر الرئيسي ترتيب تصاعدي.
9. ترتيب عناصر القطر الثانوي ترتيب تنازلي
10. إيجاد العنصر الأكبر في المصفوفة وعدد تكرار هذا العنصر وأماكن تواجده.

الحل:

```
# include <iostream>
```

```
#include <conio.h>
using namespace std;

void main()
{
	//clrscr();
	int A[3][3],B[3][3],C[3][3],D[3][3];
	//--------------reading A----------------------------
	for ( int i =0 ; i < 3; i++)
	{
		for (int j=0; j<3; j++)
		{
			cout << "enter element ";
			cout << "A"<<"["<<i<<"]["<<j<<"]=";
			cin >> A[i][j];
		}
	}
	//--------------reading B----------------------------
	for ( int i =0 ; i < 3; i++)
	{
		for (int j=0; j<3; j++)
		{
			cout << "enter element ";
			cout << "B"<<"["<<i<<"]["<<j<<"]=";
			cin >> B[i][j];
		}
	}
	//---------------the sum ------------------------
	for ( int i =0 ; i < 3; i++)
	{
		for (int j=0; j<3; j++)
		{
			C[i][j] = A[i][j]+B[i][j];
			cout << C[i][j];
		}
		cout << endl;
	}
	//--------------الجمع عن الناتجة المصفوفة تناظر من التحقق
	int t =1;
```

```
for ( int i =0 ; i < 3; i++)
        for (int j=0; j<3; j++)
                if (C[i][j] != C[j][i])
                {
                        t = 0;
                        break;
                }
        if (t == 0)
                cout << "the matrix is not sym";
        else
                cout << "the matrix is sym";
//----------المصفوفتان ضرب حاصل إيجاد----------
for ( int i =0 ; i < 3; i++)
{
        for (int j=0; j<3; j++)
        {
                D[i][j] = 0;
                for (int k = 0; k < 3; k++)
   D[i][j] += A[i][k]*B[k][j];
        }
}
for ( int i =0 ; i < 3; i++)
{
        for (int j=0; j<3; j++)
                cout << "  "<<D[i][j];
        cout << endl;

}
//----------مجموع عناصر القطر الثانوي----------
int sum1 = 0;
for ( int i =0 ; i < 3; i++)
        for (int j=0; j<3; j++)
                if (i+j == 2)
                        sum1 += D[i][j];
cout << "sum1 = "<< sum1 ;
cout << endl;
//----------مجموع العناصر فوق وتحت القطر الثانوي----------
```

```
int s2 = 0,s3 = 0;
for ( int i =0 ; i < 3; i++)
        for (int j=0; j<3; j++)
        {
                if (i+j <= 2)
                        s2 += D[i][j];
                if (i+j > 2)
                        s3 += D[i][j];
        }
cout << "sum2 = "<< s2 ;
cout << endl;
cout << "sum3 = "<< s3;
cout << endl;
//----------إيجاد مجموع عناصر القطر الرئيسي------
int s4 = 0;
for ( int i =0 ; i < 3; i++)
        s4 += D[i][i];
cout << "Mainer = " << s4;
cout << endl;
//----------مجموع العناصر فوق وتحت القطر الرئيسي---------

int s5=0 ,s6=0;
for ( int i =0 ; i < 3; i++)
        for (int j=0; j<3; j++)
        {
                if (i<j)
                        s5 += D[i][j];
                if (i>j)
                        s6 +=D[i][j];
        }
cout << "s5 = "<< s5 ;
cout << endl;
cout << "s6 = "<< s6;
cout << endl;

//----------ترتيب عناصر القطر الثانوي ترتيباً تصاعدياً---------
int v[3];
for ( int i =0 ; i < 3; i++)
```

```
        for (int j=0; j<3; j++)
                if (i+j == 2)
                        v[i] = D[i][j];
int c;
for ( int i =0 ; i < 3; i++)
        for (int j=0; j<3; j++)
                if (v[i]>v[j])
                {
                        c = v[i];
                        v[i] = v[j];
                        v[j]=c;
                }
for ( int i =0 ; i < 3; i++)
{
        for (int j=0; j<3; j++)
        {
                if (i+j == 2)
                        D[i][j] = v[i];
                cout << "  "<<D[i][j];
        }
        cout << endl;
}
//----------إيجاد العنصر الأكبر في المصفوفة----------
int max = D[0][0];
for ( int i =0 ; i < 3; i++)
        for (int j=0; j<3; j++)
                if(max < D[i][j])
                        max = D[i][j];
cout << " max = " << max;
int cont = 0;
for ( int i =0 ; i < 3; i++)
        for (int j=0; j<3; j++)
                if (max == D[i][j])
                {
                        cont += 1;
                        cout << "  "<<i<<","<<j;
                }
                cout << "\n time of max = "<<cont;
                cout << endl;
```

```
getch();

}
```

6- المصفوفة الثلاثية الأبعاد

تعريف المصفوفة الثلاثية الأبعاد بواسطة ثلاثة أبعاد محصورة بين قوسين.

```
int C[3][4][2];
```

يتمّ الوصول إلى كل عناصرها بوساطة التعبير C[1][2][3]. يمكن تعريف مصفوفات ذات أبعاد أكثر والوصول إلى عناصرها بطريقة مماثلة.

7- المصفوفات كأعضاء بيانية:

يمكن استخدام المصفوفات كأعضاء بيانية في الصفوف. سنستخدم المصفوفة كسلسلة محارف وكأنها مصفوفة من المحارف المنفصلة وسندخل كل محرف ونخرجه بشكل مستقل عن البقية.

مثال:

```
// string.cpp
# include <iostream.h>
#include <conio.h>
        class string1
        {
          private:
              char name[20]
              int n;
              int serial_number;
          public:
              void inpu( )
                 {
                    char ch;
                     n=0;
```

```
        cout<<"Enter name:";
      do
        {
            ch= getche( );
            name[n] = ch;
              ++n;
              } while (ch != '\r');
          cout<<"\n Enter serial_number:";

          cin>> serial_number;
          }
      void output ( )
          {
            cout<< " Name =";
              for (int i=0 ; I ,n ; ++i)
      cout << name[i];
       cout<<"\n  serial number = " << serial_number;
            }
};
void main ( )
{
    string1 obj1, obj2;     //create two variables (objects)
      cout<<"Enter string1 1 data "<< endl;
      obj1.input( );
      cout<<"Enter string1 2 data "<< endl;
      obj2.input( );
      cout<<"\n  string1 1  "<< endl;
      obj1.output( );
      cout<<"\n string1 2  "<< endl;
      obj2.output( );
}
```

عند تنفيذ البرنامج ستظهر رسائل تحذير من المترجم، كـ

Functions containing do are not expanded inline و **Functions containing for are not expanded inline**

وسبب هذه الرسالة وجود حلقات **for** ضمن الصف وهذا لايؤثر على تنفيذ البرنامج. ينشئ البرنامج في **main()** كائنين **obj1, obj2**، ويحصل على البيانات من المستخدم لكل واحد منهما، ثم يعرض تلك البيانات.

8- الدالة المكتبية getche( ):

تنتظر هذه الدالة إلى أن نضغط مفتاحاً ما على لوحة المفاتيح ثم تعود مع رقم الآسكي **ASCII** الخاص بذلك المفتاح. لا حاجة لضغط **Enter** من أجل الحصول على المحرف. تتطلب الدالة **getche()** شمل ملف الترويسة **conio.h** في البرنامج.

مثال:

سنستخدم الدالة **getche( )** في حلقة **do**.

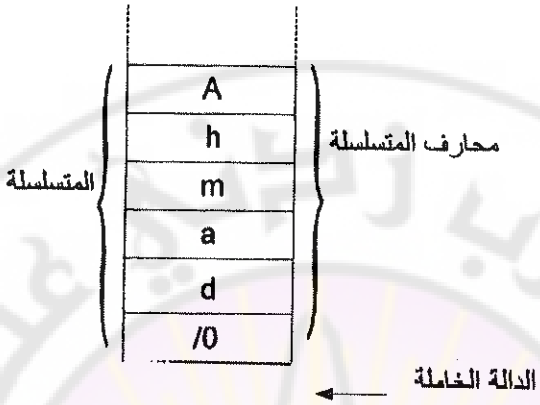
```
do
{
ch = getche( );
name[i] = ch ;          //storig
++i;                    // increment
} while (ch !='\n');
```

في كل دورة في الحلقة تعود الدالة **getche()** مع محرف جديد، يتمّ إسناده إلى عنصر في المصفوفة. يبدأ الفهرس من 0 ويتزايد كل دورة في الحلقة بواسطة العبارة **i++** . إن المفتاح **Enter** يقابل الرقم 13 في النظام العشري في جدول الآسكي **ASCII** ، إن دالة الهروب **\n** تقابل الرقم 13 في النظام العشري في جدول الآسكي **ASCII** . عند إدخال مفتاح **Enter** فإن البرنامج يخرج من الحلقة **do**.

لإخراج عناصر المصفوفة "أي عرض الاسم" نستخدم حلقة **for**، وهي عملية عكس عملية التخزين. فبذلك نعرض عناصر المصفوفة محرفاً محرفاً.

```
for ( ;; t i- . , .      ++)
  cout<<name[i];
```

يعرف البرنامج عدد عناصر المصفوفة المدخلة بواسطة المتغير n، وهو عضو بياني في الصف، ويمكن الوصول إليه من كل الأعضاء الدالية.



الشكل (5-3) سلسلة محارف مخزنة في متغير متسلسلة محارف

9- معامل التزايد اللاحق:

عندما نخزن المحارف في المصفوفة **name[]** يبدأ الفهرس بالقيمة 0، ثم زيادته مع إضافة كل محرف.

```
name[i] = ch ;
++i;
```

يمكن للفهرس أن يزداد من ضمن العبارة نفسها التي يتمّ استخدامها كدليل للمصفوفة.

```
name[++i] = ch;
```

وهذا التعبير هو تركيب نحوي قانوني بالنسبة للمترجم. لكن هناك مشكلة. يبدأ الفهرس بالصفر وبالتالي يجب وضع المحرف الأول في عنصر المصفوفة الذي دليلة 0. لذلك نزيد المتغير i, بعد وضع محتويات المتغير **ch** في المصفوفة.

```
name[i++] = ch;
```

عندما يأتي المعامل ++ بعد المتغير، فإنه يسمى معامل تزايد لاحق **(postfix)**. وعندما يأتي قبل المتغير، فإنه يسمى معامل تزايد متصدر **(prefix)**. يتمّ تطبيق

المعامل المتصدر قبل استعمال المتغير، بينما يتمّ تطبيق المعامل اللاحق بعد استعمال قيمة المتغير. أي:

```
++n ;          // Prefix operator , imcrement before being used
i++;           // postfix operator , imcrement after being used
```

وبالتالي تصبح حلقة **do** باستعمال المعامل اللاحق كما يلي:

```
do
{
ch = getche( );
name[i++] = ch ;
} while (ch !='\n');
```

ملاحظة:

بالطريقة نفسها نعرف معامل التناقص اللاحق (n--) معامل التناقص متصدر (--n)

10- المكدس Stack:

المصفوفة هي طريقة لتخزين البيانات، لكنها غير مناسبة في العديد من الحالات. توجد مهمة رئيسة في البرمجة وهي إنشاء بنيات تخزين كالقوائم المرتبطة والمكدسات وصفوف الانتظار والقواميس والمتجهات (vectore). ولكل نوع من بنيات التخزين هذه حسناتها وسيئاتها. بعض الأساليب يسهل الوصول إلى عضو البيانات بشكل عشوائي أسرع أو إضافة عضو البيانات أسهل، بينما مع بعضها الآخر يكون من الأسهل البحث عن البيانات أو فرزها. المكدس هو بنية تخزين للبيانات مناسبة عندما نريد الوصول أولاً إلى أحداث عضو بيانات تمّ تخزينه. وتسمى هذه البنية "LIFO" وهي اختصار للعبارة **Last In First Out** أي الداخل آخراً هو الخارج أولاً.

المكدسات مفيدة في العديد من حالات البرمجة للمسائل الجبرية .

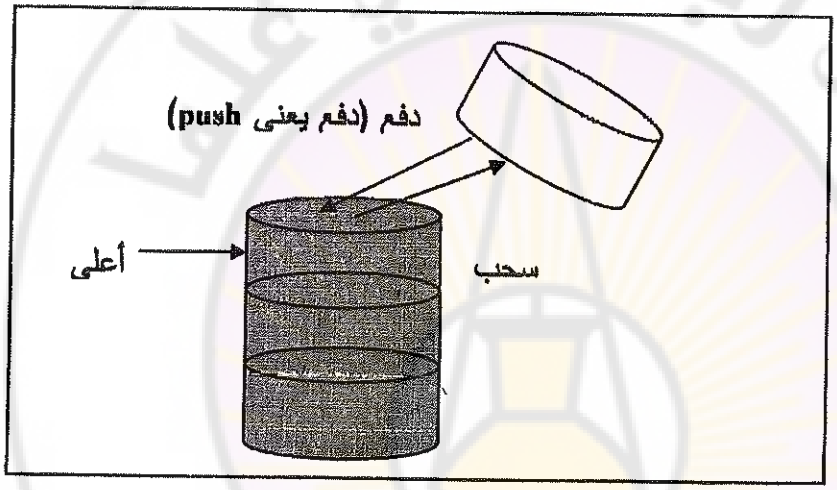
مثال:

2*x+4*(3+y)

يكون المكدس ملائماً لتخزين النتائج المرحلية.

- **الدفع والسحب:**

تمكن المكدسات من تخزين أي نوع من البيانات، من الأنواع الأساسية وتمكن من تخزين كائنات الصفوف المعقدة. لكن يخزن كل مكدس نوعاً واحداً من البيانات، **int** أو **float** أو أي شيء آخر، لكن ليس خليطاً من الأنواع. عندما نضع قيمة في المكدس، فإننا نقوم بعملية دفع (**push**)، وعندما نخرج القيمة من المكدس فإننا نقوم بعملية سحب (**pop**).



الشكل (5-4) الدفع في المكدس والسحب من المكدس

في المثال مصفوفة، وهي عضو بياني دالة للصف **stack**، من أجل تخزين كمية من الأعداد الصحيحة. فيما يلي البرنامج **stack1** الذي يحدد الصف **Stack** ثم يختبرها بإنشاء مكدس، ثم بدفع ثلاثة أعداد صحيحة فيه، ثم بسحبها منه وعرضها.

مثال:

```
// stack1.cpp
# include <iostream.h>
# include <conio.h>
struct stack
{
 private :
```

```
    int st[20];
     int top;
  public :
        stack ( ) : top (-1);              // constructor
                {     }
      void push (int var  )            // place an item on the stak
           {
             st[++st.top] = var ;
             }
       int pop( )          // remove an item from the stak
            {
              return st[st.top--];
              }
};
void main ( )
       {
          Stack s1;                    // Objects
           s1.push(11);
           s1.push(12);
           s1.push(13);
           cout<<s1.pop( ) <<endl;
           cout<<s1.pop( ) <<endl;
           cout<<s1.pop( ) <<endl;
       }
```

11- مصفوفات من الكائنات:

يمكن أن يكون هناك كائنات تحتوي على مصفوفات، ويمكن أن يكون هناك مصفوفات عناصرها كائنات أيضاً. قد نحتاج في أغلب الأحيان معاملة عدد كبير من الكائنات التابعة للصف نفسه بطريقة مماثلة. إن وضع الكائنات في مصفوفة واستعمال حلقة على عضو دالي هي طريقة بسيطة لتنفيذ هذه المهمة.

- تعريف مصفوفة كائنات:

إن التركيب النحوي لتعريف مصفوفة كائنات هو نفسه المستعمل لتعريف مصفوفة من أي نوع أساسي. نكتب مواصفات الصف في البرنامج قبل أن نحاول تعريف أي مصفوفة كائنات.

إذا أنشأنا مواصفات الصف NumberClass، عندئذٍ يمكن تعريف مصفوفة كائنات دالة للصف، ولتكن المصفوفة Numbermat، وفق مايلي:

```
NumberClass Numbermat [5];    // array of 5 NumberClass
objects
```

ويتمّ تخزين بيانات كل كائن في المصفوفة بشكل متجاور في الذاكرة. بحيث أن الصف معرفة بالشكل التالي:

مثال:

```
class NumberClass
{
private:
      int x;
      float y;
public:
      void funct ( )
         {
           // statements;
         }
      // member function
};
void main ( )
  {
    NumberClass Numbermat [5];
     // statements;
   }
```

كل عنصر في المصفوفة Numbermat، هو كائن دالة للصف

.NumberClass

- التركيب النحوي للوصول عناصر المصفوفة الكائنات:

يمكن الوصول إلى الأعضاء الدالية للكائنات المخزنة في المصفوفة. وفق مايلي:

Numbermat [2] . funct ( ); // accesses funct ( ) for 3rd elemnt in Numbermat

هذا التركيب النحوي يصل إلى هدفه مباشرة. إنّ معامل الدليل ([ ]) له أولوية أعلى من معامل النقطة.

مثال:

اكتب برنامجاً بلغة C++ يمكن من تعريف مصفوفة كائنات، ثم يحصل على بيانات لعدد من الكائنات ، وعندما ينتهي المستخدم من إدخال البيانات، يخرج البرنامج كل البيانات.

```
        // arrayobj.cpp
        // creates array of object
// time.ccp
# include <iostream.h>
#include <conio.h>
#include <iomanip.h>
          class time1
          {
            private:
                int hours;          //0 --23
                int minutes;     // 0 --59
            public:
                void set( )
                  {
                    char ch
                    cout<<"Enter time (format 23:59 ):";
                     cin>> hours>>ch>> minutes;
                     }
```

```
        void display ( )
            {
              cout<< hours<<":"
                  <<setfill('0') <<setw(2)<< minutes;
              }
    };
void main ( )
    {
        time1 timearray[5];          //array of 5 time1 objects
         int i= o;
        char choice;
        do
         {
            cout<<"time1"<<n<<".";
            timearray[i++].set( );   /insert data into object in
array
            cout<<" Do another (y/n) ?";
            cin>> choice;
          } while (choice !='n')
      for (int j = 0; j<i ; j++)
      {
      cout<< " \n time1 "<< j << " =";
      timearray[j] . display ( );   // display data from object in
array
      }
  }
```

العبارت المهمة في البرنامج هي:

```
timearray[i++].set( ); //insert data into object in array
```

التي تستدعي العضو الدالي set() لكي يستطيع المستخدم إدخال بيانات للكائن المخزن في المصفوفة عند الدليل i . والعبارة

```
timearray[5] . display ( );  // display data from object in array
```

التي تستدعي العضو الدالي **display()** لإخراج البيانات من الكائن المخزن عند الدليل **j**.

ملاحظة:

إن المشكلة مع المصفوفات هي أنه يجب تعريف المصفوفة بحيث نستطيع تخزين أكبر قدر ممكن من الكائنات التي نتوقع أن ننشئها ، مما يؤدي إلى ضياع في مساحة الذاكرة. تمكن القوائم المرتبطة، وهي مصفوفة مؤشرات إلى الكائنات، تخفيض الضياع في الذاكرة.

12- سلاسل المحارف:

تمكن لغة البرمجة C++ من معالجة النص كسلسلة (**string**)، وهي مجموعة من الأحرف تنتهي بحرف خاص. رغم أن سلاسل المحارف مفهوم قديمة إلا أنها ميزة مهمة في لغة البرمجة C وفي لغة البرمجة C++، وغالباً ما تشكل أساساً لصفوف سلاسل المحارف.

- متغيرات متسلسلة المحارف:

السلسلة هي مجموعة من المحارف للمحرف الأخير فيها قيمة رقمية وهي 0 (صفر). وبصفته محرفاً، يمكن تمثيل هذه القيمة بدالة تحكم 0\. يسمى هذا المحرف بالمحرف الخامل **null**. إن استعمال قيمة خاصة كهذه للإشارة إلى نهاية سلسلة محارف النص يعني عدم تخزين طول النص في متغير عددي صحيح منفصل، تبحث دوال معالجة سلاسل المحارف عن المحرف الخامد 0\ من أجل معرفة مكان انتهاء سلسلة المحارف. إن متغير متسلسلة المحارف هو مصفوفة من النوع **char**. قد يحتوي هذا المتغير أو لا يحتوي على قيمة في وقت ما.

- تعريف متغير متسلسلة المحارف:

```
char str[50];    //string variable
```

عند تعريف متغير متسلسلة المحارف لا يتمّ تخزين أي قيمة أولية فيه. خلافاً

للمتغيرات ذات الأنواع الأساسية، يمكن أن تكون متغيرات متسلسلة محارف بأحجام مختلفة. هذا المتغير مثلاً يستطيع تخزين سلسلة تصل إلى 80 حرفاً. رغم أنها في الواقع مصفوفات من النوع **char**، إلا أن في لغة البرمجة C++ تمكن من التعامل مع سلاسل المحارف على أنها نوع ما كنوع بيانات أساسي. وتمكن الدالتان **cin** و**cout** من استعمال عمليات دفق الدخل / الخرج الاعتيادية لدخل سلسلة أو خرجها بواسطة عبارة واحدة.

مثال:

```
char str[50];    //string variable
cin>>str;        // get text from uscr
cout<<str;       //display text
```

يكتب المستخدم محارف السلسلة ثم يضغط **Enter**.

- متسلسلة محارف الثابتة:

يمكن أن نسند لمتغير متسلسلة المحارف قيمة ما عند تعريفه. وتعرف متسلسلة المحارف الثابتة وفق مايلي:

```
char name[10] = {'a' , 'h' , 'm' , 'a' , 'd' , '\0' }
```

ويمكن أن نعرف متسلسلة المحارف الثابتة أيضاً بالشكل التالي:

```
char name[10] = " ahmad ";
```

إن مجموعة من الأحرف تحيطها علامات اقتباس مزدوجة كهذه المبينة هنا تسمى ثابت متسلسلة محارف **string constant**. إنّ الحرف الخامد، (\0) مشمول على أنه المحرف الأخير في الثابت المتسلسلة محارف **ahmad** . لقد أضافه المترجم بشكل تلقائي. لذا، وبالرغم من أن هناك خمسة محارف فقط في الاسم **ahmad**، إلا أن ثابت متسلسلة المحارف " **ahmad** " يحتوي على ستة محارف: المحارف الخمسة والمحرف \0. لذلك عند تعريف المصفوفة يجب أن تكون كبيرة كفاية لتخزين أقصى عدد ممكن من المحارف إضافة إلى مكان واحد للحرف الخامد.

ملاحظة:

يمكن أن نعرف صفوفة متسلسة المحارف الثابتة، من غير تحديد عدد العناصر في متغير متسلسلة المحارف. أي:

```
char  name[ ] = " ahmad ";
```

يجعل المترجم متسلسلة المحارف الثابتة **[]name** بالطول المناسب لتخزين ثابت متسلسلة المحارف بما في ذلك الحرف الخامل.

- عمليات دخل / خرج سلاسل المحارف:

- الدالة get() :

عند استخدام الدالة **cin** لإدخال سلسلة المحارف، مع العامل <<، فعند إدخال فراغاً (المحرف ' ') ، يتوقف **cin** عن قراءة الدخل.

مثال:

```
char str[50];
cin>>str;
cout<<str;
```

عند إدخال سلسلة المحارف التالية:

```
Ahmad is student
```

إن الخرج سيكون كما يلي:

```
Ahmad
```

لقد تمّ تخزين الكلمة **Ahmad** فقط في السلسلة **str**. أما بقية المتسلسلة فلم تخزن. لذلك نستخدم العضو الدالي **cin.get()** الذي يمكن من إدخال سلسلة محارف تحتوي فراغات . تحتاج هذه الدالة لشمل الترويسة **istream.h**.

مثال:

```
char str[50];
cin.get (str, 50 );   ;        // get text from user
cout<<str;                     //display text
```

إن دالة الإدخال cin.get() تعالج الفراغات أيضاً، وتضمن أن المستخدم لن يجعل مصفوفة سلسلة المحارف تفيض.

- الدالة ignore( ):

تمكن الدالة من تجاهل السطر الجديد،

مثال:

```
const int size = 50;        //constant variable , cannot be changed
char str [size];
int age;
cout << " Enter your age: ";
cin>> age;
cin.ignore(10 , '\n');                  // eat the new line.
cout << " Enter yourname : ";
cin.get (str , size);    ;              // get text from user
 cout<<str;
```

مثال:

اكتب برنامجاً يمكن من إنشاء قاعدة بيانات لدليل هاتف ل100 مشترك، وتتضمن القاعدة الاسم ورقم الهاتف.

```
// telephon.cpp
// models data base of telephon
# include <iostrem.h>
# include <conio.h>
const int size = 30 ;
const int max = 100 ;
class telephon
{
  private :
char name [size];
```

```
int tel_number ;
public :
  void input ( )
     {
        cout<< " Enter the name : ";
        cin.get( ) (name , size);
        cout<< " Enter the telephon number : ";
        cin>> tel_number;
     }
  void output ( )
     {
        cout<< " Name : "<<name;
        cout<< " Telephon number : "<< tel_number;
     }

};
void main ( )
{
 telephon tel_database [max ];    // array of object
int n = 0;
int i ;
char choice = 'x';
while (choice != 'q')                    // exit on 'q'
   {
     cout << "\n 'a ' to add an telephon ";
     cout << "\n ' d ' to display all telephon ";
     cout << "\n ' q ' to quit program ";
     cin>> choice;
      cin.ignore(10 , '\n');          // eat extra '\n'
switch (choice)
{
     case ' a ' :
        {
```

```
            cout << " Enter data for telephon "<<(n+1)<<endl;
                    tel_database [i++] . input ( );
                     break;
                }
            case ' d' :
                {
                    for (i=0 ; i<n ; i++)
                          cout << "\n Data for telephon "<<(i+1)<<endl;
                          tel_database [i] . output ( );
                          break;
                          }
            case ' q ' :
                        {
                          break;     //terminale program
                          }
            default :
                    cout << " Unknown command";
                }
            }
        }
```

لقد استخدمنا سلاسل حقيقية لتسهيل الأعضاء الدالية input(0 و output() التابعة للصف telephon . لقد اختفت الحلقات do و for.

لقد استعملنا الدالة cin.get() لقراءة الاسم في العضو الدالي input(). يتطلب هذا أن نستعمل الدالة cin.ignore() في main() لالتهام المحرف \n الموجود بعد خيار المستخدم لكي لا ترتبك دالة الادخال cin.get().

إن العبارة switch الموضوعة في حلقة while تجعل البرنامج تفاعلياً. فهي تعرض لائحة من خيارات الأحرف المحتملة؛ وعندما يختار المستخدم خياراً، ينفذ البرنامج المهمة المناسبة قبل العودة إلى لائحة الخيارات مرة أخرى.

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال عدد صحيح محصور بين **0** و **255** وإيجاد القيمة المقابلة لهذا العدد في جدول **ASCII** كما يمكن هذا البرنامج من إدخال محرف وإيجاد القيمة المقابلة له في النظام الست عشري والبرنامج ينفذ بنفسه ما دام المستخدم يرغب بذلك.

الحل:

```
# include <iostream>
#include <conio.h>
void main()
{
        //clrscr();
        char ch,ch1,ch2;
        int n;
        do
        {
        cout << "want character or number(c/m)";
                cin >> ch2;
        while(ch2 != 'c' && ch2 != 'C' && ch2 != 'm' && ch2 != 'M')
                {
                        cout << "enter your choos agin";
                        cin >> ch2;
                }
                if(ch2 == 'm' && ch2 == 'm')
                {
                        cout << "enter n = ";
                        cin >>n;
                        while (n<0 || n > 255)
                        {
                                cout <<"enter agin n = ";
                                cin >> n;
                        }
                        ch = n;
                        cout << n <<"is in ascii " <<ch;
                }
                else
                {
```

```
                cout << "enter the charactor";
                cin >> ch;
                n = ch;
                cout << ch <<"is in ascii " << n;
            }
            cout << "want enter agin(Y/N)";
            cin >> ch1;
        }
        while (ch1 == 'y' || ch1 == 'Y');
        getch();
    }
```

تمكن لغة البرمجة C++ من استخدام مفهوم المتجهات لإدخال سلسلة محارف.

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال سلسلة من المحارف مكونة من 9 أحرف ويتوقف البرنامج عند إدخال أو الضغط على المفتاح enter محرف enter يقابل \n ويقابل 13 في جدول ASCII.

الحل:

```
# include <iostream>
#include <conio.h>
void main()
{
        //clrscr();
        char string1[9];
        int i=0;
        char ch;
        cout << "enter character";
        ch = getch();
        while (ch != '\n')
        {
                string1[i] = ch;
                i++;
                if (i==9)
                        break;
                ch = getch();
```

```
        }
        for (i=0;i<9; i++)
        {
                cout << string1[i];
        }
        getch();

  }
```

اكتب برنامجاً يمكن من إدخال أيام الأسبوع في مصفوفة بعدها 7×10.

```
# include <iostream>
#include <conio.h>

void main()
{
        //clrscr();
        char Day[7][10];
        int i,j;
        char ch;
        for(i=0; i<7;i++)
        {
                ch = getch();
                j=0;
                while (ch != '\n')
                {
                        Day[i][j]=ch;
                        j++;
                        if(j==9)
                                break;
                        ch = getch();
                }
        }
        for (i=0;i<7;i++)
        {
                for (j=0;j<9;j++)
                {
                        cout << Day[i][j];
                }
                cout << endl;
        }
```

```
    getch();
}
```

ملاحظة:

الدالة getch تدخل محرفاً محرفاً وهذه الطريقة ليست عملية لإدخال سلسلة المحارف ولذا نستخدم.

13- دالة إدخال سلسلة المحارف ( ) gets

تمكن هذه الدالة من إدخال سلسلة محارف حسب رغبة المستخدم لإدخال سلسلة محارف وعند الضغط على **enter** تخزن هذه السلسة في المتجه المحرفي على أن يوضع في نهاية السلسلة \n. تمكن هذه السلسلة قبل التخزين من تصحيح الخطأ المرتكب أثناء إدخال سلسلة المحارف.

تحتاج دالة ( ) **gets** إلى الترويسات

```
#include <iostream.h>
#include <string.h>
```

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال أيام الأسبوع باستخدام دالة ( )gets ثم قراءة المتجه الذي يتضمن هذه المحارف.

```
# include <iostream>
#include <conio.h>
#include <stdio.h>
void main()
{
        //clrscr();
        char Day[7][10];
        int i,j;
        char ch;
        for(i=0; i<7;i++)
        {
                gets(Day[i]);
```

```
            cout <<"\n" << Day[i]<<"\n";
        }
        //clrscr();
        for (i=0;i<7;i++)
        {
            for (j=0;j<9;j++)
            {
                cout << Day[i];
            }
            cout << endl;
        }
        getch();
}
```

14- تعريف متغيرات الثوابت const:

تمكن لغة البرمجة C++ من تعريف متغيرات الثوابت التي لاتتغير قيمته في سياق البرنامج، وذلك بوساطة استخدام الكلمة المحجوزة **const**. فإذا حاولنا تغيير قيمته نحصل على رسالة خطأ عند ترجمة البرنامج.

مثال:

```
const int size = 50;      //constant variable , cannot be changed
char str [size];
cin.get (str , size);  ;          // get text from user
cout<<str;                    //display text
```

بما أن المتغير الثابت بوساطة الكلمة الأساسية **const**، لا يمكن تغيير قيمته، نسد قيمة متغير الثابت عند تعريفه، حيث يمكن تعريفة في ترويسة البرنامج. ويمكن تعريف متغير الثابت في ترويسة البرنامج باستخدام الكلمة الاساسية **define** واستخدام المرشد ما قبل المعالج.

مثال:

```
#define size = 50
```

15- المتغيرات الخارجية:

يتمّ تعريف المتغيرات الخارجية خارج كل شيء، لا توجد أقواس حاصرة تحيط بها. إن المتغيرات المعرفة بهذه الطريقة تسمى متغيرات خارجية **(external variables)**. يمكن الوصول إلى المتغيرات الخارجية من كل أجزاء البرنامج، بينما المتغيرات المعرفة ضمن الصف يمكن الوصول إليها من ضمن تلك الصف فقط كما أن المتغيرات المعرفة ضمن دالة ما يمكن الوصول إليها من ضمن تلك الدالة فقط.

16- مكتبة دوال سلاسل المحارف:

إن مكتبة الدوال التي ترافق مترجمات لغة البرمجة C وفي لغة البرمجة C++ تتضمن عشرات من الدوال التي تعمل على السلاسل. فهناك دوال تنسخ السلاسل وتقارنها وتبحث فيها عن سلاسل أخرى.

- دوال مكتبية للسلاسل:

تتطلب كل دوال سلاسل المحارف شمل ملف الترويسة **string.h**.

- دالة حساب طول متسلسلة محارف ()strlen:

تمكن هذه الدالة من حساب طول سلسلة المحارف s أي عدد الأحرف الموجودة في هذه السلسلة إضافة إلى الفراغات حيث يعتبر الفراغ محرفاً. تعيد هذه الدالة قيمة صحيحة.

مثال:

المقطع البرمجي التالي يبين استخدام دالة حساب طول سلسلة محارف.

```
char s1[] = "ahmad";
cout<< " Length of s1 = ";
cout<< strlen(s1);
```

يكون الخرج

```
Length of s1 = 5
```

إن الحرف \0 المحرف الخامل غير محسوب في الطول الذي تعيده strlen()، علماً أنه موجود في s1 ويحتل مكانه في الذاكرة. في الحقيقة يبلغ طول المصفوفة s1 6 Bytes.

- دالة نسخ متسلسلة محارف strcpy( ) :

تمكن من نسخ متغير متسلسلة محارف أو ثابت متسلسلة محارف إلى متغير متسلسلة محارف آخر باستعمال الدالة strcpy().

مثال:

المقطع البرمجي التالي يبين اسخدام دالة نسخ سلسلة محارف.

```
char s1[] = "ahmad";
char s2 [20] ;
strcpy (s2 , s1);          //copies the contents of s1 into s2
```

بعد تنفيذ هذه العبارة ستحتوي السلسلة s2 علىسلسلة المحارف ahmad. لن تتغير السلسلة s1. لاحظ أن strcpy() تنسخ السلسلة الممررة كالوسيط الثانية إلى السلسلة الممررة كالوسيط الأولى.

- دالة إلحاق متسلسلة محارف strcat( ):

إلحاق (appending) سلاسل المحارف يمكن أن يسمى جمع سلاسل المحارف.

مثال:

المقطع البرمجي التالي يبين اسخدام دالة إلحاق سلسلة محارف.

```
char s1[ ] = "Ahmad";
char s2 [ ] = "abuo anas";
strcat (s1,s2);          // now s1 needs to be 16 bytes
cout << s1;
```

لقد أصبحت السلسلة **s1** تساوي الآن **Ahmad abuo anas**. إن الدالة المكتبية **strcat( )** تلحق السلسلة بواحدة أخرى. الجزء **cat** من اسم الدالة **strcat( )** هو اختصار **concatenation**. أي أنه يتمّ إنشاء سلسلة ثالثة تتألف من السلسلتين الأخريتين. يجب التأكد من أن هناك مساحة كافية.

- دالة مقارنة متسلسلات محارف strcmp( ):

لتدقيق كلمات المرور نستخدم دالة مقارنة متسلسلات المحارف **strcmp()** التي تمكن من مقارنة متسلسلتين وتعيد رقماً يشير ما إذا كانتا متطابقتين، وفي حال لم تكونا كذلك، أي منهما يأتي أولاً أبجدياً. هذه الدالة حساسة لحالة الأحرف، لذا فالسلسلة **Ahmad** ليست كالسلسلة **ahmad** . توجد دوال سلاسل محارف تقوم بمهام مشابهة لبعض دوال متسلسلات المحارف، لكن غير مطابقة لمهامها. الدالة **stricmp** مثلاً تقارن سلسلتين مثلما تفعل **strcmp()** لكنها غير حساسة لحالة الأحرف. لذا فإنها ستبلغك أن السلسلتين **Ahmad** و **ahmad** متطابقتان. أما الدالة **strncmp()** فمشابهة للدالة **strcmp( )** أيضاً لكنها تنظر إلى عدد محدد من الأحرف فقط في السلسلة التي تقارنها. هناك العديد من دوال السلاسل المتنوعة، تتميز عن غيرها بوجود حرف إضافي في اسمها.

مثال:

المقطع البرمجي التالي يبين استخدام دالة مقارنة سلاسل المحارف.

```
char s1[ ] = "Smith";
n1 = strcmp(s1 , "Renaldo");  <<-- return 1(1st arg follows 2nd)
n1 = strcmp(s1 , "Smith");  <<-- return 1(1st arg same as 2nd)
n1 = strcmp(s1 , "Townsend");  <<-- return 1(1st arg precedes 2nd)
```

- دالة تحويل حروف سلسلة المحارف لاحرف كبيرة toupper():

تقوم هذه الدالة بتحويل جميع محارف السلسلة إلى الحجم الكبير.

```
char ch = 'a ';
```

```
toupper(ch);
```

- **الدالة تحويل حروف سلسلة المحارف لاحرف صغيرة tolower()**

تقوم هذه الدالة بتحويل جميع محارف السلسلة إلى الحجم الصغير.

```
char ch = 'D ';
tolower (ch);
```

- **الدالة اظهار المحارف towasii ( )**

تمكن هذه الدالة من إظهار المحرف من جدول **ASCII**.

ملاحظات:

- عليك تعريف مصفوفة، وليس متغيراً واحداً، لتخزين سلسلة المحارف.
- لا يمكن نسخ سلسلة محارف ما إلى سلسلة محارف أخرى بوساطة معامل التعيين (=).
- لا تحذر دوال سلاسل المحارف إذا فاضت مصفوفة إحدى سلاسل المحارف.
- لا يمكن من إلحاق سلاسل المحارف بوساطة المعامل +.
- لا يمكن مقارنة سلاسل المحارف بوساطة المعاملات == و=! و> و<.

مثال:

اكتب برنامجاً بلغة البرمجة C++ ، يمكن من إنشاء صف سلاسل محارف **String1**. تتألف أعضاؤها البيانية من سلسلة محارف اعتيادية وتتضمن أعضاء دالية لتسند لنفسها قيماً عند سلسلة محارف اعتيادية، وللحصول على سلسلة من المستخدم، ولعرض محتوياتها، ولإلحاق سلسلة أخرى بها.

```
// Stringprog.cpp
//models strings
#include <iostream.h>
#include <conio.h>
```

```
#include <string.h>
const int max = 50 ;
class String1
 {
   private :
              char str [ max];
   public :
            void init ( char s [ ] )          // initialize with string
              {
               strcpy (str , s );
               }
          void input ( )           // get string
              {
               cin.get(str , max );
               }
          void display( )           // display string
              {
               cout<<str;
               }
           void append( String1 objStr )                  // append
argument string
              {
               if (strlen(str) + strlen(objStr)<max -1)
               strcat ( str , objStr);
                else
               cout << " \n Error String1 too long"<<endl;
              }
 };
void main ( )
       {
          String1  s1 , s2 , s3;                  // Objects
          s1.init ( "Greetings,  " );             //initialize s1
          cout << " Enter your name : ";
```

```
    s2.input ( );                    // Get s2 from user
    s1.append (s2);                  // append s2 to s1
    s3 = s1;
    s3.display ( );                  // display s3
}
```

ينشئ البرنامج في **main()** ثلاثة كائنات **String1**، ويسند لسلة المحارف **s1** عند سلسلة اعتيادية باستعمال الدالة **init()**، ويحصل على نص من المستخدم لسلسلة المحارف **s2** بوساطة الدالة **input()**، ويلحق سلسلة المحارف **s2** إلى سلسلة المحارف **s1** بوساطة الدالة **append()**، ويضبط سلسلة المحارف **s3** لتساوي سلسلة المحارف **s1** باستعمال معامل التعيين، وأخيراً يخرج محتويات سلسلة المحارف **s3** بوساطة الدالة **display()**.

17- مصفوفات سلاسل المحارف:

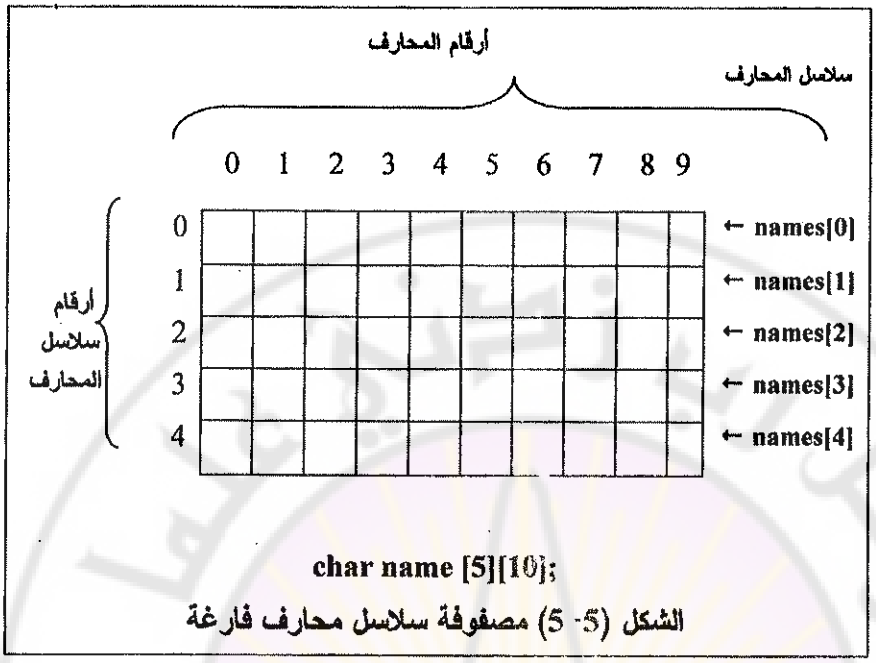
تمكن لغة البرمجة C++ من تعريف مصفوفة عناصرها سلاسل محارف. تمكن مصفوفة سلاسل المحارف من تخزين قوائم بأسماء العاملين وكلمات المرور وأسماء الملفات وغيرها.

- التركيب النحوي لتعريف مصفوفة سلاسل محارف:

نعرف متغير مصفوفة متسلسلة محارفة فارغة باستعمال مصفوفة ثنائية الأبعاد بسيطة. وبما أن سلسلة المحارف هي عبارة عن مصفوفة، فإن مصفوفة سلاسل المحارف هي مصفوفة المصفوفات.

```
char names [5][10];     //array of 5 strings
```

إن عدد سلاسل المحارف هو دائماً البعد الأول للمصفوفة، بينما طول كل سلسلة محارف (جميع سلاسل المحارف لها الطول نفسه، وذلك لأنها في المصفوفة نفسها) هو البعد الثاني.



char name [5][10];

الشكل (5- 5) مصفوفة سلاسل محارف فارغة

مثال:

اكتب مقطعاً برمجياً يمكن من إدخال بعض الأسماء في هذه المصفوفة:

```
for (i=0 ; i<5 ; i++)
  {
    cout<<" Enter name (or press enter to exit) : ";
    cin.get (strlen(names [i] == 0)   // if user press enter
     break;          }
```

لاتمكن الحلقة **for** للمستخدم من إدخال أكثر من خمسة أسماء. ويضغط **Enter** وبالتالي إدخال اسم طوله صفر في المصفوفة، يستطيع المستخدم إنهاء الحلقة بعد إدخال أقل من خمسة أسماء.

إن السلسلة الواحدة يشار إليها بـ **name[j]**، مع دليل واحد. إنها الطريقة للإشارة إلى مصفوفة فرعية في المصفوفات الثنائية الأبعاد.

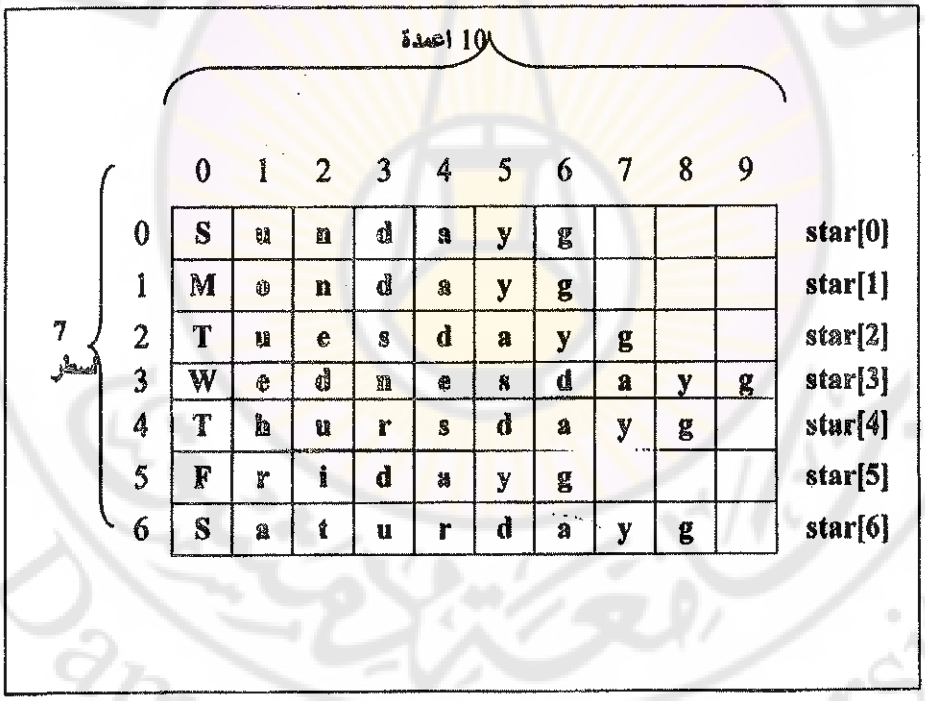
18- مصفوفات سلاسل المحارف الثابتة:

يمكن تعريف مصفوفة سلاسل محارف حيث تسند لسلاسل المحارف قيمة عند إنشائها.

مثال:

اكتب مقطعاً برمجياً يمكن من تخزين أيام الأسبوع:

```
const int max =10;
const int day =7;
conct char day_name [day][max] = { " Sunday" , "Monday" ,
"Tuesday",Wednesday","Thursday","Friday", "Saturday"};
```



الشكل (6-5) مصفوفة سلاسل المحارف

إن ثوابت سلسلة المحارف تسند لها القيم بحيث تكون مفصولة عن بعضها بعضاً بفواصل وتحيطها أقواس حاصرة، تماماً مثلما يحصل مع الثوابت في المصفوفات

الأحادية الأبعاد. يؤدي هذا إلى وضع كل ثابت متسلسلة محارف في المكان المناسب في المصفوفة day-name.

مثال:

اكتب برنامجاً يمكن تعرف مصفوفة سلسلة محارف لأيام الأسبوع وذلك باستخدام مفهوم الصفوف.

```
// weekdays
#include <iostream.h>
#include <conio.h>
#include <string.h>
const int max =10;          // maximum length of day name, +1
const int day = 7;          // days per week
                            // array of days name
conct char day_name [day][max] = { " Sunday" , "Monday" ,
                            "Tuesday" , "Wednesday" ,
                                "Thursday" , "Friday" ,
                                    "Saturday" };
class weekday
 {
   private :
          int day_number;          //sundy = 0, etc
   public :
         void inday ( )          // input day name
          {
            char tempday[max];
             int gotit = 0;
             int i ;
             while (!gotit)
                 {
            cout<< "Enter day of week (e.g.,Friday) : ";
                    cin>>tempday;
                    for (i=0 ; i<day ; i++)
```

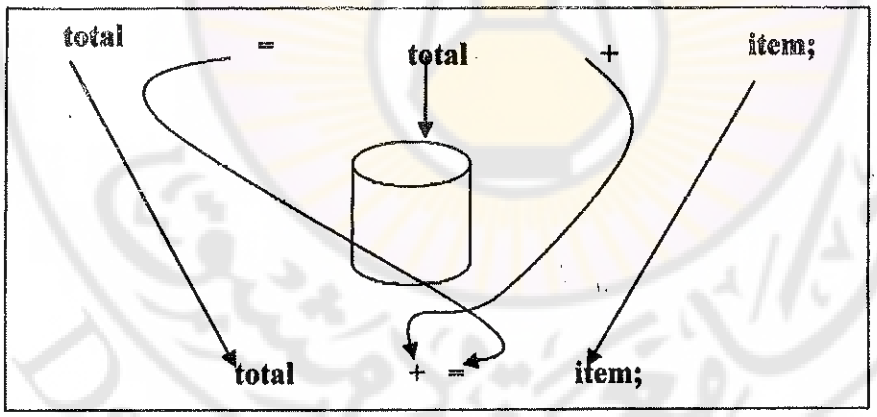
```
                {
            if (stricmp(tempday , day_name[i] ) ==0)
                    {
                      gotit = 1;
                       break;
                      }
                  }
                day_name = i ;
              }
          }
    void outday ( )          // display the day name
        {
          cout<<day_name[day_number] ;
          }
    void outnumber( )          // display the day number
        {
          cout<<(day_number + 1);
          }
     void add ( int days )          // add days to
        {
          day_number += days;
          day_number % = day
          }
};
void main ( )
    {
       weekday wd;                    // Objects
       cout << " What day is it ? "<<endl;
       wd.inday( );
       cout<<"You entered ";
       wd.outday ( );
       cout <<"\n That is day number ";
      wd.outnumber ( n);
```

```
wd.add(10);
cout<<"\n Yen days later is "
wd.outday( );}
```

- الصف weekday:

لقد عرفنا مصفوفة سلسلة المحارف الثابتة (أسماء أيام الأسبوع) day-name متغيرات خارجية. إن المتغيرات الخارجية يمكن الوصول إليها من أي مكان من البرنامج. ينشئ البرنامج main() كائناً تابعاً للصف weekday يدعى wd، ثم يحصل على اسم من المستخدم ويقارنه بأسماء كل أيام الأسبوع المخزنة في مصفوفة سلسلة المحارف الثابتة day-name. إذا كان هناك تطابق، يخزن البرنامج رقم اليوم في الكائن، ثم يبلغ ذلك الكائن أن يعرض اسم اليوم ورقمه. لقد أضفنا عضواً دالياً إلى الصف weekday للسماح للمستخدم أن يضيف عدداً من الأيام إلى كائن weekday ما. يبلغ البرنامج في main() الكائن wd أن يضيف 10 أيام إلى نفسه، ثم يعرض يوم الأسبوع الذي توقف عنده (10 أيام بعد الأربعاء هي السبت).

- معامل التعيين الحسابي:



الشكل (5-7) معامل التعيين الحسابي

يبين الشكل (5-7) كيف يبدو هذا عند إضافة item إلى total وتخزين النتيجة في total.

```
day_number = day_number + days;
```

معامل التعيين الحسابي = +. يأخذ هذا المعامل القيمة الموجودة على يمينه ويضيفها إلى المتغير الموجود على يساره، تاركاً النتيجة في ذلك المتغير الأيسر.

```
day_number + = days;
```

يوجد معاملات تعيين حسابية لكل العوامل الحسابية:

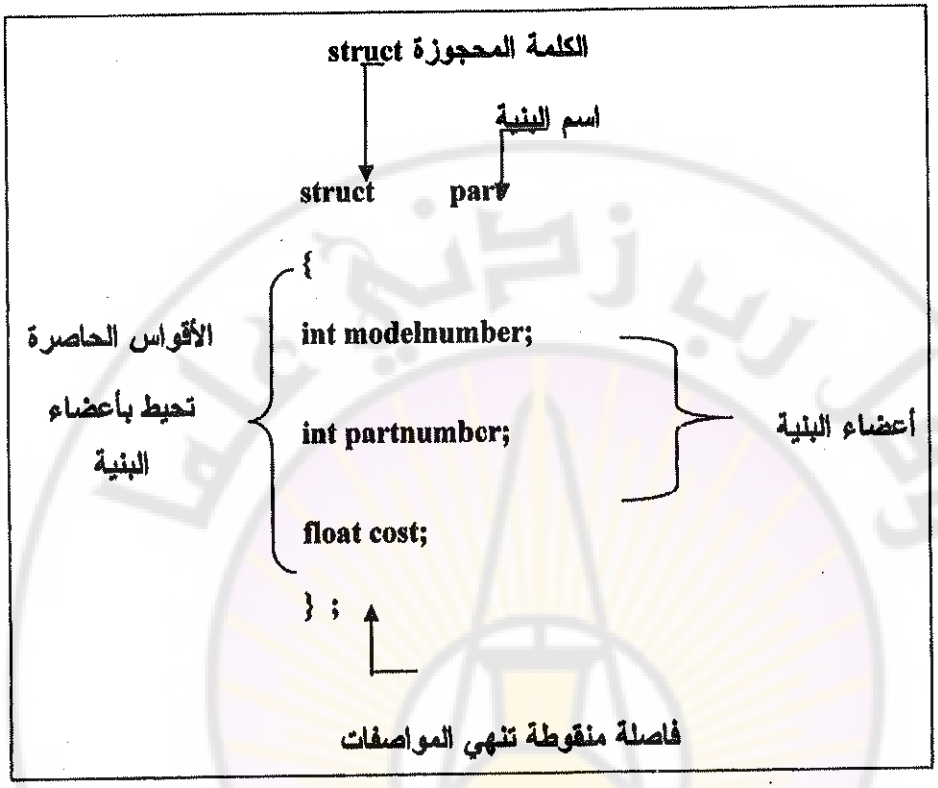
```
a + = b        //same as a = a + b
a - = b        //same as a = a - b
a * = b        //same as a = a * b
a / = b        //same as a = a / b
a%+ = b        //same as a = a % b
```

وهناك عوامل تعيين أخرى أيضاً، لكن تمكن من تجاهلها في الوقت الحاضر.

يمكن المعامل %، من التأكد من أن **day-number** يقع دائماً في المجال 0 إلى 6.

19- البنية:

إن البنية هي طريقة لتجميع عدة متغيرات بيانات يمكن أن تكون من أنواع مختلفة. مع العلم أن المصفوفة تمكن من تخزين عدد من متغيرات البيانات لكن من النوع نفسه.



الشكل (5-8) التركيب النحوي للبنية

تتألف البنية من الكلمة المحجوزة **struct** يليها اسم البنية وأقواس حاصرة تحيط بجسم البنية. تنتهي البنية بفاصلة منقوطة. يتألف جسم البنية عادة من عدة متغيرات بيانات، يمكن أن تكون من أنواع مختلفة. هذه المتغيرات تسمى أعضاء بيانات **members** . يبين الشكل (5-8) التركيب النحوي. إن البنية شبيهة جداً للصف من ناحية التركيب النحوي. لكن تحتوي الصف على متغيرات بيانات ودوال، بينما تحتوي البنية على متغيرات بيانات فقط.

مثال:

```
struct  part
{
int modelnumber;
```

```
int partnumber;
float cost;
};
```

- تعريف متغيرات البنية:

لتعريف متغيرات من النوع البنيوي **struct part**، نكتب كما يلي:

```
part cpl, cp2;
```

ممكن تعريف متغيرات البنية بطريقة مختصرة. يتمّ وضع أسماء المتغيرات في مواصفات البنية كالتالي:

```
struct part
{
int modelnumber;
int partnumber;
float cost;
} cpl, cp2;
```

- الوصول إلى أعضاء البنية:

يمكن الوصول إلى أعضاء البنية بوساطة استعمال الكائنات و معامل النقطة.

مثال:

اكتب برنامجاً يمكن من تعريف بنية تتضمن ثلاثة متغيرات صحيحة واستخدام هذه المتغيرات حيث تخزن القيمة الأولى والثانية وتخزن قيمة الثابت حاصل جمع الأول والثاني إخراج النتيجة

الحل:

```
#include <conio.h>
#include <iostream.h>
struct V1
{
        int x;
        int y;
```

```
        int z;
};

void main()
{
        clrscr();
        V1 ob;
        cout << "enter the first elemenet";
        cin >> ob.x;
        cout << "y=";
        cin >> ob.y;
        ob.z = ob.y + ob.x;
        cout << "result is"<< ob.z;
        getch(); }
```

مثال:

يتضمن البرنامج بنية كعضو بياني واحد في الصف Stack. ويبين استخدام المكدس

```
//strustak.cpp
#include <iostream.h>
#include <conio.h>
struct  stackette
 {
  int arr[20];
  int top;
 };
class Stack
 {
   private :
          stackette st;
   public :
          void init ( )          // initialize index
           {
             st.top = -1;
           }
```

```
void push (int var )          // place an item on the stak
     {
       st.arr[++st.top] = var ;
       }
int pop( )          // remove an item from the stak
     {
       return st.arr[st.top--];
       }
};
void main ( )
     {
       Stack s1;                    // Objects
        s1.init();       //initialize
        s1.push(11);
        s1.push(12);
        s1.push(13);
        cout<<s1.pop( ) <<endl;
        cout<<s1.pop( ) <<endl;
        cout<<s1.pop( ) <<endl;
     }
```

تخزن البنية stackette هنا مصفوفة أعداد صحيحة والدليل يشير إلى أعلى المدكس. العضو البياني في الصف Stack الآن هو متغير للصف stackette.

```
Stackette st;
```

وتشير الأعضاء الدالية للصف Stack الآن إلى الأعضاء البيانية الفردية في st باستعمال عامل النقطة

20- المتغيرات البنيوية الثابتة:

يمكن اسناد قيم أولية للمتغيرات البنيوية الثابتة تماماً كما تفعل مع المصفوفات.

مثال:

```
part cp1 = {6244, 373, 217.55};
```

21- متغيرات بيانات من نوع التعدادي enum :

يتيح نوع البيانات التعدادي ــ المحدد بواسطة الكلمة الأساسية **enum** ــ للمبرمج اختراع نوع بيانات جديد ثم تحديد القيم المسموحة فيها.

- تعريف متغيرات من نوع تعدادي:

نعرف متغيرات من نوع تعدادي وفق مايلي : الكلمة المحجوزة **enum** ثم اسم النوع، ثم أقواس حاصرة تحيط بلائحة من أسماء قيم تفصلها فواصل. يخزن المترجم عادة القيم التعدادية كأعداد صحيحة بدءاً من 0 للقيمة الأولى.

$$\textbf{enum days} = \{day_1, day_2, ...\}$$

يمكن تحديد بدء الترقيم بقيمة أخرى غير 0 :

$$\textbf{enum days} = \{day_1 = 11; day_2, ...\}$$

ويمكن تحديد قيم مختلفة لكل الأسماء.

$$\textbf{enum days} = \{day_1 = 11; day_2 = 15, ...\}$$

22- متغيرات من نوع بوليالي (منطقي)bool:

تمكن لغة البرمجة C++ من تعريف نوع من المتغيرات المنطقية **(Boolean)** . هذا النوع (المسمى على اسم عالم الرياضيات البريطاني جورج بوول **George Boole**) له قيمتين فقط: **true** (صح) و**false** (خطأ). يتمّ استعمال المتغيرات من هذا النوع لتخزين نتائج الاختبارات المنطقية.

مثال:

نعرف المتغير المنطقي وفق مايلي:

```
bool flag;
```

مثال:

تمكن المتغيرات من نوع **bool** الجديد استعمال نوع منطقي صح / خطأ. فيما

يلي البرنامج weekdays وقد أعيدت كتابته لاستعمال نوع منطقي للمتغير gotit.

```
// weekbool.cpp
#include <iostream.h>
#include <conio.h>
#include <string.h>
const int max =10;          // maximum length of day name, +1
const int day = 7;          // days per week
                            // array of days name
conct char day_name [day][max] = { " Sunday" , "Monday" ,
"Tuesday" , "Wednesday" ,"Thursday" ,  "Friday",
"Saturday" };
class weekday
 {
   private :
            int day_number;          //sundy = 0, etc
   public :
          void inday ( )          // input day name
            {
              char tempday[max];
              bool gotit = false;
              int i ;
              while (gotit == false)
                  {
                cout<< "Enter day of week (e.g.,Friday) : ";
                    cin>>tempday;
                    for (i=0 ; i<day ; i++)
                      {
                    if (stricmp(tempday , day_name[i] ) ==0)
                          {
                            gotit = true;
                             break;
                          }
```

```
                    }
                day_name = i ;
                }
            }
    void outday ( )          // display the day name
        {
          cout<<day_name[day_number] ;
          }
    void outnumber( )        // display the day number
        {
          cout<<(day_number + 1);
          }
     void add ( int days )        // add days to
        {
          day_number += days;
          day_number % = day
          }
};
void main ( )
    {
       weekday wd;                  // Objects
       cout << " What day is it ? "<<endl;
       wd.inday( );
        cout<<"You entered ";
       wd.outday ( );
       cout <<"\n That is day-numbcr ";
      wd.outnumber ( n);
      wd.add(10);
      cout<<"\n Yen days later is "
      wd.outday( );
    }
```

تمرين

1. اكتب برنامجاً بلغة C++ يمكن من إدخال أسماء وأرقام هواتف هؤلاء الأشخاص. وذلك باستخدام مفهوم إدخال السلاسل المحرفية بلغة C++.


وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل السادس

الدوال

1- مقدمة:

إنّ الدوال هي إحدى المكونات الرئيسة في الصفوف وفي الكائنات، وتمثل الدوال أهم الطرق الأساسية لتنظيم البرامج في لغة البرمجة C++ .

2- الوسطاء:

الوسطاء هي الألية المستخدمة لتمرير المعلومات إلى الدالة التي نستدعيها.

مثال:

المقطع البرمجي التالي يبين تمرير وسطاء الى دالة، حيث تقوم الدالة بعرض المحرف عدة مرات. يتمّ تمرير هذا المحرف المطلوب عرضه كوسيط إلى الدالة.

```
:
display ('a' , 5);          // display a 5 tims
:
void display (char ch , int n )
{
   for (int i = 0 ; i <n ; i++)
      cout<< ch;
}
```

تخرج هذه الدالة المحرف **ch** تماماً **n** مرة على سطر واحد. تحصل هذه المتغيرات (**ch** و**n**) على القيم التي تظهر في استدعاء الدالة.

```
display ('?',20);
```

فإنها ستعرض 20 علامة الاستفهام.

عند تمرير البيانات كوسطاء، يتمّ نسخ البيانات وتخزّن القيم في متغيرات منفصلة

في الدالة. تتمّ تسمية هذه المتغيرات في ترويسة الدالة. يسمى هذا الأمر التمرير بالقيمة.

ملاحظة:

وسطاء **(arguments)** يعني القيم المحددة للمتغيرات عند استدعاء الدالة، أما البارامترات **(parameters)** هي المتغيرات في تعريف الدالة التي تمّ نسخ القيم إليها.

3- قيم الإعادة:

يمكن الدالة أن تعيد قيمة إلى العبارة التي استدعتها.

مثال:

اكتب برنامجاً يمكن من تحول الباوندات (lbs) إلى كيلوغرامات (kg). تتطلب هذه الدالة وسيطاً واحداً، وهي قيمة بالباوندات، وتعيد القيمة لها بالكيلوغرامات.

```
float Kilogram (float pounds)
 {
   float kilograms = 0.453592 * pounds;
   return kilograms;
 }
```

إن نوع القيمة المعادة، من نوع **float**، يجب أن يسبق اسم الدالة نوعها عند تعريفها. إذا كانت الدالة تعيد قيمة، يجب استعمال العبارة **return**. والقيمة المطلوب إعادتها يجب تحديدها بعد الكلمة المحجوزة **return**.

```
return kilograms;
```

إن العبارة **return** تمكن من إعادة التحكم (التنفيذ) إلى العبارة التي استدعت الدالة، حتى ولو لم تكن آخر عبارة في الدالة. كما أن عبارة **return** تسند القيمة المعادة للتعبير الذي استدعى الدالة أى إسناد القيمة المعادة لمتغير العبارة.

العبارة التي استدعت الدالة كما يلي:

```
kgs = Kilogram (lbs);
```

تسند القيمة المعادة للمتغير **kgs** في هذه الحالة.

4- تصريح الدالة:

عندما يترجم المترجم تعليمات استدعاء دالة ما، فإنه يحتاج إلى معرفة اسم الدالة، وعدد وسطائها وأنواعها ونوع قيمة الإعادة. بحيث تكون الدالة قد تمَّ تعرفها قبل استدعاء الدالة. يخزن المترجم هذه الخصائص للرجوع إليها عند الحاجة.

تصريح الدالة هو عبارة عن تعليمات تبلغ المترجم عن اسم الدالة وعدد وسيطاتها وأنواعها ونوع قيمة الإعادة. إليك كيف يتمُّ تصريح **Kilogram ( ):**

```
float Kilogram (float);
```

يشبه التصريح عن الدالة تعريف الدالة (السطر الأول في تعريف الدالة). لكن تليه فاصلة منقوطة كونها عبارة مستقلة. ولا تحتاج أيضاً إلى تحديد أسماء الوسطاء، ولكن تحتاج فقط لأنواع الوسطاء في هذه الدالة.

يسمى التصريح عن الدلة أنموذجاً أولياً **prototype** كونه يبين نوع الدالة التي ستعرف فيما بعد.

- الدوال مستدعاة من الدالة الرئيسة main( ):

الدالة تستدعي func1() و func2().

```
void func1 ( );          // declaration of func1( )
 void func2 ( );         // declaration of func2( )
void main ( )
  {
    func1( );  // call to func1 ( )
    func2( );  // call to func2 ( )
   }
void func1 ( )   // definition of func1 ( )
     {
      // statements
      }
 void func2 ( )      // definition of func2 ( )
     {
```

```
        // statements
       }
```

إذا كانت الدوال المراد استدعاؤها تلي الدالة الرئيسة ( )main ، يجب التصريح عن هذه الدوال قبل استدعائها. ويمكن أن نصرح عن الدوال ( )func2 , ( )func1 داخل الدالة الرئيسة ()main كما يلي:

```
void main ( )
  {
    void func1 ( );          // declaration of func1( )
    void func2 ( );          // declaration of func2( )
    func1( );  // call to func1 ( )
    func2( );  // call to func2 ( )
   }
void func1 ( )   // definition of func1 ( )
     {
      // statements
     }
void func2 ( )       // definition of func2 ( )
     {
      // statements
     }
```

5- الدوال المستدعاة من الأعضاء الدالية:

يمكن للدالة استدعاء دالة ليست جزءاً من الصف. يتطلب هذا الأمر التصريح عن تلك الدالة.

مثال:

```
// programfunc.cpp
# include <iostream.h>
class afunc
  {
     .....
```

```
    void func1 ( )
      {
       ……..
       void func2 ( );
       func2 ( );
       ……….
       }
  };
void func2 ( )
    {
     // statemant
     }
void main ( )
    {
     // statement
     }
```

يمكن وضع التصريح عن الدالة خارج الصف، بحيث يظهر قبل أن يتمّ استدعاء الدالة.

6- الدوال السياقية (inline):

الدالة السياقية هي دالة يضعها المترجم في متن البرنامج، حيث يتمّ استدعاء هذه الدالة. تعريف الدالة السياقية بالتركيب النحوي نفسه المستخدم لتعريف الدالة الاعتيادية. يوضع جسم الدلة السياقية في متن البرنامج، حيث يتمّ استدعاء هذه الدالة. لا يوجد وقت ضائع عند استدعاء الدوال السياقية.

- تعريف الدالة السياقية:

تعرف الدالة سياقية باستخدام الكلمة المحجوزة **inline** ، ونوع الدالة، واسم الدالة، ووسطاء الدالة، وتوضع التعليمات (جسم الدالة) ضمن قوسين حاصرين:

```
inline void func ( )
```

```
{
  // statements
}
```

وإذا لم نستخدم الكلمة المحجوزة **inlin** تُعرَّفُ الدالة كدالة عادية.

يمكن التصريح عن الدالة السياقية في ترويسة البرنامج، وبوضع جسم الدالة في الموقع الذي يرغبه المبرمج وفق ما يلي:

```
inline void func ( );
.......
inline void func ( )
    {
     // statements
     }
```

ويتمُّ استدعاء الدالة بالطريقة نفسها:

```
{....
func()
........
 }
```

ملاحظة:

الأعضاء الدالية المعرفة ضمن مواصفات الصف تكون سياقية بشكل افتراضي. أي ستكون سياقية سواء استعملت الكلمة المحوزة **inline** أم لا.

ملاحظة:

لايمكن للدالة **()main** أن تكون دالة سياقية.

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال عددين صحيحين وإيجاد العدد الأكبر بينهما وذلك باستخدام الدالة السياقية:

```
#include <iostream.h>
#include <conio.h>
int inline max (int m ,int n)
```

```
{
        return (m>n)?m:n;
}
void main ()
{
        //clrser();
        int x,y,z;
        cout << "Enter x=";
        cin >> x;
        cout << "Enter y=";
        cin >> y;
        z=max(x,y);
        cout << "the max is: " << z;

        cin >> x;
}
```

7- معامل دقة المدى:

عند تعريف العضو الدالي خارج الصف، يجب أن يتضمن تعريف العضو الدالي على اسم الصف التي يتبع لها؛ واسم الدالة . يفصل بين الاسمين المعامل :: المستخدم لربط اسم الصف واسم الدالة ويسمى معامل دقة المدى scope resolution operator.

التركيب النحوي لمعامل دقة المدى وفق التالي:

```
class mm
{
private:
   variables;
public:
   int max (int,int);
   int xstring(int, char);
   {
```

```
};
}
int mm  ::max(int m,int n)
{
   statemenet;
}
```

- تعريف أعضاء الدالية خارج الصف:

يمكن أن نعرف عضواً دالياً خارج الصف، على أن يتمّ التصريح عن العضو الدالي ضمن الصف.

```
// programfun.cpp
# include <iostream.h>
 class ss
    {
      private:
        // variables
      public:
         void func1 ( );
    };
      void ss :: func1 ( )
           {
              // statements
           }
 void main ( )
   {
     // statements
    }
```

تكون الأعضاء الدالية المعرفة خارج الصف غير سياقية بشكل افتراضي، ولكن يمكن جعلها سياقية باستخدام الكلمة المحجوزة **inline**.

مثال:

اكتب برنامجاً يمكن من إدخال عددين صحيحين واختبار إحدى العمليات الحسابية (+,-,*) وذلك باستخدام مفهوم الصف ومعامل المدى لهذه الدوال.

الحل:

```
#include <iostream>
# include <conio.h>
class oper
{
public:
        int sum(int,int);
        int mul(int, int);
        int sub(int,int);
}
int oper::sum(int n,int m)
{
        return (n+m);
}

int oper::sub(int n,int m)
{
        return (n-m);
}

int oper::mul(int n,int m)
{
        return (n*m);
}

void main ()
{
        clrsrc();
        oper ob;
        int x,y,z;
        char ch;
        cin >>x;
        cin>>y;
```

```
        cout << "enter the operator + * or -";
        cin >> ch;
        while(ch != '*' && ch !='-' && ch != '+')
        {
                cout << "enter the operator + * or -";
                cin >> ch;
        }
        switch(ch)
        {
        case '*':
                {
                        z= ob.mul(x,y);
                        cout z;
                }
                break;
        case '+':
                {
                        z= ob.sum(x,y);
                        cout z;
                }
                break;
        case '-':
                {
                        z= ob.sub(x,y);
                        cout z;
                }
                break;
        }
        getch();
}
```

مثال:

أعد كتابة برنامج weekdays واستخدم معامل دقة المدى.

```
// weekbool.cpp
#include <iostream.h>
#include <conio.h>
#include <string.h>
```

```
const int max =10;          // maximum length of day name, +1
const int day = 7;          // days per week
                            // array of days name
conct char day_name [day][max] = { " Sunday" , "Monday" ,
        "Tuesday" , "Wednesday" ,
            "Thursday" , "Friday" ,
                    "Saturday" };
class weekday
 {
   private :
            int day_number;          //sundy = 0, etc
   public :
           void inday ( )           // input day name
            {
              char tempday[max];
              bool gotit = false;
              int i ;
              while (gotit == false)
                  {
          cout<< "Enter day of week (e.g.,Friday) : ";
                     cin>>tempday;
                     for (i=0 ; i<day ; i++)
                       {
                 if (stricmp(tempday , day_name[i] ) ==0)
                              {
                                gotit = true;
                                 break;
                               }
                       }
                     day_name = i ;
                  }
            }
        void outday ( )          // display the day name
```

```
        {
          cout<<day_name[day_number] ;
          }
      void outnumber( )          // display the day number
        {
          cout<<(day_number + 1);
          }
      void add ( int days )          // add days to
        {
          day_number += days;
          day_number % = day
          }
};
                                  // external member function
void weekday :: inday ( )          // input day name
        {
          char tempday[max];
          bool gotit = false;
          int i ;
          while (gotit == false)
               {
            cout<< "Enter day of week (e.g.,Friday) : ";
                 cin>>tempday;
                 for (i=0 ; i<day ; i++)
                   {
            if (stricmp(tempday , day_name[i] ) ==0)
                        {
                          gotit = true;
                          break;
                        }
                   }
                day_name = i ;
              }
```

```
}
void main ( )
    {
      weekday wd;                    // Objects
       cout << " What day is it ? "<<endl;
       wd.inday( );
        cout<<"You entered ";
       wd.outday ( );
       cout <<"\n That is day number ";
      wd.outnumber ( n);
      wd.add(10);
      cout<<"\n Yen days later is "
      wd.outday( );
     }
```

8- المايكرو Micro :

الماكرو هو وسيلة برمجية لتنفيذ مهمة صغيرة يتكرر استخدامها ضمن البرامج وتحجز لها مساحة في الذاكرة (لتحقيق ما تفعله الدوال السياقية في لغة البرمجة C++). إذا كانت المهمة البرمجية صغيرة أي لا تأخذ مساحة في الذاكرة فإنه من الأفضل استخدام الـ Micro عوضاً عن الدالة.

التركيب النحوي للماكرو Micro نميز حالتين:

أ – يعرف المايكرو في السطر نفسه

```
#define // name of micro (statement);
```

ب ـ يتضمن المايكرو Micro عبارات برمجية

```
#define //name of Micro
{
  //statement;
}
```

مثال:

اكتب برنامجاً بلغة C++ يمكن من إدخال عددين صحيحين وإيجاد العدد الأكبر بينهما وذلك باستخدام – الدوال – ثم باستخدام المايكرو Micro

الحل باستخدام الدوال:

```
#include <iostream.h>
#include <conio.h>
int max (int, int);
void main ()
{
        //clrser();
        int x,y,z;
        cout << "Enter x=";
        cin >> x;
        cout << "Enter y=";
        cin >> y;
        z=max(x,y);
        cout <<"the max is: " << z;
}
int max(int m,int n)
{
        return (m>n)?m:n;
}
```

حل البرنامج باستخدام المايكرو Micro

```
#include <iostream.h>
#include <conio.h>
#define max (m,n)((m>n)?m:n);
void main ()
{
        //clrser();
        int x,y,z;
        cout << "Enter x=";
        cin >> x;
        cout << "Enter y=";
        cin >> y;
        z=max(x,y);
```

```
    cout << "the max is: " << z;
}
```

ملاحظة:

لا يستطيع المايكرو **Micro** تدقيق نوع البيانات المدخلة إليه لذلك لا حاجة لتعريف نوع البيانات المستخدمة في المايكرو. يستخدم المايكرو المرشد ما قبل المعالج **#define**، مع وسطاء، لتقليد الدالة. الدالة **()main** المنفذة كمايكرو ستبدو كالتالي:

```
#define max (m, n) ((m>n) ? m : n)
```

يقوم المرشد **#define** باستبدال النص الموجود على اليمين، **((m>n) ? m : n)** مكان الاسم الموجود على اليسار **(max)** كلما ظهر هذا الاسم في الملف المصدر. الأمر كأنه معالج نصوص يجري عملية بحث واستبدال عامة عن هذا الاسم، ما عدا أنه يستعمل وسطاء. إذا "استدعينا" المايكرو من برنامج كما يلي:

```
cout << max(3, 5);
```

نحصل على الخرج: 5

لقد قام المرشد ما قبل المعالج بتوسيع هذه العبارة:

```
cout << ((3>5) ? 3 : 5);
```

قبل أن يتمّ نقلها إلى المترجم. عند استدعاء المايكرو **Micro** تمّ توليد قسم صغير من الشيفرة ووضعه سياقياً.

9- الدالة المحمّلة بشكل زائد:

تمكن لغة البرمجة **C++** من تحميل الدوال بشكل زائد **(overloading functions)**. أي أنه يمكن من استخدام الاسم نفسه لعدة دوال، بحيث أن كل دالة منها يجب أن يكون لها تعريف مستقل. يتعرف المترجم على هذه الدوال وفق مايلي، يبحث المترجم عن نوع وسيطات الدالة وعددها لمعرفة الدالة المقصودة. فيقوم بتشويه الأسماء، وذلك من خلال إنشاء اسم جديد عن طريق دمج اسم الدالة مع أنواع وسطائها. أن نوع إعادة الدالة ليس جزءاً من الاسم المشوه. فبذلك، لا يستطيع المترجم التمييز بين الدوال على أساس نوع الإعادة.

مثال:

اكتب برنامجاً يمكن من تعريف ثلاث دوال لها الاسم نفسه (تحميل الدوال بشكل زائد).

```
// overload_program.cpp
 #include <iostream.h>
 #include <conio.h>
 #include <string.h>
 const int Max_Length = 30 ;        // length of text
 class Display_Text
      {
       private:
          char text [Max_Length] ;         // array of text
       public:
            void set_text (char tx [ ] )
                  {
                    strcpy (text , tx)
                  }
            void box_display ( );      // line of dashes
            void box_display (char );     // line of characters
            void box_display (char , int );    // line of  n characters
      };
        void Display_Text :: box_display ( )      // line of dashes
             {
               cout<< "----------------------------------";
                cout<<endl<<text<<endl;
               cout<< "----------------------------------";
             }
   void Display_Text :: box_display (char ch ) // line of characters
              {
               int i;
               for(i=0; i< Max_Length; i++)
```

```
            cout<<ch;
        cout<<endl<<text<<endl;
        for(i=0; i< Max_Length; i++)
            cout<<ch;
    }
void Display_Text::box_display(char ch,int n)//line of  n characters
    {
     int i;
     for(i=0; i<n; i++)
         cout<<ch;
      cout<<endl<<text<<endl;
     for(i=0; i< n; i++)
         cout<<ch;
    }
void main ( )
    {
     Display_Text  obj1;
      obj1. set_text (" My first program");
     obj1. box_display ( );       // display text
        cout<<endl<<endl;
     obj1. box_display ( 'z' );       // display text
        cout<<endl<<endl;
     obj1. box_display ('v', 20 );       // display text
        cout<<endl<<endl;
    }
```

تمّ تعريف عدة دوال box-display() ستنشئ مربعات بدرجات متفاوتة من التعقيد. ستعرض الدالة الأولى خطوطاً من 30 قاطعة. وستعرض الثانية خطوطاً من 30 حرفاً، لكنها سترسمها باستعمال أي حرف محدد. وستتيح الثالثة للمستخدم تحديد الحرف وطول الخطوط معاً. لقد تمّ تعريف كل الدوال box-display( ) خارج الصف. وتمّ التصريح عن هذه الدوال ضمن الصف. إنّ الدوال الثلاث تحمل الاسم نفسه إلا أنها مختلفة كلياً بالنسبة للمترجم. ولقد قام المترجم بإنشاء أسماء جديدة منفصلة لكل دالة عن طريق تشويه أسماء أنواع الوسيطات في اسم الدالة.

10- الوسطاء الافتراضية:

إن الوسطاء الافتراضية تعني تجاهل وسيط واحدة أو أكثر عند استدعاء الدالة. عند التصريح عن الدالة نحدد الوسيط الافتراضي، وذلك من خلال وضع معامل المساواة وإسناد القيمة الافتراضية لهذا الوسيط. وعند فقدان وسطاء في حال استدعاء الدالة، يزود تصريح الدالة قيماً ثابتة لتلك الوسيطات المفقودة. تمكن الوسطاء الافتراضية المبرمج من تخفيض عدد أسماء الدوال المستخدمة. أي تمكن الوسطاء الافتراضية دالة ما أن تعمل كما لو أنها كانت عدة دوال.

مثال:

اكتب مقطعاً برمجياً يمكن من إيجاد ناتج عدد مرفوع لقوة. لدينا دالة تدعى power() هدفها رفع رقم من النوع **float** إلى أس **(power)** معين، يكون شكل الدالة كما يلي:

```
answer = power (2.0 , 3);
```

تعريف دالة كما يلي:

```
// programvur.cpp
....
float power ( float , int = 2 )
.............
void main ( )
{
 .............
  y = power( 6.5 );
 .........
  z = power (5.3 , 4 );
 ......
 }
float power ( float x , int n )
    {
      float produt = 1.0;
```

```
    for (int i = 0; i<n; i++)
      product * = x;
    return product;
  }
```

تستدعي الدالة الرئيسة **main( )** الدالة **power( )** في الطريقتين، أولاً مع وسيط واحدة ثم مع وسيطتين.

ملاحظة:

يمكن أن نستخدم الوسطاء الافتراضية بدلاً من تحميله الزائد العضو الدالي، وفقاً لعدد الوسطاء المرسلة إلى هذه الدالة.

مثال:

اكتب برنامجاً يمكن من استخدم الوسطاء الافتراضية بدلاً من تحميله الزائد العضو الدالي.

```
// overload_program.cpp
  #include <iostream.h>
  #include <conio.h>
  #include <string.h>
  const int Max_Length = 30 ;        // length of text
  class Display_Text
      {
        private:
            char text [Max_Length] ;          // array of text
        public:
            void set_text (char tx [ ] )
                {
                  strcpy (text , tx)
                }
void box_display(char='–' , int =Max_Length );// line of  n characters
      };
void Display_Text::box_display(char ch, int n) // line of  n characters
```

```
{
 int i;
 for(i=0; i<n; i++)
      cout<<ch;
 cout<<endl<<text<<endl;
 for(i=0; i< n; i++)
      cout<<ch;
}
void main ( )
{
 Display_Text  obj1;
  obj1. set_text (" My first program");
 obj1. box_display ( );       // display text
     cout<<endl<<endl;
 obj1. box_display ( 'z' );       // display text
     cout<<endl<<endl;
 obj1. box_display ('v', 20 );       // display text
     cout<<endl<<endl;
}
```

هناك ثلاث طرق لاستدعاء الدالة box-display(): تمكن من تجاهل الوسيطتين، أو تجاهل الثانية فقط.

11- الدالة cin.getline():

تستخدم الدالة cin.getline( ) وسطاء افتراضية. إن الدالة cin.getline( ) مشابهة للدالة cin.get( )، لكنها تستخدم وسيطاً افتراضياً لمحرف الإنهاء ــ المحرف الذي يدخلة المستخدم لإنهاء الدخل. إن تصريح الدالة cin.getline() في ملف الترويسة iostream.h.

مثال:

يمكن أن نكتب.

```
cin.getline (str, 40);    //get 40 characters
```

في هذه الحالة، تعود الدالة عندما يضغط المستخدم **Enter**.

أما في حالة إضافة الوسيط الثالث:

```
cin.getline (str, 40, '$');    //get 40 characters
```

في هذه الحالة، لن تعود الدالة عندما يضغط المستخدم **Enter** بل ستدالة قبول الدخل من لوحة المفاتيح إلى أن يضغط علامة الدولار ($).

ملاحظة:

تكون الوسطاء الافتراضية، الوسطاء الموجودة في نهاية قائمة الوسطاء. فإذا كان هناك لها وسيط افتراضي واحدة فقط، يجب أن يكون الأخير. وإذا كانت الدالة تتضمن وسيطين، يجب أن تكونا الأخيرين، إلخ. لا يمكن وضع وسيط افتراضية بين عدة وسطاء عادية.

مثال:

أي لا يمكن أن نكتب:

```
void func (int , int = 3 , int)
```

لأن الوسيط الافتراضي ليس الوسيط الأخير.

مثال:

يمكن أن نكتب:

```
void func (int , int = 3 , int = 5)
```

الوسيطان الافتراضيان موجودان في نهاية مجموعة الوسطاء.

12- المتغيرات التلقائية:

إن المتغيرات التي تمّ التصريح عنها ضمن الدالة هي متغيرات تلقائية.

مثال:

```
void func1 ( )
```

```
{
   int ivar1;          // automatic variables
    float fvar1;
}
```

يتمّ إنشاء المتغيرات التلقائياً عند استدعاء الدالة. وعندما تعود الدالة يتمّ تدمير متغيراتها التلقائية. حيث يتمّ دفع المتغيرات التلقائية في المكدس عند تنفيذ الدالة وعند الانتهاء يتمّ سحب هذه المتغيرات من المكدس. يمكن تعريف المتغيرات التلقائية خارج الدوال من خلال استخدام الكلمة المحجوزة **auto**

13- المتغيرات المسجلة:

يمكن تعريف المتغير المسجل **(register)** بواسطة استخدام الكلمة المحجوزة **register** و يخزن هذا النوع من المتغيرات التلقائية في مسجلات وحدة المعالجة المركزية **(CPU)** وليس في الذاكرة. إن هذا النوع من المتغيرات يجعل البرنامج أكثر فعالية.

14- المتغيرات الخارجية:

تعرف المتغيرات الخارج خارج أي صف أو دالة باستخدام الكلمة المحجوزة **external**. يمكن الوصول إلى المتغيرات الخارجية من جميع أجزاء البرنامج، وتحجز هذه المتغيرات الخارجية في الذاكرة مادام البرنامج قيد التنفذ.

مثال:

```
// start of file
int x;          //external variable
void func1 ( )
{
.....
}
```

ملاحظة:

إن المتغير الخارجي المعرف في ملف ما لا يمكن الوصول إليه من الملفات الأخرى. لكن يمكن جعله قابلاً للاستخدام من قبل الملفات الأخرى وذلك من خلال التصريح عنها باستخدام الكلمة المحجوزة **extern**، في أي ملف آخر تريد الوصول إليه منه.

مثال:

يتألف المقطع البرمجي من ثلاثة ملفات:

```
// fil1 #1
int n;
// n is known in this file
// end of file #1
// fil1 #2
extrn int n;
// n is also known in this file
// end of file #2
// fil1 #3
// n is not known in this file
// end of file #3
```

إن المتغير **n** معرف في الملف الأول، ومصرح عنه في الملف الثاني، وغير مصرح عنه في الملف الثالث. لذا فإنه يمكن الوصول إليه في الملفين الأول والثاني ولايمكن الوصول إليه الملف الثالث. أما إذا استخدمنا الكلمة المحجوزة **static** التي لا تعني ساكناً بل تعني قيداً فإنه لايمكن الوصول الى هذا المتغير من قبل الملفات الأخرى حتى ولو استخدمنا التصريح **extern**.

```
// fil1 #1
static int n;
// n is known in this file
// end of file #1
// fil1 #2
```

```
extrn int n;
// n is also known in this file
// end of file #2
```

يعتبر المترجم هنا أن المتغير n في الملف الثاني لم يتمّ تعريفه.

15- المتغيرات ساكنة محلية:

يمكن تعريف المتغير الساكن باستخدام الكلمة المحجوزة static ،حيث تمّ التصريح عن هذه المتغيرات ضمن الدالة. إن المتغيرات الساكنة تحجزاً حيز في الذاكرة مادام البرنامج قيد التنفيذ أى لايتمّ تدمير هذه المتغيرات عند الانتهاء من تنفيذ الدالة.

مثال:

اكتب مقطعاً برمجياً يمكن من لحساب المتوسط الحسابي.

```
float func (float y )
{
    static int n =0;
    static float x = 0.0;
     x +=y;
    return x/n++;
}
```

16- الأعضاء الساكنة في الصف:

- تعريف أعضاء البيانات الساكنة:

يتمّ التصريح عن أعضاء البيانات الساكنة ضمن الصف باستخدام الكلمة المحجوزة static. ويمكن الوصول لأعضاء البيانات الساكنة من خارج الصف بوساطة اسم الصف موصولاً باسم المتغير من خلال معامل دقة المدى (::).يمكن أن نسند قيمة للعضو البياني الساكن عند تعريفه، وإذا لم يتمّ إسناد قيمة لهذا المتغير فإنه سيتمّ إسناد القيمة 0 تلقائياً. وبعدما يتمّ تعريفه، يمكن الوصول إليه من الأعضاء الدالية فقط؛ يقال إنّ له مدى صف.

مثال:

```
class ss
{
private:
        static int n;          // declration
        ......
 };
int ss :: n = 77;          //definition
```

- الدوال الساكنة:

يُعرِّف العضو الدالي الساكن بوساطة الكلمة المحجوزة **static**.وينمّ استدعاء هذه الدالة من خلال ربط اسمها باسم الصف بوساطة معامل دقة المدى.

مثال:

```
class ss
{
private:
        static int n;          // declration
        ......
 public:
     static int func ( )       // definition function
          {
            // can access only static member data
           // statements
           }
    ..............
 };
 void main ( )
    {
        int ss :: n = 77;          //definition
        ..............
        ss :: func ( );            // function call
```

```
.....
}
```

لا يستطيع العضو الدالي الساكن التعامل مع أي عضو بياني غير ساكن في تلك الصف. ويستطيع العضو الدالي الساكن فقط الوصول إلى أعضاء البيانات الساكنة.

- الوسيط المرجعي:

يعرف الوسيط المرجعي بإضافة المعامل (المحرف) & إلى نوع بيانات الوسيطة في تعريف الدالة. إن الوسيط المرجعي هو اسم مستعار للمتغير الأساسي الموجود في أحد أجزاء البرنامج.

التمرير بالمرجع:

يمكن التمرير بالوسطاء المرجعية من الوصول إلى المتغيرات الأصلية والتعامل معها بدلاً من إنشاء متغيرات جديدة.

مثال:

```
void swap (int & , int & );       //declaration
......
int x;
int y;
.........
swap(x, y);
.....
void swap (int & n, int & m )       // function swap
      {
       int t = n;
           n = m;
           m = t;
      }
```

تمكن هذه الدالة من التبديل بين قيمة الوسيط المرجعي n وقيمة الوسيط المرجعي m ، وبالتالي التبديل بين قيمة المتغير الاصل x والمتغير الاصل y .

17- التبديل بين قيم (محتوى) الكائنات:

تمكن لغة البرمجة c++ من التبديل بين محتوى الكائنات، حيث يتمّ تمرير الكائنات بالمرجع.

مثال:

```
// swap.cpp
# include <iostream.h>
#include <conio.h>

        class string1
        {
          private:
                char name[20]
                int serial_number;
          public:
                void inpu( )
                  {
                   cout<<"Enter name:";
                    cin>>name;
                   cout<<"\n Enter serial_number:";
                    cin>> serial_number;
                    }
                void output ( )
                    {
                      cout<< " Name =";
                cout<<"\n  serial number = " << serial_number;
                      }
        };
      void main ( )
        {
            void swap (string1  &, string1 & ) ;    // declaration
string1 obj1, obj2;        //create two variables (objects)
```

```
    cout<<"Enter string1 1 data "<< endl;
    obj1.input( );
    cout<<"Enter string1 2 data "<< endl;
    obj2.input( );
    cout<<" swaping string1 data";
    swap ( obj1, obj2);
    cout<<"\n   data string1 1  "<< endl;
    obj1.output( );
    cout<<"\n data string1 2  "<< endl;
    obj2.output( );
}
```

ينشئ البرنامج في **main()** كائنين **string1** ويطلب من المستخدم تزويده بالبيانات لهما. ثم يستدعي الدالة **swap()** للتبديل بين بيانات هذين الكائنين.

تمرين

1. اكتب برنامجاً لحساب مجموع أول 100 حد من المتسلسلة غير المنتهية
$$\sum_{1}^{\infty} \frac{x^2 y^2}{n}$$
2. اكتب برنامجاً لحساب 100 حد من متسلسلة متناوبة.
3. اكتب برنامجاً يقوم بحساب مساحة مثلث علمت أضلاعه.
4. اكتب برنامجاً يقوم بحساب مساحة مثلث علم فيه طول الوتر وأحد ضلعيه القائمين.
5. اكتب برنامجاً لحساب مساحة شبه منحرف.
6. اكتب برنامجاً لحساب مساحة أو حجم كرة أو قبة كروية.
7. اكتب برنامجاً لحساب مساحة قطاع دائري.
8. اكتب برنامجاً لحساب مساحة مخروط.
9. اكتب برنامجاً لحساب مجموع أول 100 حد من منشور $sh(x)$ و $ch(x)$ حسب ماكلوران وقارن إذا كان النشر في جوار (1.1)


وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل السابع

الباني والمهدّم

1- تعريف الباني:

الباني هو عضو دالي خاص يتمّ استخدامة لإسناد بيانات الكائنات، لايملك قيمة إعادة ولاحتى نوعاً، ويتمّ استدعاؤه تلقائياً. يحمل الباني اسم الصف نفسه.

2- تعريف المهدّم:

المهدّم هو عضو دالي يتمّ استخدامه من أجل التخلص من كائن ما، ويتمّ استدعاؤه تلقائياً. يحمل المهدّم اسم الصف نفسه، لكن تسبقه المعامل ~ .

إن كل كائن يتمّ إنشاؤه يتمّ تدميره في وقت ما. الكائنات التي يتمّ إنشاؤها ضمن جسم الدالة يتمّ تدميرها عندما تعود الدالة. هذا الأمر صحيح مع الدالة **main()**، كما هو صحيح مع كل الدوال الأخرى. وبالعكس، الكائنات المصرحة كمتغيرات خارجية أو ساكنة يتمّ تدميرها عند انتهاء البرنامج فقط. يهتمّ المهدّم بإلغاء تخصيص مساحة الذاكرة التي كان الباني قد خصصها للكائن.

مثال:

```
//condes.cpp
//constructors and destructors
# include <iostream.h>
class omega
  {
    public:
        omega ( )
            {
               cout <<"I am the constructor"<<endl;
            }
```

```
        ~omega ( )
          {
            cout <<"I am the destructor"<<endl;
          }
};
void main ( )
        {
          cout <<"starting main "<<endl;
          omega obj1, obj2, obj3;          // create objects
          cout <<"end main "<<endl;
        }
```

يوجد استدعاء للدوال في **main()**، للباني أو للمهدّم. فاستدعاؤهما يتمّ تلقائياً. يتمّ استدعاء الباني كلما تمّ إنشاء كائن جديد. وقد حصل هذا ثلاث مرات في العبارة:

```
starting main
I am the constructor
I am the constructor
I am the constructor
end main
I am the destructor
I am the destructor
I am the destructor
```

عند انتهاء **main()**، عند الوصول إلى القوس الحاصر الغالق، ينتهي مدى الكائنات الثلاثة ويتمّ تدميرها تلقائياً. يتمّ استدعاء المهدّم لكل كائن قبل أن يتمّ تدميره. تظهر رسائل المهدّم بعد انتهاء البالة الرئيسة **main()**.

مثال:

إسناد قيم للمتغيرات باستخدام الباني.

```
// strustak.cpp
# include <iostream.h>
# include <conio.h>
struct  stack
```

```
{
 private :
     int st[20];
      int top;
  public :
        stack ( ) : top (-1);              // constructor
                 {      }
      void push (int var )          // place an item on the stak
            {
              st[++st.top] = var ;
              }
      int pop( )            // remove an item from the stak
            {
              return st[st.top--];
              }
 };
void main ( )
       {
          Stack s1;                    // Objects
           s1.push(11);
           s1.push(12);
           s1.push(13);
           cout<<s1.pop( ) <<endl;
           cout<<s1.pop( ) <<endl;
           cout<<s1.pop( ) <<endl;
        }
```

لقد تخلصنا من العضو الدالي **init()** ومن كل استدعاءاته في **main()** في البرنامج **stack1**.

يتمّ استخدام تركيب نحوي جديد، نقطتان تليان اسم الصف وذكر أسماء المتغيرات المطلوب إسناد قيم لها بعد النقطتين (إذا كان هناك أكثر من متغير واحد ، يمكن كتابتها على أن تفصلها فواصل بينها). والقيمة المستخدمة في الإسناد لكل متغير في

اللائحة توضع في أقواس تلي الاسم. إن جسم الباني فارغ لأن الإسناد قد تمَّ في لائحة الإسناد. إن العديد من البواني تقوم بأعمالها في جسم الدالة.

ملاحظة:

إن الباني الافتراضي لا يأخذ وسطاء.

3- باني ثنائي الوسطاء:

يمكن إسناد قيمة محددة للكائن بوساطة استخدام وسطاء في الباني، وذلك عند إنشاء الكائن.

مثال:

تمثيل الأوقات برمجياً وذلك باستخدام الباني:

```
// time.ccp
# include <iostream.h>
    class time1
    {
      private:
            int hours;        //0 --23
            int minutes;    // 0 --59
      public:
          //two argument constructor
            time1 (int h, int m) : hours( h ), minutes ( m);
              {
              }
            void display ( )
                {
                  cout<< hours<<":"<< minutes;
                }
    };
void main ( )
    {
```

```
    time1 obj (5, 30);          //initialize (objects)
    cout<<"\n obj1 =";
    obj1.display( );            // display obj1
}
```

لقد تمّ إسناد قيمة للكائن **obj1** . ويتمّ تمرير القيم إلى الباني بوضعها في تعريف الكائن.

ملاحظة:

إذا لم نعرّف الباني في البرنامج فإن المترجم يولد بانياً افتراضياً.

4- الباني الأحادي الوسيط:

يمكن استخدام الباني الذي له وسيطة واحدة لتحويل كائن ما من صف إلى أخرى. يتمّ هذا عادة للصفوف التي تمثل أنواع البيانات، حيث يمكن استعمال مشيداً أحادي الوسيط لتحويل قيمة من نوع بيانات ما إلى نوع آخر.

مثال:

```
class ss1
    {
      ss1 (int i);        // one orgument constructor
          {
            // convert the int value to ss1 value
          }
    };
```

يمكن استخدام الباني لتحويل متغير من النوع **int** إلى كائن من الصف **ss1**.

هناك طريقتان لكتابة هذا تحويل.

- إسناد قيمة للكائن الدالة للصف ss1:

```
void main ( )
{
int j=14;       // integer value
ss1 obj (j);    // initialize ss1 object
```

```
}
```

- استخدام معامل =

```
void main ( )
{
int j=14;       // integer value
ss1 obj = j ;   // initialize ss1 object
}
```

ملاحظة:

يمكن استخدام الباني أحادي الوسيط لتحويل سلسلة محارف تنتهي برمز خامد إلى كائن obj1.

مثال:

```
// string.cpp
// string ; uses constructor
# include <iostream.h>
#include <conio.h>
      const int max = 60;              // max length string
         class string1
         {
           private:
                char name[max]
           public:
              string1 ( )              // consructor
                {
                  strcpy (name, " ");   // make nul string
                 }
              string1 (char st[ ] )          // 1- arg. consructor
                {
                  strcpy (name, st);   // initialize with  string
                 }
              void input ( )
                  {
```

```
                cin.get (name, max);
                }
        void display ( )
                {
                cout<< name;
                }
 void append(strig1 xst )     // append argument string
                {
        if (strlen(name) + strlen(xst.name) < max – 1)
                strcat ( name, xst.nam);
                  else
                 cout<<"\n   error string1 too long "<<endl;
                }
   };
void main ( )
   {
 string1 obj2, obj3;          //create two variables (objects)
        string1 obj1 (" My last name is : ");
            cout<<"Enter your last name "<< endl;
            obj2.input( );
            obj.append(obj2 );
            obj3 = obj1;
            obj1.display( );
     }
```

تمّ تحويل سلسلة المحارف العادية " My last name is :" إلى سلسلة محارف كائن.

ملاحظة:

لعرض علامة الاقتباس الفردية (') وعلامة الاقتباس المزدوجة (") نستخدم محرف الهروب \ ، أي:

```
Cout<<" \' "<< "text"<<" \' "<<"and"<<" \" "<<"number"<<" \" ";
```

الخرج:

```
' tex ' and "number"
```

5- باني النسخ:

يُمكن باني النسخ طريقة أخرى لإنشاء الكائنات؛ يأخذ نسخة عن كائن موجود. ويمكن تمرير الوسطاء بالمرجع.

مثال:

نبين استخدام باني نسخ كائنات عند التمرير بالمرجع، لم يتمّ تعديل الوسيطاء الأصلية. من خلال التصريح من هذه الوسطاء كثابتة (const).

```
//copy.cpp
// copy constructors
# include <iostream.h>
class omega
  {
   private:
       int ivar;
    public:
        omega ( int v ) : ivar(v)    // one org. constructor
            {
            }
omega (const omega & obj1 )      // copy  constructor
            {
              ivar = obj1.ivar;
              cout<<"\n I am the copy constructor";
            }
  };
 void main ( )
        {
          omega obj2 ( 13 );    // uses one org. constructor
           omega obj3 = obj2;
```

```
omega obj4 (obj2);
}
```

لدينا باني نسخ للصف **omega**: باني أحادي الوسيط وباني النسخ. باني النسخ له اسم الصف نفسه وليس له قيمة إعادة. لكن وسيطه من الصف نفسه التي تكون الدالة عضواً فيها. تمثل الوسيطة الكائن المطلوب نسخه. يستخدم البرنامج باني النسخ مرتين: عند إنشاء **obj3** وعند إنشاء **obj4**. ضرورية لنسخ البيانات من كائن إلى آخر. إذا لم نعرف باني نسخ، يولد المترجم بانياً افتراضياً يقوم بنسخ هذه البيانات تلقائياً، باستعمال النسخ العضوي. لكن إذا عرفنا باني نسخ، فإننا مسؤولون عن تنفيذ النسخ أعضاء البيانات.

6- تعريف الكائن الثابت:

يمكن تعريف الكائن الثابت كما يلي:

```
const omega obj1;
```

فبذلك نضمن عدم تغير البيانات في الكائن.

7- تعريف الدالة الثابتة:

تعرف الدالة الثابتة بتحديد الكلمة المحجوزة **const** في تعريفها (وتصريحها أيضاً، إذا كان هناك واحد) مباشرة بعد الأقواس التي تلي اسمها. الدالة الثابتة عضو دالي يضمن أنه لن تغير بيانات الكائن الذي استدعيت من أجله.

- التركيب النحوي:

```
class ss
{
....
Void display ( ) const;        // const in declaration
.....
};
```

```
void ss :: display ( ) const    // const in definition
 {
    // statements;
  }
```

مثال:

```
// time.ccp
# include <iostream.h>
    class time1
    {
      private:
            int hours;          //0 --23
            int minutes;     // 0 --59
      public:
                  // constructor
            time1 ( ) : hours( 0 ), minutes ( 0);
               {
                 }
                      // copy construct
            time1 (int h, int m) : hours( h ), minutes ( m);
               {
                 }
    void display ( )  const;       //declaration const function
            viod get ( );        //declaration const function
    };
            void time1 :: display ( )  const     //for const object
                 {
                   cout<< hours<<":"<< minutes;
                  }
            void time1 :: get ( )
                 {
                   char ch;
```

```
                cout<< "\n Enter time ( form  12:59) : ";
                 cin >>hours >> ch >> minutes;
                }
void main ( )
   {
      const  time1 obj (12, 0);          //create  objects
      cout<<"\n obj1 =";
      obj1.display( );          // display obj1
      time1 obj2;
      obj2.get ( );
      cout<<"" obj2 = ";
      obj2.display ( );
      }
```

إن استخدام كلمة **const** مع أي تصريح عن دالة، فإنها تصبح جزءاً من اسم الدالة.

إن الدالة **void display() const** ليست نفسها الدالة **void display()**.

ملاحظة:

لا تكون البوانى ثابتة أبداً ذلك أنّ إنشاء الكائنات يتطلب تغيير **const**.


وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل الثامن

تحميل العوامل بشكل زائد

1- مقدمة:

تمكن لغة البرمجة C++ من التحميل الزائد للمعاملات + و =< و ++ و – و * و / و> و=+ و [ ] . أي أن العامل + في التعبير a + b سيستدعي دالة ما إذا كان a وb أعداداً صحيحة، لكنه سيستدعي دالة أخرى إذا كان a وb كائنات. سنبين كيفية تحميل المعاملات في لغة البرمجة C++ بشكل زائد وكيفية عملها على الكائنات التابعة للصف.

2- تحميل العوامل الحسابية الثنائية بشكل زائد:

تقسيم المعاملات في لغة البرمجة C++ إلى ثنائية وأحادية. تأخذ المعاملات الثنائية وسيطتين، a+b وa-b وa/b . أما العوامل الأحادية فتأخذ وسيط واحد:a- وa+ و--a.

- تحميل الحامل + و-:

يمكن تحميل المعامل + بشكل زائد للكائنات وفق مايلي:

نستخدم الكلمة المحجوزة **operator** يليها المعامل نفسه لتشكيل اسم الدالة. لتحميل العامل + بشكل زائد، يصبح الاسم operator+ (تليه أقواس للدلالة على أنه دالة).

- التركيب النحوي

```
class ss
 {
  ....
 s operator + (ss  ob)
  {
```

```
        // statements;
        }
    ..............
    };
```

لتحميل العامل – بشكل زائد، يصبح الاسم operator – .

– التركيب النحوي

```
ss operator – (ss sub)
{
//statements;
}
```

لتحميل العامل * بشكل زائد، يصبح الاسم operator * .

– التركيب النحوي

```
ss operator * (ss sub)
{
//statements;
}
```

لتحميل العامل / بشكل زائد، يصبح الاسم operator / .

– التركيب النحوي

```
ss operator / (ss sub);
```

مثال:

```
//overload operator
// time.ccp
# include <iostream.h>
    class time1
    {
     private:
```

```
        int hours;        //0 --23
        int minutes;    // 0 --59
    public:
        void  display ( )  const    //for const object
            {
              cout<< hours<<":"<< minutes;
             }
        void time1 :: get ( )
            {
              char ch;
              cout<< "\n Enter time ( form  12:59) : ";
               cin >>hours >> ch >> minutes;
              }
        time1 operator + (time1 ob)
              {
               time1 temp;
              temp.hours = hours + ob.hours;
              temp.minutes = minutes + ob.minutes;
              if (temp.minutes >= 60)
                 {
                   temp,hours++;
                   temp.minutes - = 60;
                  }
                   return temp;
            }
        };
void main ( )
   {
      time1 obj1, obj2, obj3;          //create  objects
      cout<<" Enter first time : ";
       obj1.get ( );
      cout<<" Enter second time : ";
       obj2.get ( );
```

```
obj3 = obj2 + obj1;          //overload operator
cout<<"\n sum =";
obj3.display( );             // display obj1
time1 obj2;
obj2.get ( );
cout<<"" obj2 = ";
obj2.display ( );
}
```

3- تحميل المعاملات العلائقية:

يمكن تحميل المعامل > بشكل زائد لمقارنة قيمتين لكائنين وإعادة قيمة من النوع Boolean.

- التركيب النحوي

```
class ss
{
....
bool operator < (const ss & obj)
{
//statements;
}
.....
};
```

4- معامل التعيين:

يمكن تحميل معامل التعيين بشكل زائد، التي تتضمن =+ و =- و =* و =/.

- التركيب النحوي

```
class ss
{
....
ss operator += (const ss & obj)
{
//statements;
}
.....
};
```

بطريقة مماثلة يمكن من تحميل معامل التعيين الأخرى =- و =* و =/ وغيرها بشكل زائد.

5- تحميل العوامل الأحادية بشكل زائد:

يمكن تحميل المعاملات الأحادية على قيمة واحدة بشكل زائد. معامل التزايد (++) والتناقص (--) ومعامل النفي (!) المنطقي.

- تحميل الإصدار المتصدر للعامل ++:

لا تأخذ المعامل الأحادية وسطاء. إنها تعمل على الكائن الذي تمَّ استدعاؤه من أجله. يظهر المعامل في الجهة اليسرى للكائن، **obj!** و**obj-** و**obj++**.

- التركيب النحوي

```
class ss
{
....
ss operator ++ ( )
{
```

```
//statements;
}
.....
};
```

- **الإصدار اللاحق للعامل ++:**

يميز المترجم بينَ الإصدارين المتصدر واللاحق بوساطة وسيط وهمي من نوع **int** للإشارة إلى أنه الإصدار اللاحق.

- **التركيب النحوي**

```
class ss
{
....
ss operator ++ (int )
{
//statements;
}
.....
};
```

لا تزود الدالة قيمة لهذا الوسيط الوهمي، فمجرد شملها يعرف المترجم أنه الإصدار اللاحق.

6- تحميل معامل التعيين (=) بشكل زائد:

يمكن تحميل معامل التعيين (=) بشكل زائد. لكن معامل التعيين يلعب دوراً أكبر بكثير من دور معظم المعاملات لأنه ينفذ عملية أساسية جداً ويتمّ استعماله بكثرة.

التركيب النحوي لعامل التعيين المحمل بشكل زائد:

```
class ss
{
....
void  operator = (const ss & obj )
{
//statements;
}
.....
};
```

تأخذ هذه الدالة وسيطاً واحداً تابعاً لصفها، يتمّ تمريرها بالمرجع الثابت. عندما يكون هناك قيمة إعادة عقيمة (void).

7- تحميل المعامل [ ] بشكل زائد:

إن معامل الدليل ([ ])، الذي يتمّ استخدامة للوصول إلى عناصر المصفوفة، يمكن تحميله بشكل زائد.

```
class ss
{
int  & operator [ ] (int n )
{
//statements;
}  };
```


وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل التاسع

الوراثة

1- تعريف الوراثة:

تعرف الوراثة بأنها اشتقاق صف جديدة من صف واحدة قديم. ولا يتمّ تعديل الصف القديم (المسما الصف الأب)، لكن الصف الجديد (الصف الأبن) يمكنها استعمال كل ميزات الصف القديم وميزات إضافية خاصة به.

- التركيب النحوي للوراثة:

نعرف التركيب النحوي للوراثة على فئتين: صف قاعدة وصف مشتقة. لا تحتاج الصف القاعدة إلى أي تركيب نحوي خاص. أما الصف المشتقة فيجب أن نشير إلى أنها مشتقة من الصف القاعدة. يتمّ هذا بوضع نقطتين بعد اسم الصف المشتقة، تليها كلمة المحجوزة **public** أو **private** أو **protected** ، ثم اسم الصف القاعدة كما يلي:

```
class A
{
// member data and member function
};
class B : public A
{
// member data and member function
};
```

النقطتان تعني "مشتق من". لذا فإن الصف **B** مشتقة من الصف **A** وترث كل البيانات والدوال الموجودة فيها.

مثال:

اكتب برنامجاً يمكن من استخدام الصف B (تمكن من إدخال عدد صحيح) التي ترث الصف A والتي تمكن من إدخال عدد حقيقي وعرض الناتج.

```
// inherit.cpp
// inheritance program
# include <iostream.h>
class A
    {
        private:
              float fvar;
        public:
             void A_get( )
                    {
                      cout<<"\n Enter the value : ";
                       cin>>fvar;
                    }
              void A_display ( )
                    {
                      cout<< " fvar = "<<fvar;
                    }
    };
class B : public A
    {
        private:
              int ivar;
        public:
             void B_get( )
```

```
{
  A_get( );
   cout<<"\n Enter the value : ";
  cin>>ivar;
}
void B_display ( )
{
  A_display ( );
  cout<< " ivar = "<<fvar;
  }
};
void main ( )
{
 B obj;
  cout<<"Requestig data from B object "
  obj.B_get( );
   cout<<"Data in B object is";
   obj.B_display ( );
}
```

ملاحظة:

إن الكائن التابع لصف مشتقة يرث كل الأعضاء البيانية والدالية الموجودة في الصف القاعدة.

2- الوصول إلى بيانات الصف القاعدة:

قد يحتوي الكائن التابع لصف مشق على بيانات معرفة في الصف القاعدة، فإن ذلك لا يعني بالضرورة أن هذه البيانات يمكن الوصول إليها من الصف المشتقة.

```
class B
{ ..............
  void B_display ( )
     {
  cout<< " fvar = "<<fvar;   //error private member of class A
  cout<< " ivar = "<<fvar;
          }
    };
```

لا يمكن الوصول إلى fvar من الصف B، لأن fvar هو عضو خاص (private) للصف A. يمكن الوصول إلى الأعضاء العامة (البيانات والدوال) من خارج الصف، لكن ليس الأعضاء الخاصة.

3- التحميل الزائد للدوال في الصف القاعدة والصف المشتقة

يمكن تحميل الدوال بشكل زائد في الصف القاعدة وفي الصف المشتقة.

```
// inherit.cpp
// inheritance program
// overload member function
# include <iostream.h>
class A
    {
      private:
            float fvar;
      public:
           void  get( )
                 {
                    cout<<"\n Enter the value : ";
                    cin>>fvar;
```

```
                    }
            void  display ( )
                    {
                     cout<< " fvar = "<<fvar;
                    }
    };
class B : public A
    {
       private:
            int ivar;
       public:
            void  get( )
                    {
                     A :: get( );
                     cout<<"\n Enter the value : ";
                    cin>>ivar;
                    }
            void  display ( )
                    {
                    A :: display ( );
                    cout<< " ivar = "<<fvar;
                    }
    };
void main ( )
    {
     B obj;
     cout<<"Requestig data from B object "
      obj. get( );
```

```
    cout<<"Data in B object is";
    obj. display ( );
}
```

ملاحظة:

يمكن لعدة صفوف أن ترث صف. ويمكن لصف أن يرث عدة صفوف.

4- الوراثة العامة:

نستخدم المحدد **public** للوصول للأعضاء العامة، نسمى هذه الوراثة وراثة عامة. تستطيع كائنات الصف المشتقة الوصول إلى الأعضاء العامة التابعة للصف القاعدة. في الوراثة العامة، تعني أن كائناً دالةاً للصف **B** هو كائن "نوع من" دالة للصف **A**. إن كائن الصف المشتقة له كل الميزات الصف القاعدة، بالإضافة إلى بعض الميزات الخاصة به.

```
class A
{
// member data and member function
};
class B : public A
{
// member data and member function
};
```

5- الوراثة الخاصة:

نستخدم المحدد **private** للوصول للأعضاء الخاصة، نسمى هذه الوراثة وراثة خاصة.

```
class A
```

```
{
// member data and member function
};
class B : private  A
{
// member data and member function
};
```

لا تستطيع كائنات الصف المشتقة الوصول إلى الأعضاء العامة في الصف القاعدة.

6- الوراثة المتتالية:

يمكن للصفوف أن ترث بعضها بالتتالي وفق عدة مستويات وراثية. يستطيع الصف الوصول إلى كل أسلافها. في الوراثة العامة، تستطيع الأعضاء الدالية التابعة للصف B الوصول إلى البيانات العامة أو المحمية في C و D وA. لكن لا يمكنها الوصول إلى الأعضاء الخاصة التابعة لأي صف ما عدا أعضائها.

```
class A                          // first generation
{
 // member data and member function
};
class B : public  A                    // second  generation
{
// member data and member function
};
class C : public B                              // third  generation
{
// member data and member function
};
```

```
class D : public  C                    // fourth  generation
{
// member data and member function
};
```

7- الوراثة المتعددة:

تحدث الوراثة المتعددة عندما ترث صف ما من فئتين قاعدتين أو أكثر، كالتالي:

```
class A
{
// member data and member function
};
class B
{
// member data and member function
};
class C : public A , public B          // third  generation
{
// member data and member function
};
```

الصف C مشتقة من الفئتين A والصف B. يتمّ في مواصفات الصف المشتقة فصل الصفوف القاعدة عن بعضها بعضاً بواسطة فاصلة ويكون هناك محدد وصـول لكـل صف قاعدة.

الفصل العاشر

المؤشرات

1- مقدمة:

يمكن المؤشرات من التخصيص الديناميكي للذاكرة، وذلك من خلال الكلمة المحجوزة **new** والكلمة المحجوزة **delete**.إن المؤشر **this** يشير إلى كائنه الخاص به. تستخدم المؤشرات لإنشاء بنيات معقدة لتخزين البيانات. وتمكن من الاستخدام الأمثل للذاكرة : صف سلاسل والقوائم المرتبطة ونوع من المصفوفات ذاتي الفرز.

2- العناوين والمؤشرات:

إن الفكرة الأساسية من المؤشرات بسيطة. تحتاج إلى معرفة عناوين الذاكرة، وأنه يمكن تخزين العناوين كالمتغيرات.

3- العناوين (الثوابت المؤشرة):

لكل **Bytes** في ذاكرة الحاسوب عنوان. تبدأ الأرقام من 0 وتتصاعد متسلسلة المحارفاً. إذا كان لدينا 1 ميغابايت من الذاكرة، فإن العنوان الأعلى هو **1.048.575**.

عند وضع أي برنامج في الذاكرة يحتل مجالاً معيناً من العناوين في الذاكرة. وهذا يعني أن كل متغير ودالة في البرنامج يبدأ عند عنوان ما.

4- معامل العنوان &:

يمكن معامل العنوان & من معرفة العنوان الذي يحتله متغير ما.

مثال:

```
// variable.cpp
```

```
// Addresses
# include <iostream.h>
void main ( )
        {
        int x = 5;
        int y = 13;
        int z = 33;
       cout<<endl<<&x
          <<endl<<&y
          <<endl<<&z;          }
```

تعتمد العناوين الفعلية التي تحتلها متغيرات البرنامج على عدة أمور، كالجزء من الذاكرة الذي يشتغل فيه البرنامج، وحجم نظام التشغيل، والبرامج الأخرى التي تحجز حيزاً في الذاكرة. لذلك قد لا تحصل على العناوين نفسها التي نحصل على عناوين مختلفة عند كل تشغيل لهذا البرنامج. إن عنوان المتغير ليس نفسه محتوياته. فمحتويات المتغيرات الثلاثة هذه هي 5 و13 و33. يخرج المعامل (<<) العناوين بالنظام الست عشري، ويشار إلى ذلك من خلال المحارف **0x** التي تظهر قبل كل رقم. إنها طريقة إظهار عناوين الذاكرة.

ترتب عناوين المتغيرات التلقائية بترتيب تنازلي يتمّ تخزينها في المكدس، الذي ينمو إلى الأسفل في الذاكرة. أما المتغيرات الخارجية ترتب عناوينها بترتيب تصاعدي، لأنه يتمّ تخزين المتغيرات الخارجية في الذاكرة الكومة، التي تنمو إلى الأعلى.

5- متغير مؤشر:

يمكن تخزين عنوان المتغير في متغير. إن المتغير الذي يخزن قيمة عنوان ما يسمى متغير مؤشر. إذا كان المؤشر يحتوي على عنوان متغير ما، فإن المؤشر يشير إلى المتغير.

6- مؤشر متغير الأنواع الأساسية:

التركيب النحوي للمتغير المؤشر الذي يخزن عناوين متغيرات من النوع الأساسي **int**.

Type of variable * pointer name;

مثال:

اكتب برنامجاً بلغة C++ يمكن من تعريف متغيرين صحيحين ومعرفة عناوين هذه المتغيرات في الذاكرة ثم تعريف مؤشر يؤشر إلى عناوين هذه المتغيرات.

يستخدم المؤشر للتأشير على عنوان في الذاكرة أو إلى معلومات موجودة في هذا العنوان.

الحل:

```
#include <iostream.h>
#include <conio.h>
void main ()
{
        //clrser();
        int x,y,z;
        int m=15,n=20;
        int * ptr;
        cout << &m << endl;
        cout << endl;
        cout << &n<< endl;
        cout << endl;
        ptr = &m;
        cout << ptr << endl;
        ptr = &n;
        cout << ptr << endl;
        getch();  }
```

النجمة تعني مؤشراً إلى. لذا تعرف تلك العبارة المتغير **ptr** كمؤشر إلى النوع **int**.

ملاحظة:

يمكن أن نعرف أكثر من مؤشر واحد من النوع نفسه في سطر واحد، على أن نضع النجمة قبل اسم كل متغير:

```
char *prt1, *prt2;
int *prt1, *prt2;
float *prt1, *prt2;
```

7- مؤشرات إلى الكائنات:

يمكن للمؤشرات التأشير إلى الكائنات. يمكن أن نعرف كائناً ثم نضع عنوانه في مؤشر ما:

```
//pointerprog.cpp
// pointer to object
# include <iostream.h>
# include <string.h>
class dataclass
  {
   private:
        enum { n =20 };
        char name [n];
         unsigned long telnum;
   public:              // 2 arg. constructor
        dataclass (char * na , unsigned long nu) : telnum (nu)
          {
           strcpy(name, na);
            }
  };
void main ( )
```

```
{
dataclass obj1 ("Donl" , 123123L);
 dataclass obj2 ("LBlance" , 123123L);
  cout<< "Address variable "
  cout << endl << &obj1;
  cout << endl << &obj2;
  dataclass * ptr;
  cout<<" \n pointer values ";
  ptr = & obj1;
   cout << endl << ptr;
   ptr = & obj2;
   cout << endl << ptr;
}
```

المتغير obj1 , obj2 هو من النوع dataclass *، أي أنه مؤشر إلى كائن دالة للصف dataclass . إنّ كائنات dataclass تحجز مساحة أكثر من المتغيرات العددية الصحيحة.

8- الوصول إلى المتغير المشار إليه:

يمكن الوصول إلى قيمة المتغير باستخدام عنوانه بدلاً من اسمه.

مثال:

```
// accessprog.cpp
// variable pointed
# include <iostream.h>
void main ( )
 {
  int x = 5;
  int y = 12;
```

```
int * ptr;
ptr = & x;
cout<<endl<< *ptr;
ptr = & y;
cout<<endl<< *ptr;
}
```

إن البرنامج لايخرج قيمة العنوان المخزنة في المؤشر **ptr**، ولكن يخرج القيمة العددية الصحيحة المخزنة عند العنوان المخزن في **ptr**. إن التعبير الذي يتيح الوصول إلى المتغيرين **var1** و**var2** هو ***ptr**، الذي ظهر في العبارتين **cout**. عند استخدام النجمة على يسار اسم المتغير، تسمى معامل الموازبة **(indirection)** وهذا يعني قيمة المتغير الذي يشير إليه المتغير المذكور على يمينه. يمكن استخدام المؤشر ليس فقط لعرض قيمة متغير ما، بل لتنفيذ أي عملية عليه بشكل غير مباشر.

مثال:

اكتب برنامجاً يستخدم مؤشراً لإسناد قيمة إلى متغير ثم لإسناد تلك القيمة إلى متغير آخر.

```
// accessprog.cpp
//using  pointer
# include <iostream.h>
void main ( )
 {
  int x, y;
  int * ptr;
  ptr = & x;
* ptr = 15;
 y = * ptr;
  cout<<endl<< y;
}
```

إن النجمة عند استخدامها كمعامل موازبة لها معنى مختلف عن معناها عن استخدامها لتصريح المتغيرات المؤشرة. يسبق معامل الموازبة اسم المتغير ويعني قيمة المتغير المشار إليه. أما النجمة المستعملة في التصريح فتعني مؤشراً إلى.

```
int * ptr;        //declaration
* ptr = 15;       // indirection      معامل الموازبة
```

إن استخدام معامل الموازبة للوصول إلى القيمة المخزنة في عنوان ما يسمى عنونة غير مباشرة.

9- مؤشرات إلى void:

يمكن أن نستخدم المؤشرات العقيمة، يمكنها أن تشير إلى أي نوع بيانات. إنها تسمى مؤشرات إلى **void**، و تعرف كما يلي:

```
void *ptr ;       //ptr can pointed to any data type
```

تستخدم هذه المؤشرات لتمرير المؤشرات إلى الدوال التي تعالج عدة أنواع بيانات مختلفة.

مثال:

```
// ptrprog.cpp
//using  pointer of void
# include <iostream.h>
void main ( )
 {
  int x;
  float y;
  int * ptri;
  float * ptrf;
  void * ptrv;
```

```
ptri = & x;        // int* to int*
ptrf = & y;        // float* to float*
ptrv = & x;        // int * to void*
ptrv = & y;          // float * to void*
}
```

يمكن تعيين عنوان المتغير إلى المؤشر **ptri** لأنهما من النوع ***int**، لكن لا يمكن تعيين عنوان المتغير y إلى المؤشر **ptri** لأن الأول من النوع ***float** والثاني من النوع ***int**. لكن يمكن تعيين أي نوع مؤشرات إلى المؤشر **ptrv**، كـ ***int** أو ***float**، كونه مؤشراً إلى **void**.

10- المؤشرات والمصفوفات والدوال:

يمكن استخدام المؤشرات للوصول إلى عناصر المصفوفة واستخدامها كوسطاء للدوال. وتظهر أهمية المؤشرات عند تمرير المصفوفات كوسيطات للدوال.

- المؤشرات والمصفوفات:

تمكن المؤشرات من الوصول إلى عناصر المصفوفة.

مثال:

```
int array1[3] = {12, 33, 65};
for (int i=0 ; i<3 ; i++)
cout<<endl<<*(array1+i);
```

فيكون الخرج

```
12
33
65
```

تمّ الوصول إلى عناصر المصفوفة باستخدام تدوين المؤشرات وتدوين المصفوفات.

التعبير (array1+j)* هنا له نفس تأثير التعبير arrary1[i].

للحصول على قيمة عنصر المصفوفة الثالث، وليس عنوانه. لأخذ القيمة، نستخدم معامل الموارية *. التعبير الناتج وهو محتويات عنصر المصفوفة الثالث، 65.

يجب أن يتضمن تصريح المؤشر نوع المتغير الذي يشير إليه. يحتاج المترجم إلى معرفة ما إذا كان المؤشر مؤشراً إلى **int** أو مؤشراً إلى **double** لكي يتمكن من تنفيذ العملية الحسابية الصحيحة من أجل الوصول إلى عناصر المصفوفة.

- المؤشرات والدوال:

هناك ثلاث طرق لتمرير الوسطاء إلى الدالة: بالقيمة، وبالمرجع، وبالمؤشر. يمكن استعمال التمرير بالمرجع أو بالمؤشر للسماح للدالة تعديل الوسيط الممررة إليها.

مثال:

اكتب برنامجاً يمكن من التحويل من انش الى سم وذلك باستخدام التمرير بالمرجع

```
//covertprog.cpp
# include <iostream.h>
void main ( )
 {
   void centim ( double *);      // prototype
    double y = 10.0;             // inches
    cout<< "y= "<<y <<"inches";
   centim (y);          //change value
    cout<< "y= "<<y <<"centimiters";
 }
void centim (double  & x)
 {
  x *=2.54;             // *ptrd is same as y
```

```
}
```

مثال:

اكتب برنامجاً يمكن من التحويل من انش الى سم وذلك باستخدام المؤشر

```
//covertprog.cpp
# include <iostream.h>
void main ( )
 {
   void centim ( double *);      // prototype
    double y = 10.0;             // inches
    cout<< "y= "<<y <<"inches";
   centim (&y);          //change value
   cout<< "y= "<<y <<"centimiters";
 }
void centim (double * ptrd)
 {
  * ptrd *=2.54;              // *ptrd is same as y
 }
```

الخرج هو نفسه في كلٌّ البرنامجين. لقد تمَّ تصريح عن الدالة **centim()** على أنها دالة تأخذ وسيطاً من نوع مؤشر إلى النوع **double**.

عندما تستدعي **main()** هذه الدالة، تقوم بتزويدها عنوان أحد المتغيرات على أنه وسيطها:

```
centim(&y);
```

إن المتغير ليس نفسه، كما هو الحال عند تمريره بالمرجع، بل هو عنوانه. بما أن الدالة **centim( )** حصلت على عنوان، عليها أن تستخدم معامل المواربة، **ptrd***، من أجل الوصول إلى القيمة المخزونة في العنوان. إن المؤشر **ptrd** يحتوي على عنوان

المتغير y، لذلك فإن أي تغير يتمّ على ptrd* يتمّ على y.

إن تمرير المؤشر كوسيطة إلى الدالة يشبه بطريقة ما التمرير بالمرجع. فكلاهما يسمح تعديل المتغير الممرر في الدالة. لكن الإلية مختلفة. المرجع هو اسم مستعار للمتغير الأصلي، بينما المؤشر هو عنوان المتغير.

11- تمرير المصفوفات كوسيطات:

يمكن استخدام تدوين المؤشرات بدلاً من تدوين المصفوفات عند تمرير المصفوفات إلى الدوال.

مثال:

اكتب برنامجاً يمكن من التحويل عناصر متجهه من انش الى سم وذلك باستخدام تدوين المؤشر

```
//covertprog.cpp
# include <iostream.h>
const max = 3;
void main ( )
 {
   void centim ( double *);      // prototype
    double farray [max] = {10.0, 12.3, 55.4};          // inches

   centim (farray);          //change elements of array
   for ( int i=0; i<max; i++)
   cout<< "farray[ "<<i<<"]="
       <<farray[i]<<"cetimiters";
 }
void centim (double * ptrd)
 {
```

```
    for ( int i=0; i<max; i++)
  * ptrd++ *=2.54;          // *ptrd is same as y
}
```

إن تصريح الدالة هو نفسه في المثالين؛ الوسيط الوحيدة إلى الدالة هي مؤشر إلى النوع **double**. يكتب هذا الأمر في تدوين المصفوفات كما يلي:

```
void centim ( double []);
```

أي أن ***double** مرادف هنا لـ **double[]**.

بما أن اسم المصفوفة هو عنوانها، لا داعي لاستخدام معامل العنوان & عند استدعاء الدالة.

12- المؤشرات وسلاسل المحارف

يمكن تعريف سلاسل المحارف باستخدام تدوين المؤشر.

مثال:

اكتب مقطعاً برمجياً يعرف سلسلتين، واحدة باستعمال تدوين المصفوفات، وواحدة باستعمال تدوين المؤشرات:

```
char str1[] = "defined as an arrary";
char str2[] = "defined as a pointer";
cout<<endl<<str1;
cout<<endl<<str2;
str2++;          //this is ok str2 is a pointer
cout<<endl<<str2;
```

إن **str1** هو عنوان ــ أي، ثابت مؤشر ــ بينما **str2** هو متغير مؤشر. لذا يمكن تغيير **str2** على عكس **str1**. يمكن زيادة **str2** لأنه مؤشر، لكن عند الزيادة لن يشير إلى المحرف الأول في السلسلة.

13- السلاسل كوسطاء دوال:

يمكن استخدام سلسلة المحارف وسيط دالة.

مثال:

```
// stringprog.cpp
# include <iostream.h>
# include <string.h>
void main ( )
 {
  void displaystr (char*);   //prototype
char str[] = "I am student";
displaystr (str);
 }
void displaystr (char * ptrs)
  {
   cout<<endl;
    while (*ptrs)          // untile null character
    cout<<*ptrs++;         // print character
  }
```

استخدمنا عنوان المصفوفة **str** كوسيط في استدعاء الدالة **displaystr()**. هذا العنوان هو ثابت لكن بما أنه ممرر بالقيمة، يتمّ إنشاء نسخة عنه في **displaystr()**. هذه النسخة هي مؤشر، تسمى **ptrs**. يمكن تغيير المؤشر لذا تقوم الدالة بزيادة **ptrs** من أجل عرض كل محرف في سلسلة المحارف. يعيد التعبير ***ptrs++** المحارف المتتالية. تتكرر الحلقة إلى أن تجد المحرف الخامل ('\0') في نهاية سلسلة المحارف. بما أن هذا المحرف قيمته 0، أي خطأ، تتوقف الحلقة **while** عنده.

14- نسخ سلسلة محارف باستخدام المؤشرات:

يمكن استخدام المؤشرات لإدراج المحارف في سلسلة المحارف.

مثال:

```
// stringprog.cpp
# include <iostream.h>
# include <string.h>
void main ( )
 {
   void copystr (char*, char* );   //prototype
char*  str1 = "I am student";
char str2[30];              // Empty string
copystr (str2, str1 );          // copy str1 to str2
cout<<endl<<str2;
 }
void copystr (char * ptrs2, char* ptrs1)
  {
    cout<<endl;
     while (*ptrs1)          // untile null character
       *ptrs2++ = *ptrs1++;     // copy character
*ptrs2 = '\0';
  }
```

تستدعي الدالة main() هنا الدالة copystr() لنسخ السلسلة str1 إلى السلسلة str2. في هذه الدالة التعبير:

```
*ptrs1++ = *ptrs1++
```

يأخذ قيمة المحرف من العنوان الذي يشير إليه ptrs1 ويضعها في العنوان الذي

يشير إليه **ptrs2**. ثم تتمّ زيادة المؤشرين، لذا سيتمّ نسخ المحرف التالي في التكرار التالي للحلقة. تنتهي الحلقة عند العثور على حرف خامل في **ptrs1**؛ عندها يتمّ إدراج محرف خامل في **ptrs2** وتعود الدالة.

15- إدارة الذاكرة بوساطة new وdelete:

- المعامل new:

المعامل **new** يخصص ذاكرة ذات حجم معين ويعيد مؤشراً إلى نقطة بداية قطعة الذاكرة.

مثال:

اكتب مقطعاً برمجياً يستخدم المؤشر **new** لحجز ذاكرة لسلسلة محارف

```
char* str = " My program ";
int len = strlen (str);
char* ptr;
ptr = new char[len + 1];        //allocate memory , size = str + '\0'
strcpy (ptr, str);
cout<<"ptr = "<<ptr;
delete[] ptr;   //release ptr's memory
```

نستخدم المؤشر **new** وفق مايلي: الكلمة المحجوزة **new** يليها نوع المتغيرات التي سيتمّ تخصيصها، وعدد المتغيرات في معقفات. يعيد المعامل **new** مؤشراً يشير إلى بداية قطعة الذاكرة. يستخدم البرنامج الدالة **strcpy( )** لنسخ السلسلة **str** إلى منطقة الذاكرة المنشأة التي يشير إليها المؤشر **ptr**. بما أن طول سلسلة المحارف صحيح ، فإن هذه السلسلة تملؤها بشكل دقيق.

يحصل المعامل **new** على الذاكرة ديناميكياً، أي أثناء تنفيذ البرنامج. يتمّ تخصيص هذه الذاكرة من منطقة تدعى ذاكرة الكومة.

- المعامل delete:

يرافق المعامل **new** معامل **delete** يعيد تحرير الذاكرة إلى نظام التشغيل. في المقطع البرمجي السبق تمّ استخدام العبارة

delete [] ptr;

تعيد إلى النظام كمية الذاكرة التي يشير إليها المؤشر **ptr**.

إن المعقفات التي تلي المعامل **delete**، تشير إلى حذف مصفوفة. إذا أنشأت متغيراً واحداً بواسطة المعامل **new**، فلن تحتاج إلى استخدام المعقفات عند حذفه:

```
ptr = new int;      //allocate a single int variable
.....
delete ptr;         // no brackets following delete
```

المعقوفات ضرورية عند حذف مصفوفة كائنات، حيث يضمن حذف كل عناصر المصفوفة ويضمن استدعاء المهدّم لكل واحد منها. فإذا لم نضع المعقوفات فسيحذف العنصر الأول فقط.

16- المؤشر this:

يمكن للأعضاء الدالية في كل كائن في لغة البرمجة C++ الوصول إلى نوع من المؤشرات **this**، الذي يشير إلى الكائن نفسه. فبذلك يستطيع أي عضو دالي معرفة عنوان الكائن الذي استدعاه.

مثال:

```
//location.cpp
//this pointer
# include <iostream.h>
class loca
    {
```

```
private:
    char charray [5];          //occupies 5 bytes
public:
     void fun1 ( )
    {
     cout<< "My objects address is "<<this;
     }
};
void main ( )
{
loca obj1, obj2;          //objects
obj1.fun1 ( );            // see location
obj2. fun1 ( );
}
```

ينشئ البرنامج في الدالة **main()** كائنين من النوع **loca**. ثم يطلب من كل كائن أن يعرض عنوانه باستعمال العضو الدالي **( ) fun1**. تعرض هذه الدالة قيمة المؤشر **this**. تكون قيمة مؤشره **this** هي قيمة عنوان الكائن الذي استدعى عضواً دالياً. يمكن للمؤشر **this** الوصول إلى بيانات الكائن الذي يشير إليه.

مثال:

```
//valuepointer.cpp
//this pointer referring to data
# include <iostream.h>
class what
   {
    private:
      int x;
    public:
```

```
        void fun ( )
        {
          this->x = 5;              //same as x = 5
          cout<<this;->x;           // same as cout <<x;
        }
    };
    void main ( )
    {
    what obj1;
    obj1.fun1 ( );
    }
```

يخرج البرنامج القيمة 5 يقوم العضو الدالي **fun()** بالوصول إلى المتغير x باستخدام التعبير:

this -> x;

إنّ المؤشر **this** يشير إلى الكائن **obj1**، وهو الكائن الذي استدعى العضو الدالي **fun()**.

17- المؤشرات const:

تطبيق الكلمة المحجوزة **const** على المؤشرات.

- مكانان لـ const:

يمكن تحديد مؤشر نفسه ثابت، و أن القيمة التي يشير إليها ثابتة، أو كليهما. نأخذ بمتغير عادي ونعرف ثلاثة مؤشرات إليه:

مثال:

```
int x;                    //variable
const int* p=&x;          //pointer to constant integer  x
```

```
++p;                    //ok
++(*p);                 // error , cant's modify const x
int* const q =&x;       //constant pointer to int
++q;                    // error, cant's modify const q
++(*q);                 // ok
const int * const r=&x;     //constant pointer to constant int
 ++r;                       // error, cant's modify const r
++(*r);                     // error, cant's modify const r
```

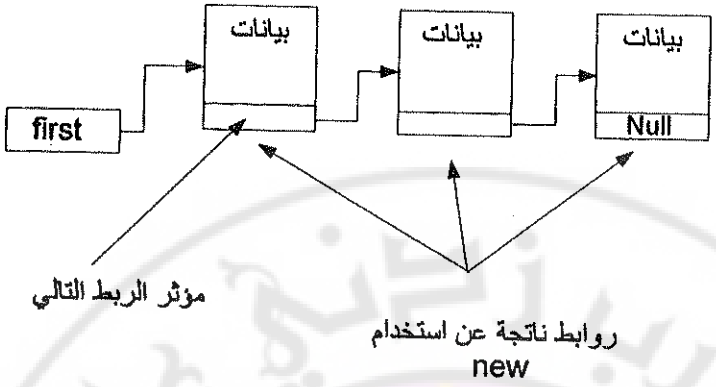
تمَّ وضع const قبل نوع البيانات (تعريف p) فالنتيجة هي مؤشر إلى متغير ثابت. وإذا تمَّ وضع const بعد نوع البيانات (تعريف q) فالنتيجة هي مؤشر ثابت إلى متغير. لا يمكن في الحالة الأولى تغيير قيمة المتغير الذي يشير إليه المؤشر؛ ولا يمكن في الحالة الثانية تغيير العنوان المخزن في المؤشر. إن استعمال const في كلَّ المكانين (تعريف r) يعني أنه لا تمكن من تغيير المتغير أو المؤشر الذي يشير إليه.

18- صف القوائم المرتبطة:

يمكن تخزين البيانات باستخدام القوائم المرتبطة. تتيح القوائم المرتبطة استخدام الحيز الذي نحتاجه من الذاكرة، حتى عندما لا تعرف تلك الكمية إلا بعد بدء تنفيذ البرنامج.

19- شبكة من المؤشرات:

يتمُّ الحصول إلى مساحة لكل عنصر بيانات عند الحاجة إلى ذلك بوساطة المعامل new ثم يتمُّ ربط كل عنصر بالعنصر التالي باستخدام مؤشر. لا داعي أن تكون العناصر الفردية مخزنة بشكل متجاور في الذاكرة، بل يمكن أن تكون مبعثرة في أي مكان يتسع لها.



الشكل (10-1) قائمة مرتبطة

إن القائمة المرتبطة بأكملها هي كائن دالة للصف **linklist**. ويتمّ تمثيل عناصر البيانات الفردية، أو الروابط ببنيات من النوع **link**. وتحتوي كل بنية مماثلة على عدد صحيح ــ يمثل عنصر البيانات الوحيد في الكائن ــ ومؤشر إلى الربط التالي. يخزن كائن **linklist** مؤشراً إلى الربط الموجود عند رأس القائمة؛ إنه عنصر بياناته الوحيد.

تحتوي الصف **linklist** على أعضاء دالية تتيح لمستخدم الصف إضافة ربط إلى القائمة وعرض البيانات الموجودة في كل الروابط. سيكون هناك مزيد من البيانات مخزنة في كل ربط في البرامج الفعلية.

مثال:

```
//linklist.cpp
# include <iostream.h>
struct link
        {
        int data;
        link* next;          //pointer to next link
        };
class  linklist
        {
```

```
private:
        link* first;              //pointer to first link
public:
    linklist ( )
            {
              first = null;
            }
     void additem (int );       //add data item (one link)
     void display ( );
};
void linklist :: additem (int d )
{
 link* newlink = new link;      //mak a new link
 newlink->data = d;             //giv it data
  newlink->next = first;        //it point to next link
 first = newlink;               // now first points to this
}
void linklist :: display ( )
{
 link* current = first;      //set pointer to first link
 while (current != NULL)             //quit on last link
  {
    cout<<endl<<current->data;  //print data
    current = current->next;       //mov to next link
    }
}
void main ( )
{
```

```
linklist  li ;              //mak linked list
li.additem (5);             //add three item to list
li.additem (38);
li.additem (14);
li.display ( );             //display entire list
}
```

تحتوي الصف **linklist** على عنصر بيانات واحد فقط: مؤشر إلى بداية القائمة، يدعى **first**. عند إنشاء القائمة في البدء، يسند الباني هذا المؤشر إلى **NULL** (خامل). إن الثابت **NULL** معرف في ملف الترويسة **mem.h** (المشمول بوساطة **#include** في ملف الترويسة **iostream.h**) وتساوي قيمته **0**. تلعب هذه القيمة دوراً تنبيهياً بأن المؤشر لا يخزن أي عنوان صالح. إن الربط الذي تساوي قيمة عضوه **next** هذه القيمة الخاملة (**NULL**) يعتبر أنه موجود عند نهاية القائمة.

20- إضافة عنصر إلى القائمة:

يضيف العضو الدالي **additem()** عنصراً إلى القائمة المرتبطة. ويتمّ إدراج ربط جديد في بداية القائمة. إن الخطوات المطلوبة في الدالة **additem()** لإدراج ربط جديد.

أولاً: يتمّ إنشاء بنية جديدة من النوع **link** بواسطة السطر

```
link* newlink = new link;
```

ينشئ المعامل **new** ذاكرة للبنية **link** الجديدة ويخزن المؤشر إليها في المتغير **newlink**.

نضبط الأعضاء في البنية المنشأة عند قيم مناسبة. البنية تشبه الصف في أنه عند التأشير إليها بوساطة مؤشر وليس بوساطة اسمها، يتمّ الوصول إلى أعضائها باستخدام معامل الوصول إلى أعضاء <-. يضبط السطران التاليان المتغير **data** عند القيمة الممررة كوسط إلى الدالة **additem()** والمؤشر **next** ليشير إلى العنوان الذي كان

مخزناً في **first**، وهو المؤشر الذي يشير إلى بداية اللائحة.

```
newlink->data = d;
newlink->next = first;
```

ثم يشير المتغير **first** إلى الربط الجديد:

```
first = newlink;
```

إن تأثير هذه العبارة هو فك الوصلة بين المؤشر **first** وبين الربط القديم وإدراج الربط الجديد بينهما، نقل الربط القديم إلى الموضع الثاني.

21- عرض محتويات القائمة:

بعد إنشاء القائمة يصبح من السهل الانتقال بين كل الأعضاء، وعرضها. وبعدئذ ننتقل من مؤشر **next** إلى مؤشر آخر إلى أن نجد مؤشر **next** قيمته **NULL**، مما يعني أننا وصلنا إلى نهاية القائمة. إن السطر التالي في الدالة **()display**.

```
cout<<endl<<current->data;
current = current->next;
  current != NULL;
```

يخرج قيمة البيانات:

```
5
38
14
```

إن القوائم المرتبطة هي أكثر طرق تخزين البيانات استخداماً بعد المصفوفات. القوائم المرتبطة لاتبذر بمساحة الذاكرة. أما سيئتها فهي أن إيجاد عنصر معين فيها يتطلب اتباع تسلسل الروابط بدءاً من رأسها وصولاً إلى العنصر المطلوب.


وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل الحادي عشر

الدفق والملفات

1- تعريف الدفق:

الدفق يعني سيل البيانات في حالة دخل/خرج.يمكن تمثيل دفق دخل/خرج بكائن دالة لصف معينة.

2- الصف ios:

إن الصف **ios** هي سلف كل صفوف الدفق الأخرى وتحتوي على أغلبية الميزات التي نحتاج إليها لاستخدام الدفق في لغة البرمجة C++. إن أهم ثلاث ميزات هي أعلام التنسيق. وحالات الأخطاء، ونمط عمل الملفات.

3- أعلام التنسيق:

أعلام التنسيق هي مجموعة من تعريفات تعدادية **(enum)** في الصف **ios** تعمل كمفاتيح قلابة تنشيط / تعطيل **(on/of)** تحدد خيارات لعدة نواح في عمل وتنسيق الدخل والخرج.

المعلم	معناه
skipws	تخطي (تجاهل) المسافات الفارغة الموجودة في الدخل
left	محاذاة الخرج إلى اليسار [12.34]
right	محاذاة الخرج إلى اليمين [12.34]

internal	استعمال المسافة الموجودة بين مؤشر العلامة أو القاعدة وبين الرقم [12.34+]
dec	تحويل إلى عشري
oct	تحويل إلى ثماني
hex	تحويل إلى ست عشري
showbase	استعمال مؤشر القاعدة في الخرج (0 للثماني و0x للست عشري)
showpoint	إظهار النقطة العشرية في الخرج
uppercase	استعمال X وE وأحرف الست عشري ABCDEF الكبيرة (الخيار الافتراضي هي استعمال الأحرف الصغيرة)
showpos	عرض "+" قبل الأعداد الصحيحة الموجبة
scientific	استعمال التنسيق الأسي في الخرج العائم [9.1234E2]
fixed	استعمال التنسيق الثابت في الخرج العائم [912.34]
unitbuf	مسح كل الدفوق بعد الإدراج
stdio	مسح stdout وstderror بعد الخرج

الجدول (11-1) أعلام تنسيق ios

المناور	هدفه
ws	تنشيط ميزة تخطي المسافات البيضاء الموجودة في الدخل
dec	التحويل إلى عشري
oct	التحويل إلى ثماني
hex	التحويل إلى ستعشري
endl	إدراج سطر جديد ومسح دفق الخرج
ends	إدراج حرف خالد لإنهاء سلسلة خرج
flush	مسح دفق الخرج
lock	قفل مقبض (handle) الملف
unlock	إلغاء قفل مقبض الملف

الجدول (11-2) مناورات ios التي ليس لها وسطاء

4- الدوال التحكم مسبقة التعريف:

دوال التحكم هي تعليمات تنسيق مدرجة في الدفق مباشرة.

يعرض الجدول (11-3) الدوال التي تأخذ وسطاء. تحتاج إلى ملف الترويسة iomanip.h لكي تستعمل هذه الدوال. إن الدوال مسبقة التعريف التي تأخذ وسطاء تؤثر فقط في العنصر التالي في الدفق.

الدالات مسبقة التعريف	الوسيطة	هدفه
setw()	عرض الحقل (int)	ضبط عرض الحقل المطلوب خرجه
setfill()	حرف الحشو (int)	ضبط حرف الحشو في الخرج (الحرف الافتراضي هو المسافة)
setprecision()	الدقة (int)	ضبط الدقة (كمية الأعداد المعروضة)
setiosflags()	أعلام التنسيق (long)	ضبط الأعلام المحددة
resetiosflags()	أعلام التنسيق (long)	ضبط الأعلام المحددة

الجدول (11-3) دوال ios التي تأخذ وسطاء:

5- الدوال:

تحتوي الصف ios على عدد من الدوال التي يمكن استخدامها لضبط اعلام التنسيق وتنفيذ مهام أخرى.

الدالة	هدفها
Ch=fill();	إعادة حرف الحشو (تحشو الجزء غير المستعمل من الحقل، الفراغ هو الافتراضي)
fill(ch);	ضبط حرف الحشو

p=precision();	الحصول على الدقة (كمية الأعداد المعروضة للأرقام العائمة)
precision(p);	ضبط الدقة
w=width();	الحصول على عرض الحقل الحالي (بالأحرف)
width(w);	ضبط عرض الحقل الحالي
Setf(flags);	إلغاء ضبط أعلام التنسيق المحددة
setf(flags,field);	مسح الحقل أولاً، ثم ضبط الأعلام

الجدول (11-4) دوال ios

الوسيط الأولى: الأعلام للضبط	الوسيط الثانية: الحقل للمسح
dec, oct, hex	basefield
left, right, internal	adjustfield
scientific, fixed	floatfield

الجدول (11-5) إصدار ثنائي الوسطاء من الدالة ()setf

يمكن استخدام عدة دوال لمعالجة ios مباشرة.

إن إصداراً ثنائي الوسطاء من الدالة ( )setf يستخدم الوسيط الثانية لإعادة ضبط كل أعلام نوع أو حقل معين. يتمّ عندها ضبط المعلم المحدد في الوسيط الأولى. يسهل هذا عملية إعادة ضبط الأعلام ذات الصلة بالموضوع الحالي قبل ضبط معلم جديد.

6- الصف istream:

تنفذ الصف istream، المشتقة من الصف ios، نشاطات خاصة بالدخل، أو نشاطات إضافية.

الدالة	هدفها
>>	استخراج منسق لكل الأنواع الأساسية (والمحملة بشكل زائد)
get(ch)	استخراج حرف واحد إلى ch
get(str)	استخراج أحرف إلى الصفوفة str، وصولاً إلى '\0'
get(str, MAX)	استخراج حتى MAX أحرف إلى الصفوفة
get(str, DELIM)	استخراج أحرف إلى الصفوفة str وصولاً إلى الحدود المذكورة ('\n' عادة). ترك الحدود في الدفق
get(str, Max, DELIM)	استخراج أحرف إلى الصفوفة str وصولاً إلى MAX أحرف أو إلى الحرف DELIM. ترك الحدود هنا في الدفق
getline(str, MAX, DELIM)	استخراج أحرف إلى الصفوفة str وصولاً إلى MAX أحرف أو إلى الحرف DELIM. استخراج الحدود.
putback(ch)	إعادة إدراج الحرف الأخير المقروء في دفق الدخل
ignore(MAX,	استخراج وتجاهل ما يصل إلى MAX أحرف، وصولاً

DELIM)	إلى (ومع) حرف الحدود المذكور ('\n' عادة)
peek(ch)	قراءة حرف واحد، وتركه في الدفق
count=gcount()	إعادة عدد الأحرف التي قرأها استدعاء (يسبق مباشرة) الدالة get() أو getline() أو read()
read(str, Max)	للملفات. استخراج ما يصل إلى MAX أحرف إلى str وصولاً إلى EOF (نهاية الملف)
seekg(position)	ضبط المسافة (بالبايتات) لمؤشر الملف من بداية الملف
seekg(position, seek_dir)	ضبط المسافة (بالبايتات) لمؤشر الملف من المكان المحدد في الملف: seek_dir يمكن أن تكون ios::beg أو ios::cur أو ios::end
position=tellg(pos)	إعادة موقع الملف (بالبايتات) من بداية الملف.

الجدول (11–6) دوال istream:

الدالة	هدفها
<<	إدراج منسق لكل الأنواع الأساسية (والمحملة بشكل زائد)
put(ch)	إدراج الحرف ch في الدفق
flush()	مسح محتويات الدارئ وإدراج سطر جديد
write(str, SIZE)	إدراج SIZE الحرف من الصفوفة str في الملف

seekp(position)	ضبط المسافة بالبايتات لمؤشر الملف من بداية الملف
seekg(position, seek_dir)	ضبط المسافة بالبايتات لمؤشر الملف من المكان المحدد فيه. Seek_dir يمكن أن تكون ios::beg أو ios::cur أو ios::end
position=tellp()	إعادة موقع مؤشر الملف، بالبايتات.

الجدول (11-7) دوال ostream

لقد رأينا بعضاً من هذه الدوال، كـ get()، من قبل. معظمها يعمل على الكائن cin الذي يمثل لوحة المفاتيح وملفات الأقراص أيضاً. لكن الدالات الأربع الأخيرة تتعامل مع ملفات الأقراص خصيصاً.

7- الصف ostream:

يعالج الصف ostream نشاطات الخرج أو الإدراج.

- دخل / خرج ملفات الأقراص بوساطة الدفق:

تحتاج الملفات مجموعة من الصفوف مختلفة عن تلك التي تحتاجها الملفات بوساطة لوحة المفاتيح والشاشة. إنها ifstream للدخل، وfstream للدخل والخرج، وofstream للخرج. ويمكن ربط الكائنات التابعة لهذه الصفوف بملفات الأقراص ويمكن استخدام أعضائها الدالية للقراءة من الملفات والكتابة عليها.

8- كتابة البيانات:

يكتب البرنامج التالي محرفاً وعدداً صحيحاً ورقماً مزدوجاً وسلسلتين على ملف قرص يدعى fdata.txt. لا يوجد خرج على الشاشة.

مثال:

```
//ftext.cpp
//write formatted output to a file
# <include <fstream.h>
void main ( )
      {
      char ch = 'x';
       int I  = 23;
       double d = 3.09;
      char str1 [] = " My program";
      char str2 [] = " I am student";
      ofstream outfile ("fdata.text"); // create ofstream object
      outfile<<ch              //write data
            <<i<<' '
            <<d<<' '
            <<str1<<' '
            <<str2;
       }
```

يعرف البرنامج هنا كائناً يدعى **outfile** ليكون عضواً في الصف **ofstream**، ويسند لنفسه اسم الملف **fdata.txt**. إذا لم يكن الملف موجوداً، يتمّ إنشاؤه. وإذا كان موجوداً يتمّ حذف البيانات القديمة وتحل البيانات الجديدة محل البيانات القديمة. يتمّ استخدام معامل الإدراج (<<) لإخراج المتغيرات ذات الأنواع الأساسية إلى الملف. يعمل هذا الأمر لأن معامل الإدراج محمل بشكل زائد في الصف **ostream**، الذي تمّ اشتقاق الصف **ofstream** منه.

عندما ينتهي البرنامج، يصبح الكائن **outfile** خارج المدى، يؤدي هذا إلى استدعاء مهدّمه الذي يغلق الملف.

نفصل الأرقام (كـ 77 و6.02) بوساطة أحرف غير رقمية. إنها الطريقة الوحيدة التي يعرف فيها معامل الإخراج، عند قراءة البيانات من الملف، مكان انتهاء الرقم وبداية رقم آخر. ونفصل سلاسل المحارف عن بعضها بعضاً بمسافة بيضاء للسبب نفسه.

9- قراءة البيانات:

يستطيع أي برنامج قراءة الملف المولد في البرنامج ftext باستخدام كائن دالة الصف ifstream مسند له اسم الملف. يتمّ فتح الملف تلقائياً عند إنشاء الكائن. يستطيع البرنامج عندها القراءة منه باستعمال معامل الاخراج (>>).

مثال:

```
//ftext.cpp
//read formatted output to a file
# <include <fstream.h>
const int max = 40;
void main ( )
      {
       char ch = 'x';
       int I  = 23;
       double d = 3.09;
       char str1 [max] ;
       char str2 [max] ;
       ifstream infile ("fdata.text"); // create ifstream object
       infile>>ch>>endl          //read data
             <<I<<endl
               <<d<<endl
                 <<str1<<endl
```

```
<<str2;          }
```

يتصرف كائن ifstream الذي اسمه infile، مثلما فعل cin في البرامج السابقة.

10- سلاسل محارف مع فراغات:

يمكن إدخال سلاسل المحارف وإخراجها بوساطة استخدام الملفات. بعض سلاسل المحارف التي تحتوي على فراغات فيها.

مثال:

```
//fstr.cpp
//file output
# <include <fstream.h>
void main ()
     {
      ofstream outfile ("fdata.text"); // create file for output
     outfile<<"I am student \n";
     outfile<<"My first program\n";
      }
```

عند تنفيذ البرنامج، تتمّ كتابة سطري النص في ملف. وينتهي كل سطر بحرف سطر جديد ('\n').

لإخراج السلاسل المحارف من الملف، ينشئ البرنامج كائناً ifstream ويقرأ منه سطراً واحداً كل مرة باستخدام الدالة getline()، وهي عضو في الصنف istream. تقرأ هذه الدالة المحارف، بما في ذلك المسافات البيضاء إلى أن تجد المحرف '\n' وتضع سلسلة المحارف الناتجة في دالة كوسيط لها.

مثال:

```
//inline.cpp
//file input
```

```
# <include <fstream.h>
void main ( )
    {
     const int max = 40;
     char buffer [max];
     ifstream infile ("fdata.text"); // create  file for input
    while (!inlile.eof( ) )
    {
    infile.getline (buffer , max); //read a line of text
    cout<<buffer<<endl;
    }
    }
```

إن الخرج من البرنامج **inline** إلى الشاشة هو البيانات نفسها التي كتبها البرنامج **fstr** في الملف **fdata.txt**: أسطر النص. يتابع البرنامج القراءة سلسلة تلو الأخرى إلى أن يجد علامة نهاية الملف **EOF**. لا يستخدم هذا البرنامج لقراءة ملفات نصية عشوائية، فهو يتطلب أن تكون الأسطر منتهية بالحرف '\n'، وإذا وجدت ملفاً لا ينطبق عليه هذا الشرط، يتوقف البرنامج عن العمل.

11- الدالة open():

لقد أنشأنا ملفات باستخدام الدالة **open()**، وهي عضو في الصف **fstream**. هذا الأسلوب مفيد في الحالات التي قد يفشل الفتح فيها. يمكنك إنشاء كائن دفق ثم محاولة فتحه باستمرار من دون الاضطرار إلى إنشاء كائن دفق كل مرة.

النمط	النتيجة
in	فتح للقراءة (الافتراضي للصف **ifstream**)
out	فتح للكتابة (الافتراضي للصف **ofstream**)
ate	بدء القراءة أو الكتابة عند نهاية الملف **(AT END)**
app	بدء الكتابة عند نهاية الملف **(APPend)**
trunc	بتر الملف ليصبح طوله صفراً إذا كان موجوداً **(TRUNCate)**
nocreate	خطأ عند الفتح إذا لم يكن الملف موجوداً مسبقاً
noreplace	خطأ عند الفتح للخرج إذا كان الملف موجوداً مسبقاً، إلا إذا كان بت النمط **ate** أو **app** مضبوطاً
binary	فتح الملف في النمط الثنائي (ليس نصاً)

الجدول (11-8) الأنماط للدالة ()open:

12- خرج الطابعة:

تكون الطابعة في معظم الكمبيوترات موصولة بالمنفذ المتوازي الأول، لذا فإن اسم ملف الطابعة هو **prn** أو **lpt1**.

الاسم	الجهاز
con	الكونسول (لوحة المفاتيح والشاشة)
aux أو com1	المنفذ التسلسلي الأول
com2	المنفذ التسلسلي الثاني
prn أو lpt1	المنفذ المتوازي الأولى
lpt2	المنفذ المتوازي الثاني
lpt3	المنفذ المتوازي الثالث
nul	الجهاز الزائف (غير موجود)

الجدول (11-9) أسماء ملفات العتاد المعرفة مسبقاً

مثال:

```
//filprint.cpp
//simple printer
# <include <fstream.h>
void main ( )
      {
       char * s1 = "\n I am student";
       int n = 1234;
      ofstream outfile; // mak fil
      outfile.open("PRN");       // open it for printer
      outfile<<s1<<n1<<endl;   // send data to printer
     outfile <<'\x0c';          }
```

يمكن طباعة محتويات ملف قرص، محدد في سطر الأوامر، على الطابعة. إنه يستعمل أسلوب المحرف ــ تلو ــ المحرف في إرسال البيانات إلى الطابعة.

مثال:

```
//filprint.cpp
//simple printer
# include <process.h>
# include <fstream.h>
void main ( int arg, char* argv [])
     {
     if (arg !=2)
      {
       cerr<<"\n format file";
       exit(-1);
       }
      char  ch;
      ifstream infile;                    //creat file for input
      infile.open(argv[1]);               //open file
       if (!infile)                       //check for error
        {
          cerr<<"\n can't open";<<argv[1];
          exit(-1);              }
      ofstream outfile;          // mak fil
      outfile.open("PRN");       // open it for printer
      while(infile.get(ch) !=0)        //read a character
      outfile.put(ch);       // write character to printer        }
```

يمكن استخدام هذا البرنامج لطباعة أي ملف نصي.


وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل الثاني عشر

الرسم بلغة البرمجة c++

1- مقدمة:

الرسم بستخدام الحاسوب هو طريقة إدخال وإخراج الأشكال إلى الحاسوب، ومعالجة الأشكال وعرضها باستخدام الحاسوب. حيث تظهر باستمرار طرق حديثة لحل مشاكل جديدة ويتمّ تطوير التقنيات بغية التوصل إلى أفضل أنظمة للرسوم سهلة الاستخدام حتى من قبل غير المتخصصين في مجال العمل على الحاسوب.

والبحث في هذا المجال مستمر لتحسين خوارزميات الرسم لجعل عملية عرض الرسوم أسرع. فالرسم على الحاسوب يمثل أحدث التطورات في تحسين كفاءة الاتصالات بين البشر والحاسوب.

2- دوال إعداد وحدة الرسم Setting Graphics unit Routine

هناك بعض الدوال التي تعتبر بمنزلة أدوات حيث إنها لا تقوم بالرسومات ولكنها تعد وحدة الرسم وتهيئتها لكي نتمكن من تنفيذ برامج الرسوم عليها. إن كل هذه الدوال جاهزة تقريباً وتخص نظام الرسوم المستخدم. فيما يلي قائمة بهذه الدوال مع توضيح الغرض وكيفية استخدامها وبيان نوع هذه البارامترات وعددها.

- الدالة **cleardevice ( ):**

صيغتها العامة كما يلي:

void far cleardevice(void);

تستخدم هذه الدالة لمسح شاشة الرسم البياني وذلك بملئها بلون خلفية الشاشة قيد الاستخدام، ويعيد المؤشر إلى الموقع (0,0)، الزاوية العليا اليسرى للشاشة، في الإحداثيات المطلقة.

مثال:

البرنامج التالي يبين كيفية استدعاء الدالة cleardevice ( )، حيث تظهر رسالة في وسط الشاشة.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
        // request auto detection
        int gdriver = DETECT,gmode,errorcode;
        int midx,midy;

        // initialize graphics and local variables
        initgraph(&gdriver,&gmode,"");
        // read result of initialization
        errorcode = graphresult();
        if( errorcode != grOk) // an error occurred
        {
        printf("Graphecs error :%s\n",grapherrormsg(errorcode));
                printf("Press any key to halt");
                getch();
                exit(1); // terminiate with error code
        }
        midx = getmaxx()/2;
        midy = getmaxy()/2;
        setcolor(getmaxcolor());
        // for centring screen messages
        settextjustify(CENTER_TEXT,CENTER_TEXT);
        // output a meesage to the screen
        outtextxy(midx,midy,"Press any key to clear the screen:");
        // wait for a key
        getch();
        // clear the screen
        cleardevice();
```

```
    // output another message
    outtextxy(midx,midy,"Press any key to quit");
    // clean up
    getch();
    closegraph();
    return 0;
}
```

- الدالة **detectgraph ( ):**

صيغتها العامة كما يلي:

void far detectgraph(int far * graphdriver, int far * graphmode);

وهو يستكشف سواق الرسم (GD) قيد الاستخدام ونمط الرسم (GM) المناسب الذي يمكن أن نستخدمها من قبل موافق Adapter العرض المستخدم. القيم الناتجة تعود في البارمترات GraphDriver و Graphmode. يمكن استخدام الدوال DetectGraph لفحص قيم (GD) و (GM) المستكشفة بصورة أتوماتيكية، فيما إذا كانت مناسبة أم لا، وإذا كانت مناسبة فيمكن تغييرها قبل أن تمرر إلى الدالة ( ) initgraph.

- الدالة **initgraph ( ):**

صيغتها العامة كما يلي:

void far initgraph (int far * graphdriver, int far * graphmode,char far * pathtodriver);

تقوم هذا الدالة بالإعداد الأولي لنظام الرسوم وللمكونات المادية لنمط الرسوم أتوماتيكاً. البارمتر Driverpath هو المسار الذي تستخدمه الدالة للبحث عن الملفات المناسبة والتي يكون امتداداها BGI. ولكي تسمح لنواة الرسم أن تختار أفضل نمط رسم يجب إسناد القيمة الثابتة في وحدة الرسم Detect للمتغير GraphDriver كما في المثال التالي:

مثال:

```
GraphDrive = DETECT;
initgraph(GraphDriver,Graphmode,'C:\TP\BGI');
```

تقوم هذا الدالة بالإعداد الأولي Initialize لنظام الرسوم وللمكونات المادية لنمط الرسوم أوتوماتيكياً. البارامتر DriverPath هو المسار الذي تستخدمه الدالة للبحث عن الملفات المناسبة والتي يكون امتدادها BGI، ولكي تسمح لنواة الرسم أن تختار أحسن نمط رسوم متوفر يجب إسناد القيمة الثابتة في وحدة الرسم Detect للمتغير GraphDriver:

إذا كانت قيمة GraphDriver صفراً في InitGraph، يتمّ استدعاء الدالة DetectGraph أوتوماتيكياً للحصول على أفضل سواقة رسم ونمط رسم بأعلى قوة تحليل. القيم المحصلة أوتوماتيكياً تعاد إلى المتغيرين GraphDriver و Graphmode. ولإلغاء هذه القيم الناتجة أوتوماتيكياً يتمّ ذلك بإسناد القيم المرغوب بها للسواقة ونمط الرسوم للمتغيرين GraphDriver و Graphmode. أما القيمة "C:\TP\BGI" فهي تمثل المسار الذي سيبحث فيه الدالة عن الملفات ذات الامتداد BGI.

البرنامج التالي يستدعي الدالة **initgraph ( )** وقد أسندت القيمة **DETECT** إلى gdriver، أما المسار فقد ترك فارغاً أي أنه يتمّ اختياره أوتوماتيكياً.

مثال:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
      // request auto detection
      int gdriver = DETECT,gmode,errorcode;
      // initialize graphics and local variables
      initgraph(&gdriver,&gmode,"");
      // read result of initialization
```

```
errorcode = graphresult();
if( errorcode != grOk) // an error occurred
{
printf("Graphecs error :%s\n",grapherrormsg(errorcode));
        printf("Press any key to halt");
        getch();
        exit(1); // terminiate with error code
}
//draw a line
line (0,0,getmaxx(),getmaxy());
// clean up
getch();
closegraph();
return 0;
}
```

☜ الدالة ( ) grapherrormsg:

صيغتها العامة كما يلي:

```
char * far grapherrormsg(int errorcode);
```

تعيد هذه الدالة رسالة خطأ تتناسب مع حالة الخطأ المبينة في البارمتر Codc على شكل قيمة صحيحة. القيم التي يمكن تمريرها إلى ( ) grapherrormsg محددة بالقيم من 0 إلى 15 الثوابت التالية يمكن أن تمرر إلى ( ) grapherrormsg.

const	
grOK	0
grNoInitGraph	-1
grNotDetect	-2
grFileNotFound	-3
grInvaliadDriver	-4
GrNoLoadMem	-5

grnoScanMem	-6
grNoFloodMem	-7
grFontNotFound	-8
grNoFunctionMem	-9
grInvalidMode	-10
grEror	-11
grIOError	-12
grInvalidFont	-13
grInvalidFontNum	-14
GrInvalidDeviceNum	-15

الجدول (12-1)

تعيد هذه الدالة **grapherrormsg ( )** مؤشراً إلى بداية السلسلة النصية التي تحوي رسالة الخطأ.

مثال:

البرنامج التالي يستدعي الدالة **grapherrormsg ( )** :

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#includc <conio.h>
int main(void)
{
        // request auto detection
        int gdriver = DETECT,gmode,errorcode;
```

```
// initialize graphics and local variables
initgraph(&gdriver,&gmode,"");
// read result of initialization
errorcode = graphresult();
if( errorcode != grOk) // an error occurred
{
printf("Graphecs error :%s\n",grapherrormsg(errorcode));
        printf("Press any key to halt");
        getch();
        exit(1); // terminiate with error code
}
//draw a line
line (0,0,getmaxx(),getmaxy());
// clean up
getch();
closegraph();
return 0;
}
```

☜ الدالة graphresult ( ):

وصيغتها العامة كما يلي:

```
int far graphresult(void);
```

هذه الدالة للتدقيق فيما لو حصل أي خطأ في الرسوم في وقت تنفيذ البرنامج، وتكون القيمة العددية التي ستعيدها هذه الدالة محددة بالقيم من 0 إلى 15.

ويمكن ترجمة هذه القيمة العددية إلى رسالة خطأ نصية تصف نوع الخطأ وذلك باستخدام الدالة grapherrormsg ( ). يتمّ إسناد القيمة صفر إلى الدالة graphresult ( ) حالما يتمّ استدعاؤها.

يبين الجدول (12-1) بعض قيم graphresult ( ) التي تعيدها دوال الرسم عند حصول أي خطأ أثناء تنفيذ البرنامج مع بيان نوع الخطأ.

مثال:

البرنامج التالي يستدعي الدالة ( ) graphresult:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
        // request auto detection
        int gdriver = DETECT,gmode,errorcode;

        // initialize graphics and local variables
        initgraph(&gdriver,&gmode,"");
        // read result of initialization
        errorcode = graphresult();
        if( errorcode != grOk) // an error occurred
        {
        printf("Graphecs error :%s\n",grapherrormsg(errorcode));
                printf("Press any key to halt");
                getch();
                exit(1); // terminiate with an error code
        }
        //draw a line
        line (0,0,getmaxx(),getmaxy());
        // clean up
        getch();
        closegraph();
        return 0;
}
```

الدالة setgraphmode ( ) ⇐

صيغتها العامة كما يلي:

```
vod far setgraphmode (intmode);
```

تعمل الدالة **setgraphmode ( )** على إعداد نمط الرسوم قيد الاستخدام إلى النمط المخصص في البارامتر Mode، ثم ينظف شاشة العرض. القيمة Mode المستخدمة في هذا الدالة **initgraph ( )**. يمكن الاستفادة من هذا الدالة لإلغاء قيمة نمط الرسوم المعدة من قبل الدالة **initgraph ( )**.

مثال:

البرنامج التالي يوضح كيفية استدعاء الدالة **setgraphmode ( )** .

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
        // request auto detection
        int gdriver = DETECT,gmode,errorcode;
        int x,y;
        // initialize graphics and local variables
        initgraph(&gdriver,&gmode,"");
        // read result of initialization
        errorcode = graphresult();
                if( errorcode != grOk) // an error occurred
        {
        printf("Graphecs error :%s\n",grapherrormsg(errorcode));
                printf("Press any key to halt");
                getch();
                exit(1); // terminiate with an error code
        }

        x= getmaxx()/2;
        y=getmaxy()/2;
        // output a message
        settextjustify(CENTER_TEXT,CENTER_TEXT);
        outtextxy(x,y,"Press any key to exit graphics:");
        // restore system to text mode
        restorecrtmode();
        printf("We are now in text mode.\n");
```

```
    printf("Press any key to return to graphics mode");
    getch();
    //return to graphic mode
    setgraphmode(getgraphmode());
    // output a message
    settextjustify(CENTER_TEXT,CENTER_TEXT);
    outtextxy(x,y,"We are Back in graphic mode:");
    outtextxy(x,y+textheight("W"),"Press any key to halt");
    // clean up
    getch();
    closegraph();
    return 0;
}
```

وعند التنفيذ كان الخرج على الشكل:

press any key to exit graphics

وبعد الضغط على أي مفتاح، تظهر الرسائل التالية:

We are now in text mode

Press any key to return to graphics mode

وعند الضغط على أي مفتاح مرة أخرى، تظهر الرسالة التالية:

We are back graphics mode

press any key to halt

الدالة ( ) getgraphmode:

صيغتها العامة كما يلي:

```
int far getgraphmode(void);
```

تعيد القيمة العددية لنمط الرسوم قيد الاستخدام في الحاسوب والتي تمّ إعدادها باستدعاء سابق لأحد الإجرائيتين ( ) setgraphmode أو ( ) initgraph.

مثال:

البرنامج التالي يستدعي الدالة ( ) setgraphmode والدالة ( ) getgraphmode.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
        // request auto detection
        int gdriver = DETECT,gmode,errorcode;
        int midx,midy,mode;
        char numname[80],modename[80];
        // initialize graphics and local variables
        initgraph(&gdriver,&gmode,"");
        // read result of initialization
        errorcode = graphresult();
        if( errorcode != grOk) // an error occurred
        {
        printf("Graphccs crror :%s\n",grapherrormsg(errorcode));
                printf("Press any key to halt");
                getch();
                exit(1); // terminiate with an error code
        }
        midx = getmaxx()/2;
        midy = getmaxy()/2;
        // get mode numberand name string
        mode = getgraphmode();
        sprintf(numname,"%d is the current mode number: ",mode);
          sprintf(modename,"%d is the current graphic mode:
",getmodename(mode));
        // display the information
    settextjustify(CENTER_TEXT,CENTER_TEXT);
        outtextxy(midx,midy,numname);
        outtextxy(midx,midy+2*textheight("W"),modename);
        // restore system to text mode
        outtext("<Enter> to leave graphics :");
        getch();
        restorecrtmode();
        printf("now in text mode.\n");
        //return to graphic mode
        printf("<Enter> to enter graphics mode");
        getch();
```

```
    setgraphmode(getgraphmode());
    // output a message
    settextjustify(CENTER_TEXT,CENTER_TEXT);
    outtextxy(0,0," Back in graphic mode:");
    outtextxy(0,0+textheight("H"),"<Enter to quit>");
    // clean up
    getch();
    closegraph();
    return 0;
}
```

عند تنفيذ البرنامج كان الخرج على الشكل التالي:

2 is the current mode number

640x480 VGA is the current graphics mode

press any key to exit graphics

وبعد الضغط على أي مفتاح، تظهر الرسائل التالية:

We are now in text mode

Press any key to return to graphics mode

وعند الضغط على أية مفتاح مرة أخرى، تظهر الرسالة التالية:

We are back graphics mode

press any key to halt

- الدالة **getdrivername ( )**:

صيغتها العامة كالتالي:

```
char *far getdrivername(void);
```

تعيد الدالة **getdrivername ( )** نصاً يصف اسم سواقة الرسم graphDriver. يمكن استدعاء هذه الدالة بعد استدعاء الدالة **initgraph ( )**. و تعيد مؤشراً إلى سلسلة تحتوي على اسم سواقة الرسم المستخدمة.

مثال:

البرنامج التالي يستدعي الدالة ( ) getdrivername .

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
        // request auto detection
        int gdriver = DETECT,gmode,errorcode;
        int midx,midy;
        // stors the device driver name
        char *drivername;
        // initialize graphics and local variables
        initgraph(&gdriver,&gmode,"");
        // read result of initialization
        errorcode = graphresult();
        if( errorcode != grOk) // an error occurred
        {
      printf("Graphecs error :%s\n",grapherrormsg(errorcode));
                printf("Press any key to halt");
                getch();
                // terminiate with an error code
                exit(1);
        }
        setcolor(getmaxcolor());
        // get name of the device driver in use
        drivername = getdrivername();
        // for centring text on the screen
        settextjustify(CENTER_TEXT,CENTER_TEXT);
        // output name of the device driver in use
        midx = getmaxx()/2;
        midy = getmaxy()/2;
        outtextxy(midx,midy,drivername);
        // clean up
        getch();
        closegraph();
        return 0;
```

يعيد البرنامج اسم سواقة الرسم المستخدمة في وسط الشاشة كما يلي:

EGAVGA

☜ الدالة **getmodename ( ):**

وصيغتها العامة:

```
char * far getmodename(int mode_number);
```

تعيد الدالة **getmodename ( )** نصاً يصف نمط الرسوم البينية قيد الاستخدام والمعطى رقمه في البارامتر modeNumber، أي إنها تقبل قيمة صحيحة وتعيد قيمة على شكل سلسة والتي هي مؤشر (pointer) إلى أول عنصر في السلسلة.

☜ الدالة **getmoderange ( ):**

صيغتها العامة كما يلي:

```
void far getmoderange(int graphdriver, int far *lomode,
int far *himode);
```

تعيد الدالة **getmoderange ( )** أعلى قيمة Himode وأقل قيمة Lowmode (المدى) لأطوار التحليل لسواقة الرسوم البيانية المخصص في المتغير GraphDriver. فمثلاً السواقة EGA64 ستعطي القيمة 0 للبارامتر Lowmode والقيمة 1 للبارامتر Himode عند استخدام الدالة **getmoderange ( )**.

مثال:

يعيد البرنامج التالي مدى سواقات الرسوم في الجهاز قيد الاستخدام.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
      // request auto detection
```

```
        int gdriver = DETECT,gmode,errorcode;
        int midx,midy;
        int low,high;
        char mrange[80];
        // initialize graphics and local variables
        initgraph(&gdriver,&gmode,"");
        // read result of initialization
        errorcode = graphresult();
        if( errorcode != grOk) // an error occurred
        {
        printf("Graphecs error :%s\n",grapherrormsg(errorcode));
                printf("Press any key to halt");
                getch();
                exit(1); // terminiate with an error code
        }
        midx = getmaxx()/2;
        midy = getmaxy()/2;
        // get the mode range for this driver
        getmoderange(gdriver,&low,&high);
        // convert mode range into strings
sprintf(mrange,"this driver suport modes %d..%d ",low,high);
        // display the information
        settextjustify(CENTER_TEXT,CENTER_TEXT);
        outtextxy(midx,midy,mrange);
        // clean up
        getch();
        closegraph();
        return 0;
}
```

وعند التنفيذ كان خرج البرنامج على الشكل التالي:

```
this driver supports modes 0..2
```

- الدالة graphdefaults( ):

صيغتها العامة كما يلي:

```
void far graphdefaults(void);
```

يحدد إعدادات الرسم البياني إلى الحالة التي تلي استدعاء الدالة InitGraph

مباشرة.

- الدالة **restorecrtmode( ):**

صيغتها العامة كما يلي:

```
void far restorecrtmode (void);
```

تعيد الدالة تخزين نمط العرض المستخدم سابقاً قبل الإعدادات الابتدائية لنمط الرسوم. هذا الدالة يفيد في العودة إلى نمط النص بعد أن يتمّ استخدام نمط الرسوم على فرض أن نمط النص قد تمّ استخدامه مسبقاً وقبل نمط الرسوم.

- الدالة **setvisualpage( ):**

صيغتها العامة بلغة C++ كما يلي:

```
void far setvisualpage(int page);
```

تختار الدالة **restorecrtmode( )** صفحة الرسم البياني المحدد رقمها في البارامتر Page والتي هي من النوع الصحيح لعرضها.

- الدالة **setactivepage( ):**

صيغتها العامة كما يلي:

```
void far setactivepage(int page);
```

تحدد الدالة **setactivepage( )** صفحة الرسم الفعالة والتي سيظهر عليها الخرج بالصفحة المخصصة في البارامتر Page. أي خرج رسومي ستوجه إلى هذه الصفحة.

مثال:

البرنامج التالي يستدعي كلٌ من **setactivepage( )** و **setvisualpage( ):**

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
```

```
{
    // request auto detection
    int gdriver = DETECT,gmode,errorcode;
    int x,y,ht;
    // initialize graphics and local variables
    initgraph(&gdriver,&gmode,"");
    // read result of initialization
    errorcode = graphresult();
    if( errorcode != grOk) // an error occurred
    {
    printf("Graphecs error :%s\n",grapherrormsg(errorcode));
            printf("Press any key to halt");
            getch();
            exit(1); // terminiate with an error code
    }
    x = getmaxx()/2;
    y = getmaxy()/2;
    ht = textheight("W");
    // select the screen page for drawing
    setactivepage(1);
    // draw a line on page # 1
    line(0,0,getmaxx(),getmaxy());
    // output message on page # 1
    settextjustify(CENTER_TEXT,CENTER_TEXT);
    outtextxy(x,y,"this is page # 1");
    outtextxy(x,y + ht, "Press any key to halt");
    getch();
    // select draing to page 0
    setactivepage(0);

    // output message on page # 0
    settextjustify(CENTER_TEXT,CENTER_TEXT);
    outtextxy(x,y,"this is page # 0");
    outtextxy(x,y + ht, "Press any key to view page #1:");
    getch();
    // select page 1 as visible page
    setvisualpage(1);
    // clean up
    getch();
    closegraph();
```

```
return 0;
```

- الدالة setgraphbuffsize( ):

صيغتها العامة كما يلي:

```
void far setgraphbuffsize (unsigned BuffSize);
```

يمكن للدالة setgraphbuffsize( ) من تحديد حجم خزان الشكل.

مثال:

البرنامج التالي يستدعي الدالة setgraphbuffsize( ).

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#define BUFSIZE 1000 /*internal graphics
buffer size*/
int main(void)
{
      // request auto detection
      int gdriver = DETECT,gmode,errorcode;
      int x,y,oldsize;
      char msg[80];
      /* set size of internal graphics buffer
      before making a call to initgraph. */
  oldsize = setgraphbufsize(BUFSIZE);
      // initialize graphics and local variables
      initgraph(&gdriver,&gmode,"");
      // read result of initialization
      errorcode = graphresult();
      if( errorcode != grOk) // an error occurred
      {
      printf("Graphecs error :%s\n",grapherrormsg(errorcode));
            printf("Press any key to halt");
            getch();
            exit(1); // terminiate with an error code
      }
```

```
    x = getmaxx()/2;
    y = getmaxy()/2;
    // output some messages
    sprintf(msg,"Graphics buffer size: %d", BUFSIZE);
    settextjustify(CENTER_TEXT,CENTER_TEXT);
    outtextxy(x,y,msg);
    sprintf(msg, "Old graphics buffer size: %d",oldsize);
    outtextxy(x,y + textheight("W"),msg);
    // clean up
    getch();
    closegraph();
    return 0;
}
```

عند تنفيذ البرنامج ستظهر العبارتان

graphics buffer size : 1000

old Graphics buffer size : 4096

في وسط شاشة الرسم ويمكن إضافة أية أشكال إلى البرنامج.

- الدالة **registerbgidriver( ):**

وصيغتها العامة كما يلي:

```
void far registerbgidriver (void (*Driver)(void));
```

تقوم الدالة **registerbgidriver( )** بتسجيل السواقة BGI مع نواة الرسم، المؤشر Driver يخبر النواة الرسم عن مكان وجود السواقة. عندما يحصل خطأ، فإن هذه الدالة ستعيد قيمة سالبة. استخدام الدالة GraphErrorMsg والدالة GraphResult لعرض الخطأ في هذه الحالة. الدالة RegisterBGIDriver مفيدة في حالة التحميل المسبق لسواقات رسوم مختلفة في الذاكرة وبذلك استغرق أقل من وقت عند التبديل من سواقة إلى أخرى. حالما يتمّ تسجيل السواقة فسيتمّ استخدامها من قبل الدالة Initgraph وذلك بتمرير رقم السواقة له.

البرنامج التالي مكمل للبرنامج السابق ويعمل على فتح ملف السواقة DriverF

وقراءته وتسجيله.

مثال:

يستدعي البرنامج التالي الدالة **registerbgidriver( )**

```
# #include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
        // request auto detection
        int gdriver = DETECT,gmode,errorcode;
        /* register a driver that was added into graphics.lib
        for information on adding the driver, see the
        BGIOBJ section of UTIL.Doc*/
        errorcode = registerbgidriver(EGAVGA_driver);
        if( errorcode != grOk) // an error occurred
        {
        printf("Graphecs error :%s\n",grapherrormsg(errorcode));
                printf("Press any key to halt");
                getch();
                exit(1); // terminiate with an error code
        }

        // initialize graphics and local variables
        initgraph(&gdriver,&gmode,"");

        // read result of initialization
        errorcode = graphresult();
        if( errorcode != grOk) // an error occurred
        {
        printf("Graphecs error :%s\n",grapherrormsg(errorcode));
                printf("Press any key to halt");
                getch();
                exit(1); // terminiate with an error code
        }
        // drow a line
        line (0,0,getmaxx(),getmaxy());
```

```
    // clean up
    getch();
    closegraph();
    return 0;
}
```

☜ الدالة closegraph( ):

صيغتها العامة كما يلي:

```
void far closegraph(void);
```

تستخدم الدالة closegraph( ) لإعادة تخزين العرض إلى النمط الذي كان موجوداً قبل الدخول إلى نمط الرسوم أي نمط النص. وكذلك فإن الدالة closegraph( ) يمكن على تحرير الذاكرة المستخدمة من قبل نظام الرسوم بعد استدعائه وعادة يستدعي هذا الدالة في نهاية كل برنامج.

☜ الدالة setaspectratio( ):

صيغتها العامة كما يلي:

```
void far setaspectratio (int xasp,int yasp);
```

تغير الدالة setaspectratio( ) نسبة الهيئة التي تستخدم لعرض الرسومات البيانية والتي تساوي Xasp/Yasp.

☜ الدالة getaspectration( ):

صيغتها العامة كما يلي:

```
void far getaspectratio (int far* xasp,int far* yasp);
```

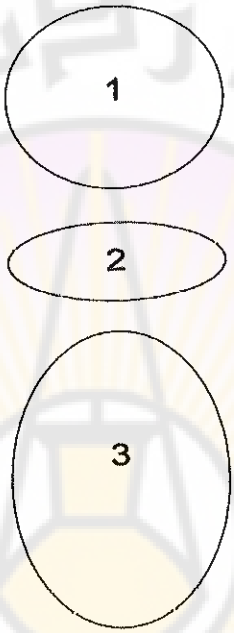
يعيد هذا الدالة قيماً في البارمترات Xasp و Yasp والتي يمكن من خلالها حساب قوة التحليل المؤثرة لشاشة العرض ويسمى أحياناً عامل التصحيح أو نسبة الهيئة من قسمة Xasp على Yasp.

مثال:

البرنامج التالي يستدعي كلٌ من الدالتين ( )setaspectratio، ( )getaspectratio .

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
        // request auto detection
        int gdriver = DETECT,gmode,errorcode;
        int xasp,yasp,midx,midy;
        // initialize graphics and local variables
        initgraph(&gdriver,&gmode,"");
        // read result of initialization
        errorcode = graphresult();
        if( errorcode != grOk) // an error occurred
        {
        printf("Graphecs error :%s\n",grapherrormsg(errorcode));
                printf("Press any key to halt");
                getch();
                exit(1); // terminiate with an error code
        }
        midx = getmaxx()/2;
        midy = getmaxy()/2;
        setcolor(getmaxcolor());
        // get current aspect ratio settings
  getaspectratio(&xasp,&yasp);

        // draw normal circle
        circle(midx,midy,100);
        getch();
        // clear the screen
        cleardevice();
        // adjust the aspect for a wide circle
        setaspectratio (xasp/2,yasp);
 circle (midx,midy,100);
 getch();
 // clear the screen
        cleardevice();
 // adjust the aspect for a wide circle
```

```
        setaspectratio (xasp,yasp/2);
  circle (midx,midy,100);
        // clean up
        getch();
        closegraph();
        return 0;
}
```



الشكل (12-1) يوضح خرج البرنامج

➪ الدالة ( )installuserdriver:

وصيغتها العامة كما يلي:

```
int far installuserdriver (char far* name,int huge (* detect)(void));
```

تسمح الدالة ( )installuserdriver للمستخدم بتركيب جهاز سواقة جديد. الدالة تعيد رقم السواقة المسند إلى هذه السواقة الجديدة، البارميتر name هو اسم الملف للسواقة الجديدة. البارمتر AutoDetectPtr هو مؤشر الدالة detetct الأتوماتيكي للجهاز الجديد، إذا كان هناك واحداً. إذا حصل أي خطأ عند تركيب السواقة الجديدة، فإن هذه الدالة ستعيد القيمة 11-.

مثال:

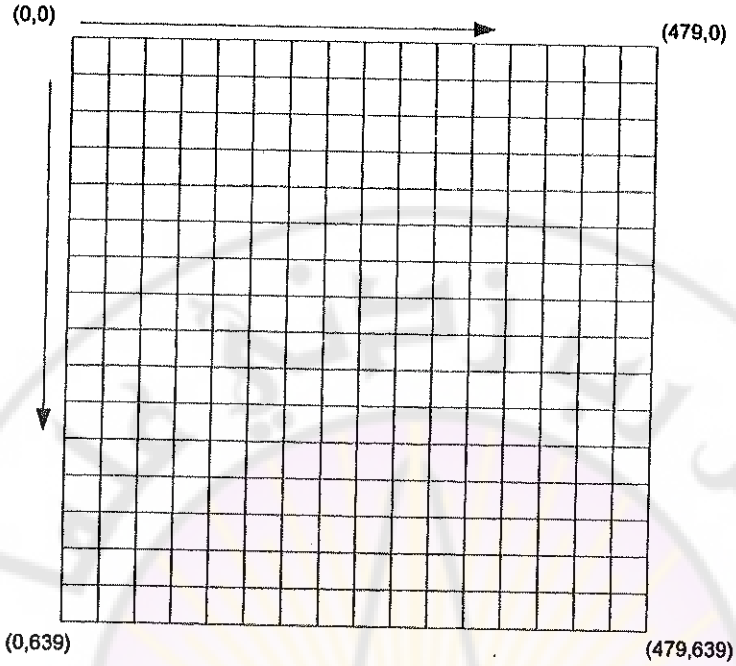
البرنامج التالي يستدعي الدالة ( )installuserdriver.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
// function prottypes
int huge detectEGA(void);
void checkerrors(void);
int main(void)
{
        int gdriver,gmode;
        // install a user writen device driver
        gdriver = installuserdriver ("EGA",detectEGA);
        // must force use of detection routine
        gdriver = DETECT;
        // check for any installation errors
        checkerrors();
        // initialize graphics and local variables
        initgraph(&gdriver,&gmode,"");
        // check for any installation errors
        checkerrors();
        // draw a line
        line (0,0,getmaxx(),getmaxy());
```

```
        // clean up
        getch();
        closegraph();
        return 0;
}
// detect EGA or VGA cards
int huge detectEGA(void)
{
        int driver,mode,sugmode = 0;
        detectgraph(&driver, &mode);;
        if((driver == EGA) || (driver == VGA))
                // return suggest vidio mode number
                return sugmode;
        else
                return grError;
}
// check for and report any graphic errors
void checkerrors(void)
{
        int errorcode;
        // read result of last graphics operation
        errorcode = graphresult();
        if( errorcode != grOk) // an error occurred
        {
        printf("Graphecs error :%s\n",grapherrormsg(errorcode));
                printf("Press any key to halt");
                getch();
                exit(1); // terminiate with an error code
        }
}
```

عند التنفيذ تظهر الرسالة

press any key to halt



الشكل (12-2) نظام الإحداثيات للشاشة VGA

إن المحاور على الشاشة لاتحوي قيماً سالبة لإحداثيات x,y على الشاشة مطلقاً، وكذلك يجب استخدام قيم صحيحة فقط على الشاشة.

3- رسم النقطة:

العنصر الأساسي لرسم أية صورة على شاشة الرسم هو النقطة الضوئية Pixel وهي أصغر وحدة عرض حيث تتكون كل شاشات عرض بياني من عدد كبير من النقاط مرتبة بشكل خطوط أفقية وعمودية، الفرق الأساسي بينها هو حجم النقطة. ففي مهايء الرسم CGA وهو ذو نمط التحليل الواطيء تكون النقطة كبيرة نسبياً، لهذا فإن هناك 320 نقطة أفقياً و 200 نقطة عمودياً (أي 200x320)، أما معد الرسم VGA فهو ذو قوة التحليل العالية جداً ونقاط صغيرة جداً (480x640). وبالطبع فكلما كانت النقاط صغيرة ازداد عددها للصورة الواحدة وبالتالي فإن ذلك يعني الحصول على صور أفضل.

ولإضاءة نقاط في مواقع مختلفة من الشاشة يستخدم الدالة :

void far putpixel(int x,int y,int color);

حيث يتمّ بالدالة **putpixel** إضاءة النقطة في الموقع (x,y) على شاشة الرسم البياني باللون المحدد في البارامتر Color. قيم (x,y) يمكن أن تكون ثوابت صحيحة أو تعابير حسابية ينتج عنها قيم صحيحة موجبة.

لإضاءة النقطة (10,10) باللون الأصفر ، نكتب:

Putpixel (10,10,yello);

ويمكن الإشارة إلى اللون برقمه كما هو مبين في الجدول (2-12).

DARKGRAY	8	BLACK	0
LIGHTBLUE	9	BLUE	1
LIGHTGREEN	10	GREEN	2
LIGHTCYAN	11	CYAN	3
LIGHTRED	12	RED	4
LIGHTMAGENTA	13	MAGENTA	5
YELLOW	14	BROWN	6
WHITE	15	LIGHTGRAY	7

الجدول (2-12).

مثال:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
// request auto detection
int gdriver = DETECTE,gmode,errorcode;
```

```
int x;
// initialize graphics and local variables
initgraph(&gdriver,&gmode,"");
// read result of initialization
errorcode = graphresult();
if( errorcode != grOK) // an error occurred
{
printf("Graphecs
error:%s\n",grapherrormsg(errorcode));
printf("Press any key to halt");
getch();
exit(1); // terminiate with an error code
}
for (x=0; x < (10+getmaxx()) ; x++)
{
putpixel(x,x*0.75,14);
delay(40);
putpixel(x-2,x*0.75-1.5,0);
delay(200);
}
outtextxy(getmaxx()/2 -120,getmaxy()/2,"Press any key to
continue");
// clean up
getch();
closegraph();
return 0;
}
```

مثال:

البرنامج التالي يستخدم الدالتين getmaxx و getmaxy.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
// request auto detection
int gdriver = DETECTE,gmode,errorcode;
```

```
int midx,midy;
char xrange[80],yrange[80];
// initialize graphics
initgraph(&gdriver,&gmode,"");
// read result of initialization
errorcode = graphresult();
if( errorcode != grOK) // an error occurred
{
printf("Graphecs error
:%s\n",grapherrormsg(errorcode));
printf("Press any key to halt");
getch();
exit(1); // terminiate with an error code
}
midx = getmaxx()/2;
midy = getmaxy()/2;
// convert max resulation values into strings
sprintf(xrange,"X values range from 0 .. %d",getmaxx());
sprintf(yrange,"Y values range from 0 .. %d",getmaxy());
// display the information
settextjustify(CENTER_TEXT,CENTER_TEXT);
outtextxy(midx,midy,xrange);
outtextxy(midx,midy+textheight("W"),yrange);
// clean up
getch();
closegraph();
return 0;
}
```

وعند التنفيذ تظهر العبارتان التاليتان

```
X value wrang from 0..639
Y value rang from 0..479
```

في وسط الشاشة.

مثال:

البرنامج التالي يستدعي كلٌ من الدالتين **getx( )** و **gety( )**.

```
#include <graphics.h>
```

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
// request auto detection
int gdriver = DETECTE,gmode,errorcode;
int midx,midy;
char xrange[80],yrange[80];
// initialize graphics
initgraph(&gdriver,&gmode,"");
// read result of initialization
errorcode = graphresult();
if( errorcode != grOK) // an error occurred
{
printf("Graphecs error
:%s\n",grapherrormsg(errorcode));
printf("Press any key to halt");
getch();
exit(1); // terminiate with an error code
}
midx = getmaxx()/2;
midy = getmaxy()/2;
// convert max resulation values into strings
sprintf(xrange,"X values range from 0 .. %d",getmaxx());
sprintf(yrange,"Y values range from 0 .. %d",getmaxy());
// display the information
settextjustify(CENTER_TEXT,CENTER_TEXT);
outtextxy(midx,midy,xrange);
outtextxy(midx,midy+textheight("W"),yrange);
// clean up
getch();
closegraph();
return 0;
}
```

وعند تنفيذ البرنامج ستظهر العبارة في الموقع (319,239) على شاشة الرسم.

4- رسم الخط المستقيم Drawing Straight line.

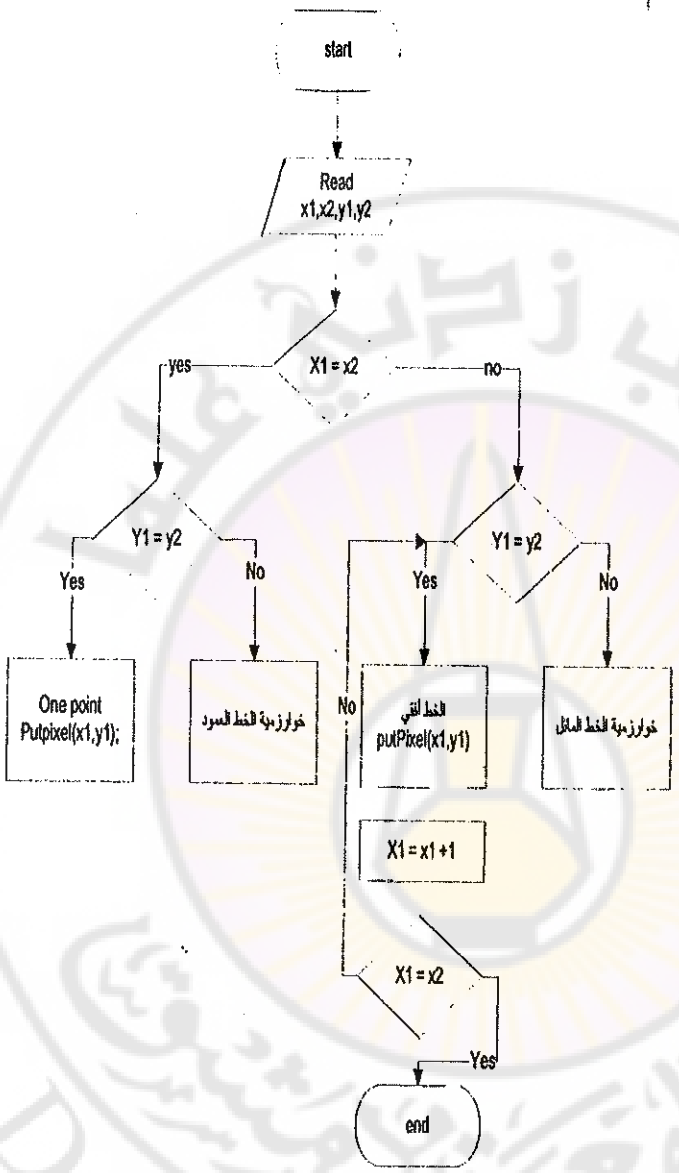
يعتمد نظام الرسوم البيانية على الخوارزميات Algorithms والتي لا تحتاج أن تكون مرتبطة بعامل قياس محدد، أي أنه يمكن استخدام الخوارزمية نفسها لرسم أشكال هندسية أخرى.

لرسم خط أفقي (موازٍ للمحورox) بين أي نقطتين مثل (x_1,y_1) و (x_2,y_2)، تكون قيمة إحداثيات y فيه ثابتة أي أن $y_1=y_2$ وقيم x تتغير من x_1 إلى x_2 تدريجياً بزيادة أو نقصان نقطة واحدة في كل خطوة.

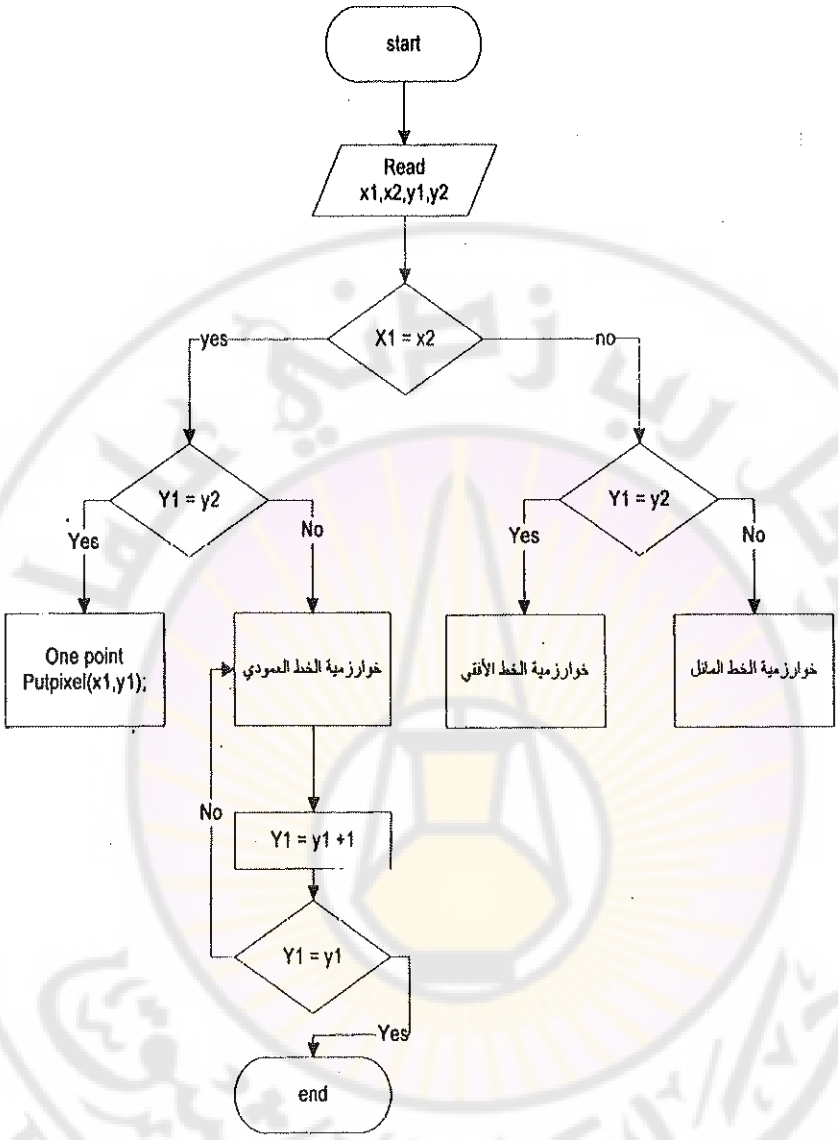
ولرسم خط عمودي (موازٍ للمحورoy) بين أي نقطتين مثل (x_1,y_1) و (x_2,y_2)، تكون قيمة إحداثيات x فيه ثابتة أي أن $x_1=x_2$ وقيم y تتغير من y_1 إلى y_2 تدريجياً بزيادة أو نقصان نقطة واحدة في كل خطوة.

ولرسم خط مستقيم بزاوية مقدارها 45 درجة نعطي قيم x,y بزيادة متساوية لكل منهما في كل خطوة.

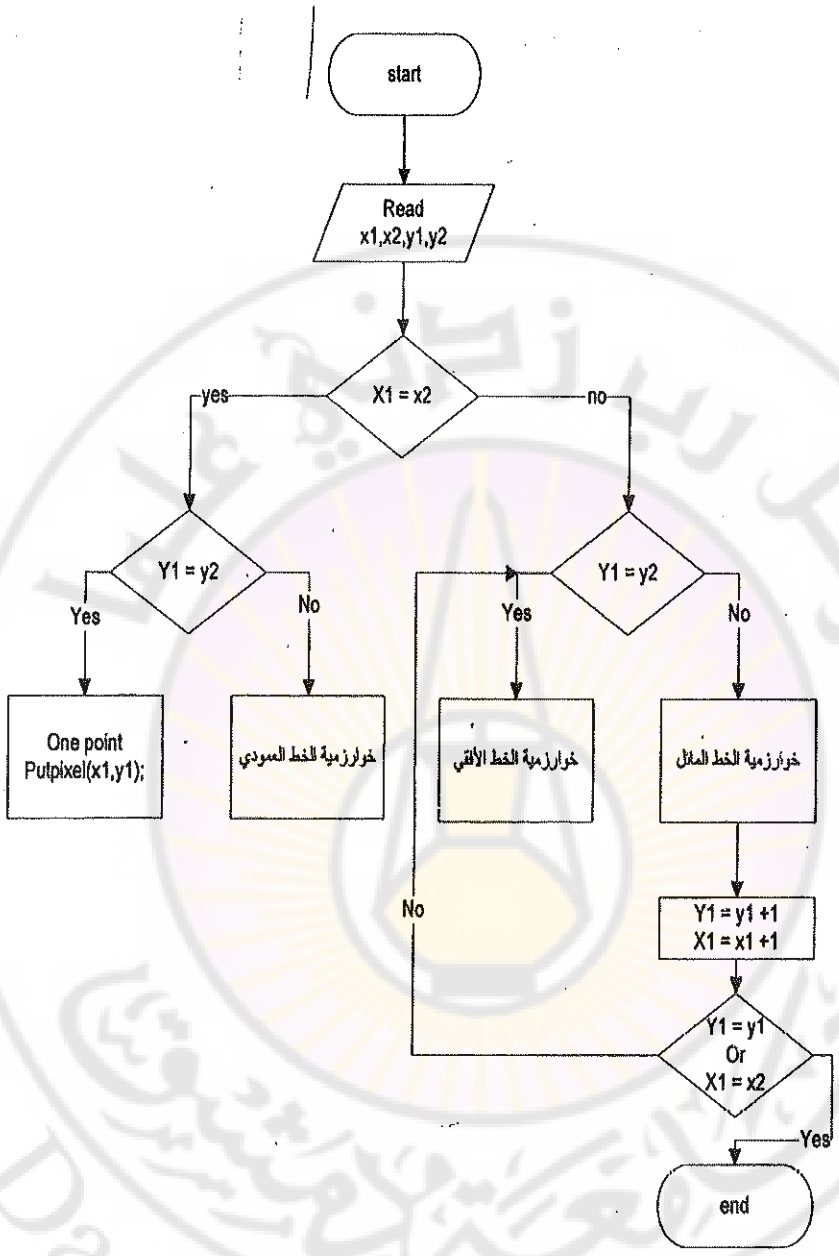
ولرسم خط وبأي ميل لأنه قد تتغير قيم x و y في الوقت نفسه وبنسب مختلفة، فلرسم خط مستقيم بين النقطتين (x_1,y_1) و (x_2,y_2) نبدأ بإضاءة النقطة الأولى (x_1,y_1) ، لكن العملية ستصبح أكثر تعقيداً ابتداءً من النقطة الثانية وذلك لصعوبة تحديد موقعها الصحيح وهكذا بالنسبة لباقي نقط المستقيم. هذا يتمّ بخوارزمية تحسب الإحداثيات.



شكل (12-3) المخطط التدفقي لخوارزمية رسم الخط المستقيم الأفقي.



شكل (12-4) المخطط التدفقي لخوارزمية رسم المستقيم العمودي



شكل (12-5) المخطط التدفقي لخوارزمية رسم المستقيم المائل بزاوية 45 درجة

5- خوارزميات رسم المستقيم.

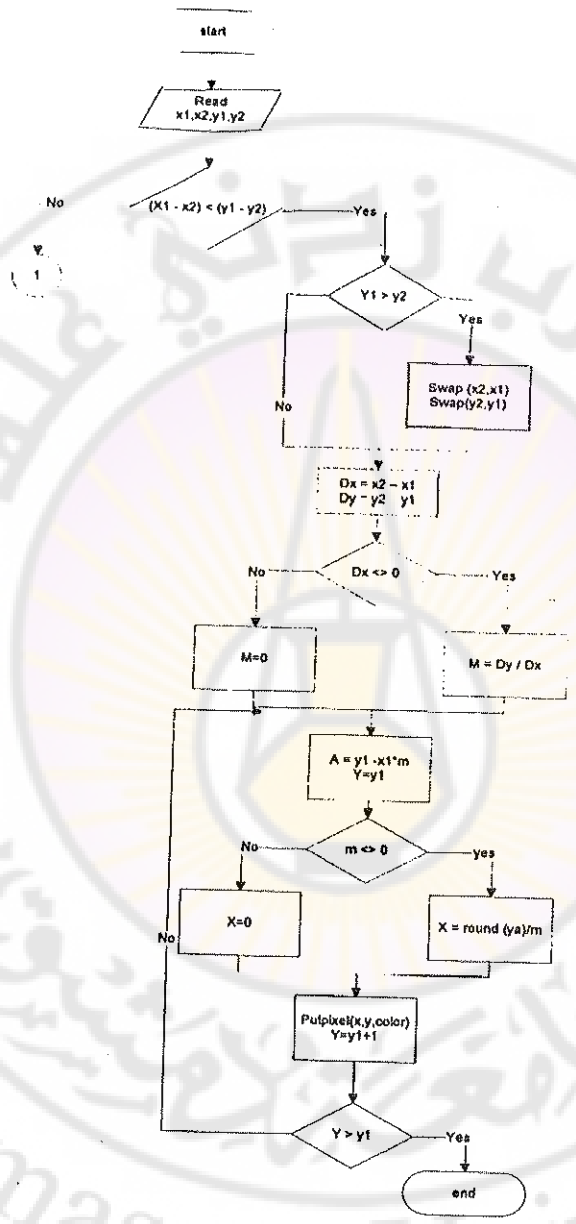
أ- الخوارزمية المباشرة:

المعادلة الجبرية للمستقيم:

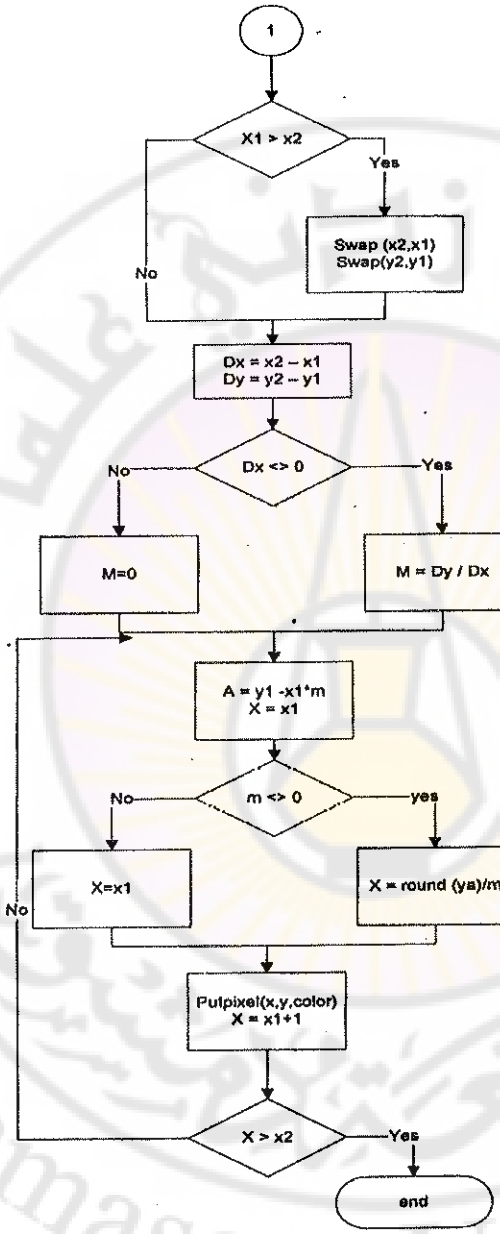
$$y = A + Bx$$

حيث y هي الإحداثي العمودي، x الإحداثي الأفقي، A هي ثابت، B تمثل ميل الخط المستقيم. بمعرفة كل من A, B سيكون من السهل رسم الخط المستقيم.

إن الميل B هو النسبة بين تغير الإحداثيات الإفقي مع تغير الإحداثيات العمودية. فإذا كانت $x_1 = x_2$ فإن B تكون صفراً، يكون المستقيم عمودياً. عندما نحسب B يكون حساب A سهلاً باستخدام زوج الإحداثيات. بعد معرفة A و B تصبح الخوارزمية جاهزة.



شكل (12-6) المخطط التدفقي للخوارزمية المباشرة لرسم مستقيم



تابع الشكل (12-7) المخطط التدفقي للخوارزمية المباشرة لرسم مستقيم بين أي

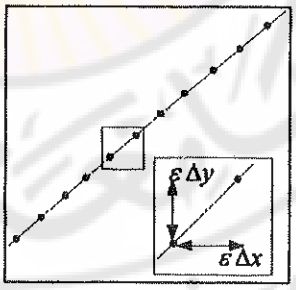
ب- خوارزمية محلل الفروق الرقمية البسيط

The simple Digital Differential analyzer SDDA

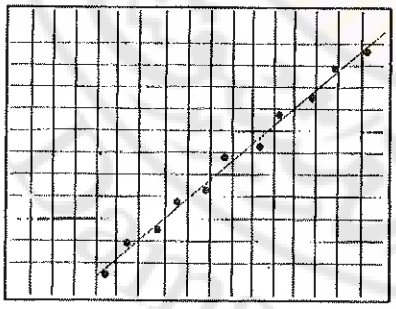
نعطى نقطة البدء للمستقيم (x_1, y_1) ونقطة النهاية (x_2, y_2) ثم نحسب قيم

$$\Delta x = abs(x_2 - x_1) \quad \text{و} \quad \Delta y = abs(y_2 - y_2)$$

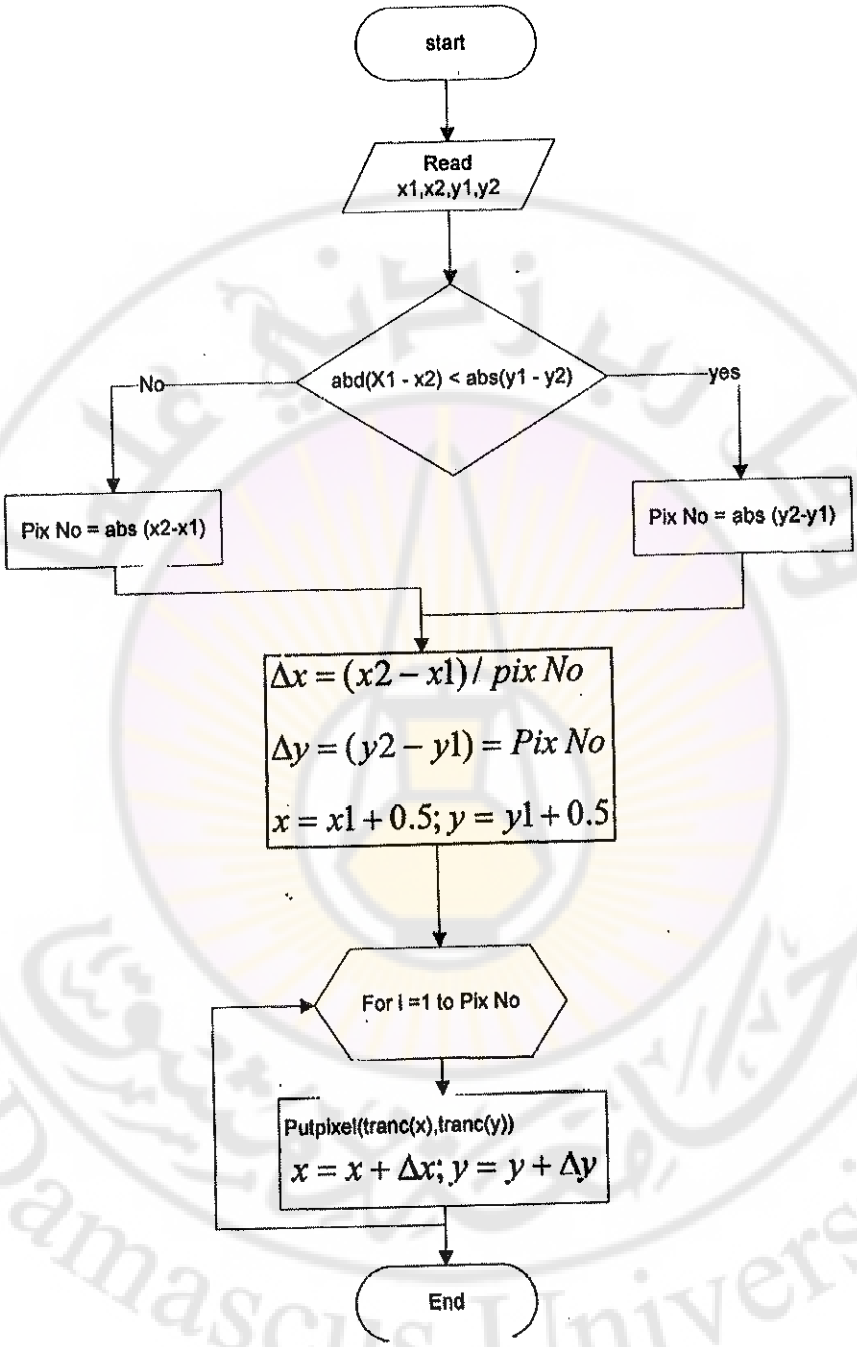
وتستخدم القيمة الأكبر لأي من Δx و Δy كتخمين لطول المستقيم (عدد النقاط Length في البرنامج) ويتمّ توليد نقاط المستقيم بتكرار زيادة قيمة x بمقدار $\Delta x/length$ وقيمة y بمقدار $\Delta y/Length$ إلى أن يتمّ الوصول إلى آخر نقطة وهي Length (تمثل طول المستقيم) بحيث لا تزيد قيم $\varepsilon\Delta x$ أو $\varepsilon\Delta y$ عن واحد، حيث ε هي أي مقدار صغير ، أي يتمّ توليد خطوة واحدة باتجاه الحركة الأكبر أي بزيادة مقدارها 1μ لأحد الإحداثيين. تكون قيم x, y المولدة حقيقية وليست قيم صحيحة، وبما أنه لا يمكن رسم القيماً الحقيقية على الحاسوب فيتمّ التقريب rounding إلى أقرب قيمة صحيحة بعد كل زيادة ورسم النقطة، ولكن تمّ استخدام البتر truncating هنا بدلاً من التقريب بعد إعطاء قيم ابتدائية لكل من x, y بمقدار 0.5 لتحقيق تقريب معقول ، حتى لانرسم النقطة نفسها مرتين. ومن عيوب هذه الخوارزمية هي استخدام القيم الحقيقية التي تتطلب وقت معالجة كبير نسبياً.



شكل (12-8)



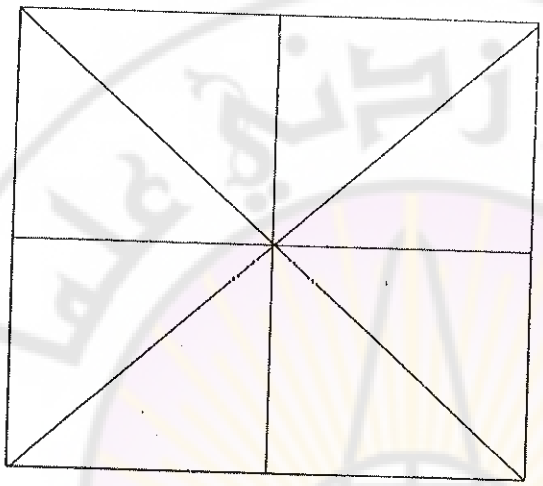
شكل (12-9)



شكل (12-10) المخطط التدفقي لخوارزمية SDDA لرسم المستقيم

مثال:

اكتب برنامجاً لرسم مستقيم بين نقطتين، حيث البرنامج يستخدم الخوارزمية SDDA يطلب البرنامج إدخال النقاط عم طريق لوحة المفاتيح.



شكل (11-12)

```
#include <iostream.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
int main(void)
{
        // request auto detection
        int gdriver = DETECT,gmode,errorcode;
        // initialize graphics
        initgraph(&gdriver,&gmode,"");
        // read result of initialization
        errorcode = graphresult();
        if( errorcode != grOk) // an error occurred
        {
                printf("Graphecs error
:%s\n",grapherrormsg(errorcode));
```

```
            printf("Press any key to halt");
            getch();
            exit(1); // terminiate with an error code
        }
// define some varibles
int x1,pixno,x2,y1,y2,dx,dy;
float x,y;
cout << "enter the point(1) - 2 coordinates - :";
cin >> x1 >> y1;
cout << "enter the point(2) - 2 coordinates - :";
cin >> x2 >> y2;
cleardevice();
if (abs (x1 - x2) < abs (y2 - y1))
      pixno = abs(y2 -y1);
else
  pixno = abs(x2 -x1);
dx = (float) (x2-x1)/pixno;
dy = (float) (y2-y1) /pixno;
x = (float) x1 + 0.5;
y = (float) y1 +0.5;
for (int i=1; i<pixno;i++)
{
      putpixel(floor(x),floor(y),15);
  x = x+ dx;
  y = y+dy;
}
      // clean up
      getch();
      closegraph();
      return 0;

}
```

ج - خوارزمية محلل الفروق الصحيحة (خوارزمية برزنهام)

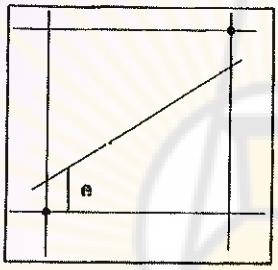
Digital Differential Integer Analyzer (DDIA)

مصممة الخوارزمية لتغير قيمة أحد الإحداثيين بمقدار 1 في كل تكرار اعتماداً على قيمة (Error) c، حيث c تمثل المسافة العمودية بين النقطة المرسومة والإحداثي

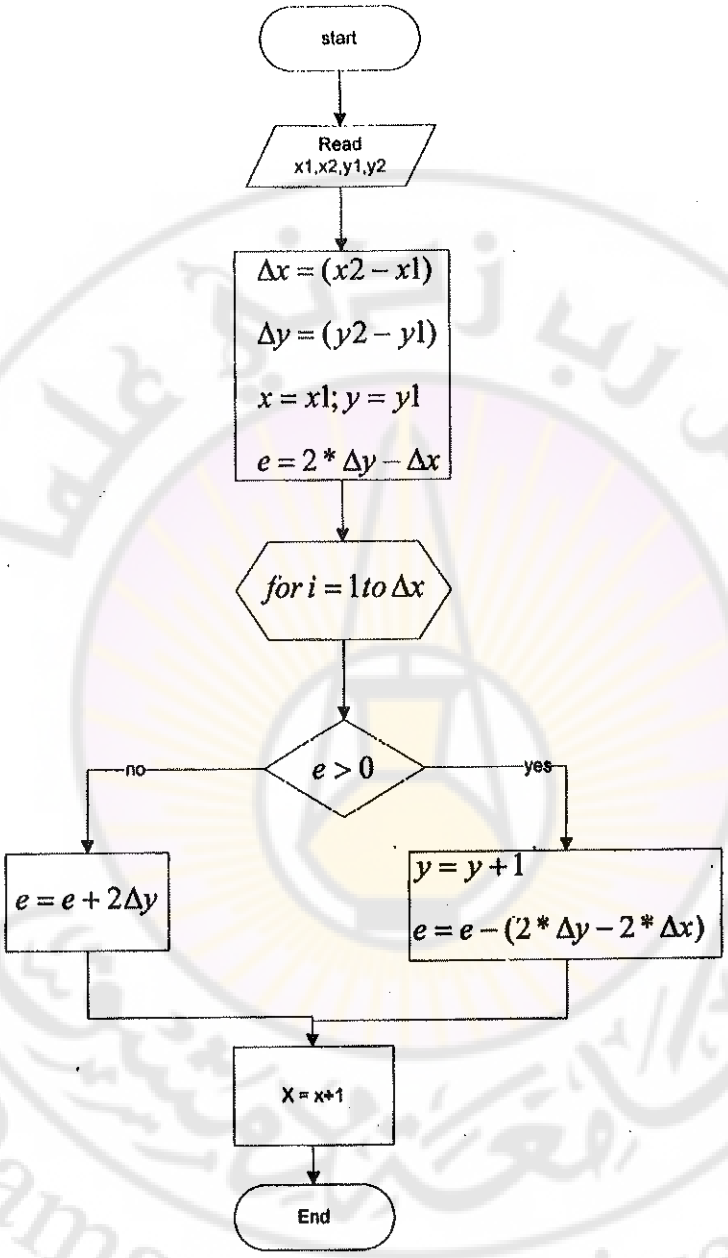
ذو التغير الأكبر (المسافة بين المسار الحقيقي للمستقيم والنقطة الحقيقية المتولدة)، شكل (12-12).

يضاف في كل تكرار الميل $\Delta x/\Delta y$ إلى e بعد أن يتمّ اختبار قيمة e، فإذا كانت موجبة يعني أن المسار الحقيقي للمستقيم يقع فوق النقطة الجارية. لهذا يتمّ زيادة الإحداثي y وتقل قيمة e بواحد، أما إذا كانت e سالبة فتبقى قيمة y كما هي.

إن الكلفة الزمنية لهذه الخوارزمية في الوقت المطلوب لعملية القسمة لحساب قيمة e، وعند زيادة قيمة e عالية. ويمكن الاستغناء عن هذه القسمة بضرب ث بثابت، واختبار إشارة e فقط. تعتبر الخوارزمية فعالة عند تطبيقها على المعالجات البسيطة.



شكل (12-12)



شكل (12-13) المخطط التدفقي لخوارزمية DDIA لرسم المستقيم

مثال:

اكتب برنامجاً يستخدم الخوارزمية DDIA لرسم خط مستقيم بين أي نقطتين، يتمّ إدخالهما من لوحة المفاتيح.

```
#include <iostream.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
int main(void)
{
        // request auto detection
        int gdriver = DETECT,gmode,errorcode;
        // initialize graphics
        initgraph(&gdriver,&gmode,"");
        // read result of initialization
        errorcode = graphresult();
        if( errorcode != grOk) // an error occurred
        {
                printf("Graphecs error
:%s\n",grapherrormsg(errorcode));
                printf("Press any key to halt");
                getch();
                exit(1); // terminiate with an error code
        }
 // define some varibles
 int x1,pixno,x2,y1,y2,dx,dy,x,y;
 float e;
 cout << "enter the point(1) - 2 coordinates - :";
 cin >> x1 >> y1;
 cout << "enter the point(2) - 2 coordinates - :";
 cin >> x2 >> y2;
 cleardevice();
 x = x1;
 y = y1;
 dx = x2 - x1;
 dy = y2 - y1;
```

```
e = 2 * dy -dx;
for (int i = 1 ; i < dx; i++)
{
      putpixel(x,y,3);
  if (e > 0)
  {
     y++;
    e = e + (2 * dy - 2 * dx);
  }
  else
      e = e + 2 * dy;
  x++;
}
// clean up
      getch();
      closegraph();
      return 0;
}
```

الدالة line( ) ⇦

لرسم خط مستقيم بين نقطتين معلومتين

صيغتها العامة كما يلي:

```
void far line(int x1, int y1, int x2, int y2);
```

حيث ترسم مستقيماً ابتداءً من النقطة ذات الإحداثيات (x1,y1) وانتهاءً بالنقطة ذات الإحداثيات (x2,y2) وباللون قيد الاستخدام.

مثال:

ارسم مستقيماً من النقطة (30,30) إلى النقطة (180,230).

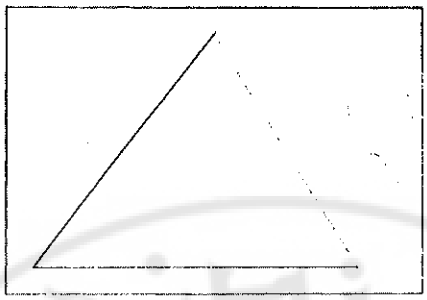
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
```

```
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int xmax, ymax;
    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");
    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
    }
    setcolor(getmaxcolor());
    xmax = getmaxx();
    ymax = getmaxy();
    /* draw a diagonal line */
    line(0, 0, xmax, ymax);
    line (xmax,0,0,ymax);
    line (xmax,ymax/2,0,ymax/2);
    line (xmax/2,0,xmax/2,ymax);
    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

6- رسم الأشكال باستخدام الخطوط المستقيمة فقط

يمكن استخدام دالة رسم النقطة والمستقيم الواردة في رسم أشكال ثنائية الأبعاد مختلفة. العبارات التالية تستدعي الدالة line ثلاث مرات لرسم مثلث رؤوسه النقاط (320,50) ، (200,200) ، (400,200):

```
line (320,50,200,200);
line(200,200,400,200);
line(400,200,320,50);
```



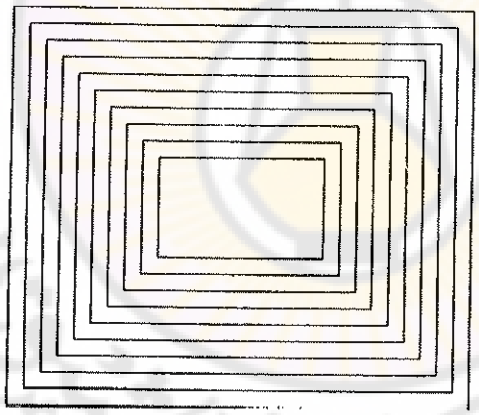
شكل (12-14)

مثال:

اكتب برنامجاً لرسم مربعات متداخلة:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void square (int x1,int y1,int x2,int y2);
int main(void)
{
  /* request auto detection */
  int gdriver = DETECT, gmode, errorcode;
  int xmax, ymax;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
  }
  int x1 = 150 , y1 = 130,x2 = 300 , y2 = 280;
  int count = 0;
  int i = 10;
  while (count < 10)
  {
```

```
        count++;
      square (x1,y1,x2,y2);
      x1 += i; y1 += i;
      x2 -= i; y2 -= i;
    }
    /* clean up */
    getch();
    closegraph();
    return 0;
}
void square (int x1,int y1,int x2,int y2)
{
line(x1,y1,x2,y1);
line (x2,y1,x2,y2);
line(x2,y2,x1,y2);
line(x1,y2,x1,y1);
}
```



شكل (12-15)

☜ الدالة moveto:

وصيغتها العامة:

```
void moveto(int x,int y);
```

تحرك الموقع الحالي على الشاشة إلى الموقع المحدد بالإحداثيات x,y. فالاستدعاء **moveto (90,90)** يتمّ بواسطة التحرك إلى النقطة في الموقع (90,90).

- الدالة **lineto( ):**

صيغتها العامة:

```
void lineto(int x,int y);
```

ترسم مستقيماً ابتداءً من الموقع الحالي إلى الموقع (x,y) المحدد في الدالة ، حيث يبدأ الرسم من النقطة (0,0) إذا لم يسبقه أي استدعاء.

```
lineto (50,50);
moveto (90,90);
lineto (350,410);
```

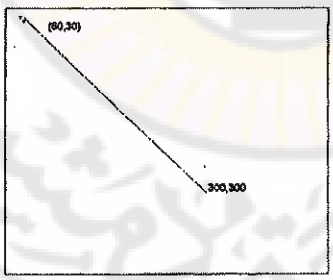
الاستدعاء الأول ترسم مستقيم من النقطة (0,0) إلى النقطة (50,50)، أما الاستدعاء الثاني فترسم المستقيم من النقطة (90,90) إلى النقطة (350,410) وذلك لأنه تمّ التحرك أولاً إلى الموقع (90,90) باستدعاء moveto وقبل الاستدعاء الثاني لـ lineto.

مثال:

البرنامج التالي يستخدم الدالتين **moveto( )** و **lineto( )** ليرسم مستقيماً من النقطة (60,30) إلى النقطة (300,300).

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  char msg[80];
```

```
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
  }
  /* move the CP to location (60,30) */
  moveto(60, 30);
  /* create and output a message at (20,30) */
  sprintf(msg, " (%d, %d)", getx(), gety());
  outtextxy(60,30, msg);
  /* draw a line to (100,100) */
  lineto(300, 300);
  /* create and output a message at CP */
  sprintf(msg, " (%d, %d)", getx(), gety());
  outtext(msg);
  /* clean up */
  getch();
  closegraph();
  return 0;
}
```



شكل (12-16)

مثال:

البرنامج التالي يستخدم كلٌ من الدالتين القياسيتين ( )lineto، ( )moveto ليرسم

شكل رباعياً (مربعاً أو مستطيلاً) بعد أن يطلب إدخال قيمة اللون المطلوب للرسم وكذلك قيم الإحداثيات نقطتي نهاية القطر الرئيسي للشكل الرباعي:

```
#include <iostream.h>
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  char msg[80];
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
  }
  int x,w,y,x1,y1;
  cout <<"Enter the color :";
  cin >> w;
  setcolor(w);
  cout <<"enter the 4 numbers : ";
  cin >> x >> y >> x1 >> y1;
  cleardevice();
  moveto(x,y);
  lineto(x1,y);
  lineto(x1,y1);
  lineto(x,y1);
  lineto(x,y);
  /* clean up */
  getch();
  closegraph();
  return 0;
```

}

☜ الدالة moverel( ):

الصيغته العامة كما يلي:

```
void moverel (int dx,int dy);
```

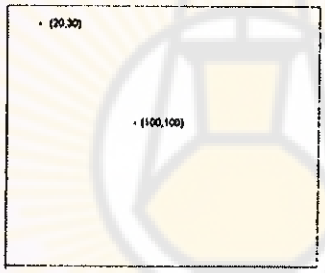
يتمّ بواسطة الدالة moverel( ) التحرك من Dx من النقاط أفقياً و Dy من النقاط عمودياً نسبة للموقع الحالي وليس لنقطة الأصل.

مثال:

البرنامج التالي يستدعي الدالة moverel( )، حيث يتحرك أولاً إلى النقطة (20,30) بوساطة الدالة moveto( ) ويرسم نقطة بواسطة الدالة putpixel( ) ثم يتحرك 100 نقطة على المحور x و 100 نقطة على المحور y بالنسبة للنقطة الأولى، ويرسم نقطة هناك.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  char msg[80];
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);              /* terminate with an error code */
  }
  /* move the CP to location (20,30) */
```

```
    moveto(20,30);
    /* plot a pixel at the CP */
    putpixel(getx(), gety(), getmaxcolor());
    /* create and output a message at (20,30) */
    sprintf(msg, " (%d, %d)", getx(), gety());
    outtextxy(20,30, msg);
    /* move to a point a relative distance away from the current CP */
    moverel(100, 100);
    /* plot a pixel at the CP */
    putpixel(getx(), gety(), getmaxcolor());
    /* create and output a message at CP */
    sprintf(msg, " (%d, %d)", getx(), gety());
    outtext(msg);
    /* clean up */
    getch();
    closegraph();
    return 0;
}
```



شكل (12-17)

☜ الدالة linerel( ):

صيغتها العامة كما يلي :

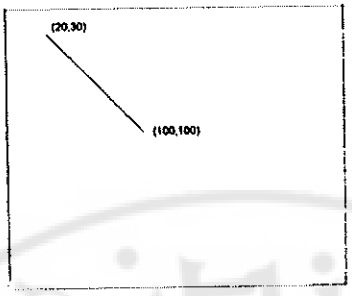
```
void linerel(int dx,int dy);
```

ترسم مستقيم من الموقع الحالي إلى الموقع الذي يبعد عنه Dx من النقاط أفقياً و Dy من النقاط عمودياً، فإذا كنا في الموقع (50,50) على شاشة الرسم فالاستدعاء linerel(150,150) سيؤدي إلى مستقيم من النقطة (50,50) إلى النقطة (200,200).

مثال:

البرنامج التالي يستدعي الدالة linerel( ) ليرسم مستقيماً .

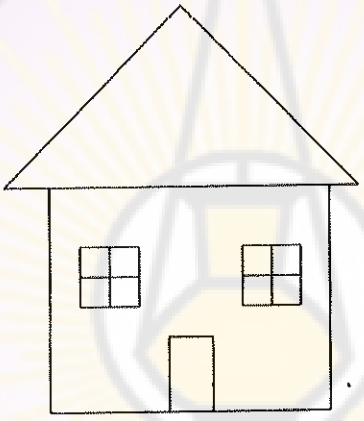
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  char msg[80];

  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
  }
  /* move the CP to location (20,30) */
  moveto(20,30);
  /* create and output a message at (20,30) */
  sprintf(msg, " (%d, %d)", getx(), gety());
  outtextxy(20,30, msg);
  /* draw line to a point a relative distance away from current CP */
  linerel(100, 100);
  /* create and output a message at CP */
  sprintf(msg, " (%d, %d)", getx(), gety());
  outtext(msg);
  /* clean up */
  getch();
  closegraph();
  return 0;
}
```



شكل (12-18)

مثال:

البرنامج التالي يرسم بيت



شكل (12-19)

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  char msg[80];
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
```

```
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);              /* terminate with an error code */
  }

  setcolor (14);
  outtextxy(getmaxx()/2,getmaxy()/2,"Press Enter to see the house");
  getch();
  cleardevice();
  setcolor (7);
   /*the house roof*/
 line (320,50,200,200);
 linc (200,200,430,200);
 line (430,200,320,50);
 /*the hose*/
 rectanglc(230,200,400,350);
 /*the door*/
 rectangle(300,300,330,350);
 /*the left window*/
 line (350,250,380,250);
 line (380,250,380,280);
 line (380,280,350,280);
 line (350,280,350,250);
 linc (365,250,365,280);
 line (350,265,380,265);
 /*the Right window*/
 line (250,250,280,250);
 line (280,250,280,280);
 line (280,280,250,280);
 linc (250,280,250,250);
 line (265,250,265,280);
 line (250,265,280,265);
 /* clean up */
 getch();
 closegraph();
 return 0;  }
```

الدالة bar( ):

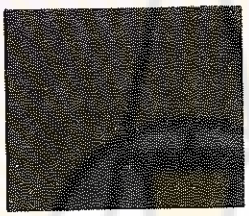
صيغتها العامة كما يلي:

void far bar (int x1, int y1, int x2 , int y2);

ترسم الدالة مستطيل، أو مربع مطلي باللون والتشكيلة قيد الاستخدام بحيث تكون الزاوية العليا اليسرى للشكل هي النقطة (x1,y1) والزاوية السفلى اليسرى هي النقطة (x2,y2) و لا ترسم الخط المحيط بالشكل.

مثال:

البرنامج التالي يستدعي الدالة **bar( )** ويعطي الخرج الموضح في الشكل حيث يستخدم تشكيلات وألوان مختلفة.



شكل (12-20)

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int midx, midy, i;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");

  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {  /* an error occurred */
```

```
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);            /* terminate with an error code */
  }

  midx = getmaxx() / 2;
  midy = getmaxy() / 2;
  /* loop through the fill patterns */
  for (i=SOLID_FILL; i<USER_FILL; i++)
  {
    /* set the fill style */
    setfillstyle(i, getmaxcolor());
    /* draw the bar */
    bar(midx-50, midy-50, midx+50, midy+50);
    getch();
  }
  /* clean up */
  closegraph();
  return 0;
}
```

- الدالة bar3d():

صيغتها العامة كما يلي:

```
void far bar3d (int x1 , int y1, int x2 , int y2, int depth, int topflag);
```

ترسم متوازي مستطيلات وتطلي واجهته باللون والتشكيلة قيد الاستخدام. حيث:

(x1,y1): تمثلان النقطة التي تمثل الزاوية اليسرى العليا للواجهة.

(x2,y2): تمثلان النقطة التي تمثل الزاوية اليمنى السفلى للواجهة.

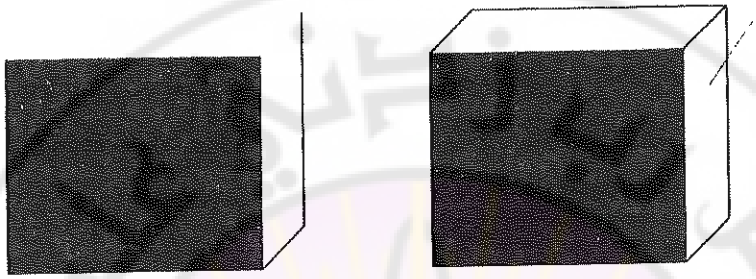
Depth: يمثل عرض القاعدة.

Topflag: يتمّ رسم القاعدة العليا (الغطاء) لمتوازي المستطيلات وهذا البارامتر هو من النوع الصحيح. ويتمّ رسم الغطاء إذا كانت قيمته 1 ولا ترسم إذا كانت القيمة صفر.

مثال:

نسند القيمة 1 للبارامتر Topflag :

```
Topfalg := 1;
bar3d (50,30,150,100,40,Topflag);
```



(ب) (أ)

شكل (12-21)

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int midx, midy, i;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);              /* terminate with an error code */
  }
  midx = getmaxx() / 2;
```

```
    midy = getmaxy() / 2;
    /* draw the 3-d bar */
    bar3d (midx -50, midy - 50, midx + 50, midy + 50, 10, 1);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

وعند استبدال القيمة 1 في الاستدعاء بالقيمة 0 كما يلي:

```
bar3d (midx -50, midy - 50, midx + 50, midy + 50, 10, 0);
```

فنحصل على الشكل الثاني.

- الدالة **drawPoly( ):**

الصيغة العامة كما يلي:

```
void far drawPoly (int numpoints, int far * PolyPoints);
```

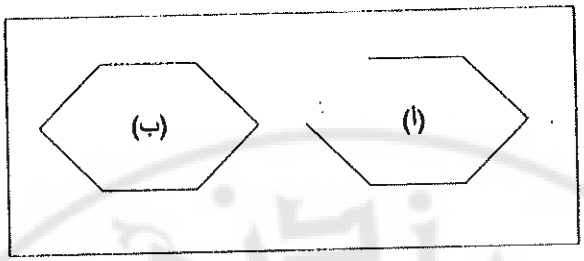
حيث:

numpoints تمثل عدد نقاط رؤوس المضلع.

polypoints*: نشير إلى مصفوفة الأعداد الصحية وعددها ضعف عدد numpoints.

كل زوج من القيم تمثل نقطة رأس من رؤوس مضلّع. لرسم المضلع مغلق ذي N ضلع يجب أن نمرر له N+1 من النقاط حيث إحداثيات النقطة الأخيرة (N+1) يجب أن تكون إحداثيات النقطة الأولى نفسها.

مثال:



شكل (12-22)

```
poly#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int maxx, maxy;
  int poly[10];  /* our polygon array */
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk){ /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);          /* terminate with an error code */
  }
  /*maxx = getmaxx();   maxy = getmaxy();*/
  poly[0] = 150;        /* first vertex */
  poly[1] = 150;
  poly[2] = 190;          /* second vertex */
  poly[3] = 190;
  poly[4] = 230;          /* third vertex */
  poly[5] = 190;
  poly[6] = 270;                  /* fourth vertex */
```

```
poly[7] = 150;
poly[8] = 230;                    /* fith vertex */
poly[9] = 110;
poly[10] = 190;                   /* fith vertex */
poly[9] = 110;
poly[12] = poly[0];     /* drawpoly doesn't automatically close */
poly[13] = poly[1];     /* the polygon, so we close it */
drawpoly(7, poly);  /* draw the polygon */
/* clean up */
getch();
closegraph();
return 0;    }
```

7- رسم الدوائر والأقواس

الدائرة هي مجموعة من النقط التي لها البعد نفسه عن المركز (x_0, y_0). هناك عدة خوارزميات لرسم الدائرة يصلح بعضها لرسم شكل القطع الناقص والحلزوني.

أ- خوارزمية المضلع المنتظم

يعبر عن معادلة الدائرة التي مركزها (x_0, y_0) ونصف قطرها a، في الإحداثيات الديكارتية بالعلاقة:

$$(x - x_0)^2 + (y - y_0)^2 = a^2 \qquad (1\text{-}12)$$

في الطرق غير تزايدية عادة يقترب المضلع المنتظم ذو الـ n ضلع من الدائرة تحسب نقاط رؤوس المضلع من العلاقات التالية:

$$x[i] = x_0 + a\cos(\frac{2\pi i}{n})$$
$$y[i] = y_0 + a\sin(\frac{2\pi i}{n}) \qquad (2\text{-}12)$$

حيث $0 \le i \le n$. يتمّ رسم الأضلاع باستخدام دوال رسم المستقيمات ولكي يكون التمثيل الدائري دقيقاً يجب أن تكون الأضلاع صغيرة جداً مقارنة بنصف القطر a، و

n كبيرة بما فيه الكفاية. ويمكن أن نحسب n من العلاقة:

$$n = \frac{2\pi}{\sin^{-1}(\frac{d}{a})}$$

حيث d هي المسافة بين المستقيمات الأفقية . وإذا كان نصف القطر a كبيراً مقارنة بالمسافة النقطية بين المستقيمات d يمكن أن تقرب هذه العلاقة إلى

$$n \approx \text{int}(\frac{2\pi a}{d})$$

مثال:

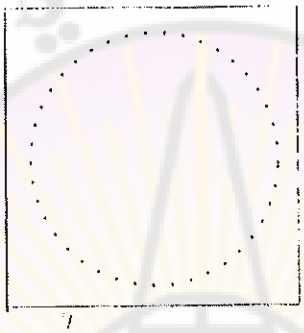
البرنامج التالي يستخدم العلاقات السابقة ليوضح كيفية تحويل المضلع إلى متعدد الأضلاع إلى دائرة بزيادة عدد الأضلاع بالتدريج من خلال البارامتر n حيث يبدأ بقيمة n=8 ويرسم المضلع الثماني ثم ينتظر حتى يضغط على مفتاح Enter ليظهر مضلع يزيد عدد أضلاعه عن السابق بعشرين ضلعاً وهكذا تكرر العمليات نفسها إلى أن تصبح (n>120) أي عدم تحقق الشرط في عبارة while.

لقد ترك أحد الأضلاع بدون رسم (أي المضلع مفتوح) وذلك ليتسنى مشاهدة كيف أن طول الضلع يقل كلما ازداد عدد الأضلاع واقترب الشكل من الدائرة حيث يصبح الضلع المفقود نقطة تقريباً ، الأشكال مرقمة بالتسلسل من 1 إلى 6 وحسب ظهورها بعد كل ضغطة للمفتاح Enter.

```
#include <graphics.h>
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <dos.h>
int main(void)
{
        clrscr();
```

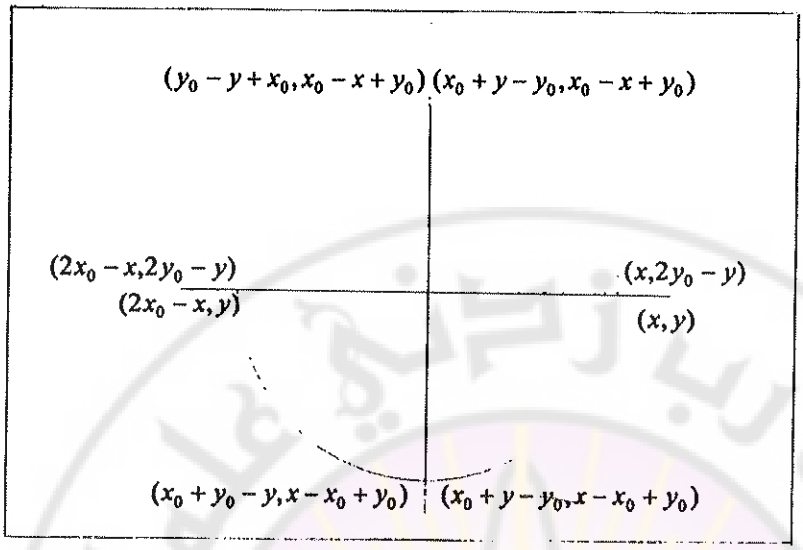
```
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int x0,y0,n,a,colr;
  float xi,yi,pi =3.14;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
  }
  cout << "Please Enter the radius :";
  cin >> a;
  cout << "\n\n Please Enter the center : \n x =";
  cin >> x0;
  cout << " y=";
  cin >> y0;
  n = 360;
  cleardevice();
  for (float i = 0; i <= n ; i += 0.1)
  {
        xi = x0 + a*cos ((2*pi)*(i/n));
                yi = y0 + a*sin ((2*pi)*(i/n));
    sound (n-1 -i);
    delay(3);
    putpixel (xi,yi,colr);
  }
  setcolor(2);
  outtextxy(getmaxx()/2-85,0,"Press any key to quite");
  nosound();

  /* clean up */
  closegraph();
  return 0;
}
```

إن الكلفة الزمنية كبيرة عندما تكون n كبيرة حيث ستكون هناك عمليات حسابية كثيرة لحساب النقاط ويمكن التقليل من هذا الحمل وذلك بالاستفادة من التناظر في الدائر، فلكل نقطة في الربع الأول توجد نقاط مناظرة في الأرباع الثلاثة الأخرى يمكن حسابها بالانعكاس حول المحور المتمركز في مركز الدائرة (x_0, y_0). فإذا كانت (x, y) نقطة على المحيط، فالنقاط $(x, 2y_0 - y)$ و $(2x_0 - x, 2y_0 - y)$ و $2x_0 - x, y$ هي الأخرى تقع على المحيط وهذه ستقلل من حسابات النقاط إلى الربع الأول.



شكل (12-23)

وبالمثل من خلال استخدام انعكاس التناظر حول الخط ذي الميل (1) الذي يمر من نقطة المركز (x_0, y_0)، فمن المعروف أنه إذا كانت (x, y) على محيط الدائرة فانعكاسها حول هذا الخط في النقطة $(y - y_0 + x_0, x - x_0 + y_0)$ سيكون أيضاً على الدائرة كما في شكل (12-23) وهذا سيقلل حسابات النقاط إلى n/8.



شكل (12-24)

ب- الخوارزمية الضمنية:

وتعتمد الخوارزمية على استخدام معادلة الدائرة التي مركزها (x_0, y_0) وبصف قطرها a وهي:

$$(x - x_0)^2 + (y - y_0)^2 = a^2 \qquad (3-12)$$

حيث:

$$y = y_0 \mu \sqrt{a^2 - (x - x_0)^2} \qquad (4-12)$$

سيتمّ استخدام الجذر التربيعي هنا بدلاً من الدوال المثلثية التي استخدمت في الطريقة السابقة. يتمّ حساب النقاط على محيط الدائرة وذلك بأن تتحرك بخطوة مقدارها 1 على طول محور x من $x_0 - a$ إلى $x_0 + a$ وحساب قيم y المتناظرة لكل قيمة من x باستخدام قواعد التناظر يتمّ حساب النقاط المناظرة في الأرباع الأخرى.

مثال:

البرنامج التالي يستخدم الخوارزمية الضمنية لرسم دائرة ويستفيد من ظاهرة التناظر لتقليل الوقت حيث تمَّ رسم نقطتين فقط وسبع نقاط مناظرة لكل منها.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
void symmetric();
int a,ax,x,y;
int x1,x2,x3,x4,x5,x6,x7,x8;
int y1,y2,y3,y4,y5,y6,y7,y8;
int main(void)
{
        clrscr();
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int x0,y0,n,a,colr;
  float xi,yi,pi =3.14;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  x = getmaxx()/2;
  y = getmaxy()/2;
  putpixel (x,y,4);
  a = 50;
  x1 = x +a -3;
  ax = floor(sqrt(abs(pow(a,2)-pow((x1-x),2))));
  y1 = y -ax;
  symmetric();
  x1 = x-a+10;
  ax = floor(sqrt(abs(pow(a,2)-pow((x1-x),2))));
  y1 = y -ax;
  symmetric();
  outtextxy(x-85,y*2-50,"Press any key to quite");
  /* clean up */
  getch();
  closegraph();
```

```
  return 0;
}
void symmetric()
{
          x2 = y1 -y+x;
    y2 = x1 - x+y;
    putpixel(x2,y2,15);
    x3 = 2*x - x1;
    y3 = 2*y - y1;
    putpixel(x3,y3,14);
    x4 = x1;
    y4 = y3;
    putpixel(x4,y4,14);
    x5 = x1; y5 = y4;
    putpixel(x5,y5,14);
    x6= y2 -y+x;
    y6 = x2 -x+y;
    putpixel(x6,y6,15);
    x7 = y3-y+x;
    y7 = x3-x+y;
    putpixel(x7,y7,15);
    x8 = y4-y+x;
    y8 = x4-x+y;
    putpixel(x8,y8,15);
}
```

ولكن هذه ليست أفضل طريقة لتوليد الدائرة، فبالإضافة إلى الحسابات الكثيرة التي تتضمنها كل خطوة فإن المسافات بين النقاط المرسومة غير متجانسة.

ج- خوارزمية التمثيل التزايدي

إحداثيات أي نقطة على محيط الدائرة باستخدام الإحداثيات القطبية تكون كما يلي:

$$x_i = r\cos\theta$$
$$y = r\sin\theta \qquad (5\text{-}12)$$

مثال:

يستخدم البرنامج التالي الخوارزمية ولكن بدون استخدام التناظر.

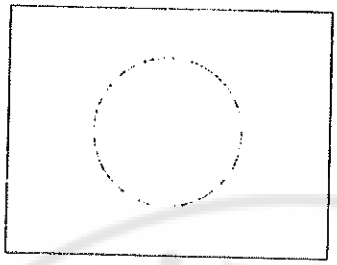
```
#include <graphics.h>
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <dos.h>
int main(void)
{
        clrscr();
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int x,y,z,r;
  double k;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
  }
  r=100;
  for(k=0; k<= 377; k+=0.6)
  {
       x=r*sin(k*(22/7)/180) +getmaxy()/2;
    y=r*cos(k*(22/7)/180) +getmaxy()/2;
    delay(2);
    putpixel(x,y,14);
  }
  /* clean up */
  getch();
  closegraph();
```

```
  return 0;
}
```

يستخدم البرنامج خوارزمية لرسم الدائرة ويستفيد من ظاهرة التناظر.

```
#include <graphics.h>
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <dos.h>
void symmetry();
int a,ax,x,y;
int x0,x1,x2,x3,x4,x5,x6,x7,x8;
int y0,y1,y2,y3,y4,y5,y6,y7,y8;
int main(void)
{
        clrscr();
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int x,y,z,r;
  double k;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
  }
  r=100;
  x0 = getmaxx()/2;
  y0 = getmaxy()/2;
  for(k=0; k<= 90; k+=0.6)
  {
        x1=r*sin(k*(22/7)/180)+x0;
```

```
    y1=r*cos(k*(22/7)/180)+y0;
    delay(2);
    putpixel(x1,y1,14);
    symmetry();
  }
  /* clean up */
  getch();
  closegraph();
  return 0;
}
void symmetry()
{
        x2 = floor(2*x0 - x1);
    y2 = y1;
    putpixel(x2,y2,15);
    x3 = x2;
    y3 = 2*y0-y1;
    putpixel(x3,y3,14);
    x4 = x1;
    y4 = y3;
    putpixel(x4,y4,14);
    x5 = floor(y1 - y0 +x0);
    y5 = floor(x1 - x0 +y0);
    putpixel(x5,y5,14);
    x6= y2 -y0+x0;
    y6 = x2 -x0+y0;
    putpixel(x6,y6,15);
    x7 = y3-y0+x0;
    y7 = x3-x0+y0;
    putpixel(x7,y7,15);
    x8 = y4-y0+x0;
    y8 = x4-x0+y0;
    putpixel(x8,y8,15);
  }
```



شكل (12-25)

في هذه الخوارزمية يتمّ حساب نقطة جديدة على محيط الدائرة باستخدام إحداثيات النقطة السابقة لها فمثلاً إذا كانت (x_1, y_1) نقطة محسوبة على محيط الدائرة وكانت (x_2, y_2) هي النقطة التي تليها وتقع أيضاً على المحيط والتي تبعد عنها بزاوية مقدارها $d\theta$ فيتمّ حسابها كما يلي:

$$x_2 = r\cos(\theta + d\theta)$$
$$y_2 = r\sin(\theta + d\theta) \qquad (6-12)$$

من العلاقتين (12-6) وباستخدام حساب المثلثات نحصل على:

$$x_2 = r\cos\theta\cos d\theta - r\sin\theta\sin d\theta$$
$$y_2 = r\sin\theta\cos d\theta + r\cos\theta\sin d\theta \qquad (7-12)$$

وباستخدام العلاقة (12-5) نحصل على:

$$x_2 = x_1\cos d\theta - y_1\sin d\theta$$
$$y_2 = y_1\cos d\theta + x_1\sin d\theta \qquad (8-12)$$

حيث $d\theta$ هي مقدار الزيادة بالزاوية θ ويتمّ اختيارها حسب طلب المستخدم فكلما صغرت $d\theta$ أدى ذلك إلى زيادة عدد النقط على محيط الدائرة وبالتالي الحصول على رسم أوضح وأعلى دقة.

هذا في حالة كون النقطة (0,0) هي نقطة مركز الدائرة، وبما أن النقطة (0,0) تقع في أقصى الزاوية العليا في الشاشة فهذا يجعلها غير مناسبة عند العرض لذلك

فسنختار نقطة أخرى تقع في وسط الشاشة ولتكن إحداثياتها (x_0, y_0) مثلاً، لتكون أكثر ملائمة لعرض الشكل بأكمله، لذلك فإن:

$$x_1 = x_0 + r\cos\theta$$
$$y_1 = y_0 + r\sin\theta \qquad (9\text{-}12)$$

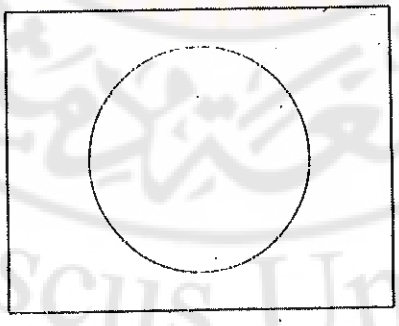
عندما تكون (x_0, y_0) نقطة مركز الدائرة بدلاً من النقطة $(0,0)$ فتصبح العلاقات كما يلي:

$$x_2 = x_0 + (r\cos\theta\cos d\theta - r\sin\theta\sin d\theta) \qquad (10\text{-}12)$$
$$y_2 = y_0 + (r\sin\theta\cos d\theta + r\cos\theta\sin d\theta)$$

ويمكن تبسيط هذه العلاقات وكتابتها بدلالة $(\sin d\theta, \cos d\theta)$ دون الحاجة إلى $\cos\theta$ و $\sin\theta$ كما يلي:

$$x_2 = x_0 + (x_1\cos d\theta - y_1\sin d\theta)$$
$$y_2 = y_0 + (y_1\cos d\theta + x_1\sin d\theta) \qquad (11\text{-}12)$$

تمتاز هذه الطريقة عن سابقتها بأن حسابات الدوال المثلثية تتمّ مرة واحدة فقط لجميع النقاط، بينما تكرر حساباتها بعدد النقاط في الطرق السابقة وفي هذا اختصار كبير للوقت.



شكل (12-26)

⇦ الدالة circle( )

صيغتها العامة كما يلي :

```
void far circle(int x , int y , int radius);
```

ترسم دائرة، مركزها النقطة (x,y) ونصف قطرها يمرر في البارامتر Radius.

مثال:

```
circle(350,245,50);
```

تمَّ إعطاء البارامتر هنا قيماً ثابتة، ويمكن أن تكون قيمها على شكل متغيرات من النوع نفسه أو علاقة رياضيات تعطي نتائج من الأنواع نفسها.

مثال:

البرنامج التالي ب يستدعي الدالة **circle** ويرسم عشر دوائر متمركزة .

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy, radius = 100;
 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");
 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) {  /* an error occurred */
   printf("Graphics error: %s\n", grapherrormsg(errorcode));
   printf("Press any key to halt:");
   getch();
   exit(1);              /* terminate with an error code */
 }
 midx = getmaxx() / 2;
 midy = getmaxy() / 2;
```

```
  setcolor(getmaxcolor());
  /* draw the circle */
  for (int i=0 ; i < 10 ; i++)
  {
        circle(midx, midy, radius);
    radius += 10;
  }
  /* clean up */
  getch();
  closegraph();
  return 0;
}
```

شكل (12-27)

مثال:

اكتب برنامجاً لرسم دائرة مركزها (100، 100) ونصف قطرها 100.

```
#include <iostream.h>
# include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>
void main ()
{
        int gd = DETECT,gm,er;
        inigraph(&gd,&gm," ");
```

```
er = graphresult();
if(er != 0)
{
        printf("the error is%d, graph error misg(er)");
        printf("press any key %d");
        exit(1);
}
putpixel(200,200);
float x,y;
for(int i=0;i<360;i++)
{
        x=float(200+100*cos(i*3.14/180));
        y=float(200+100*sin(i*3.14/180));
        putpixel(float(x),float(y));
}
getch();
closegraph();
}
```

8- رسم الأقواس والقطوع الناقصة:

يمكن استخدام أي من الخوارزميات التي تمّ توضيحها سابقاً لرسم الدائرة، أو لرسم جزء من الدائرة وهو القوس.

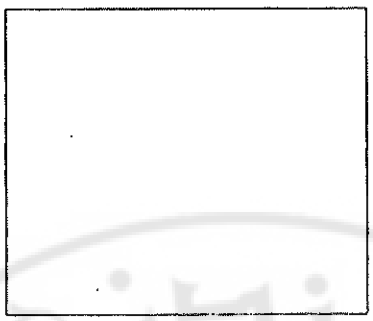
مثال:

يستخدم البرنامج التالي خوارزمية تقريب متعدد الأضلاع إلى دائرة بزيادة أضلاعه مقارنة بنصف القطر.

a: نصف قطر الدائرة.

$x0, y0$: إحداثيات مركز الدائرة.

n : عدد أضلاع المضلع التي سيتمّ رسم القوس بها



شكل (12-28)

```
#include <graphics.h>
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <dos.h>
int main(void)
{
        clrscr();
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int x0,y0,n,a,colr;
  float xi,yi,pi = 3.14;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
  }
  cout << "Please enter the radius :";
  cin >> a;
        cout << "\n\n Please enter the center : \n x0 =";
```

```
    cin >> x0;
    cout << "y0 =";
    cin >> y0;
          cout << "Please Enter the color that you want:";
    cin >> colr;
    n = 50;
    cleardevice();
    for (float i = 0; i <= n-39; i+=0.1)
    {
          xi = x0 + a*cos(2*pi*i/n);
          yi = y0 + a*sin(2*pi*i/n);
      sound (n+1-i);
      dclay(3);
      putpixel(xi,yi,colr);
    }
    setcolor(2);
    outtextxy(getmaxx()/2,getmaxy()/2,"Press any key to quit");
    nosound();
    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

الدالة arc ( ) ←

صيغتها العامة كما يلي:

```
void far arc(int x, int y, int stangle, int endangle, int radius);
```

ترسم هذا الدالة قوس دائرة حيث:

x,y : إحداثيات نقطة مركز الدائرة.

stangle : زاوية البداية بالدرجات.

endangle : زاوية النهاية بالدرجات.

radius : نصف القطر.

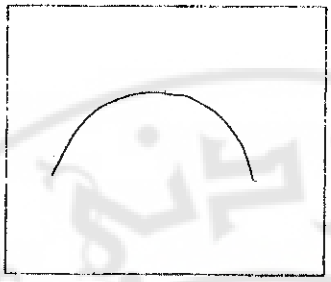
مثال:

اكتب برنامجاً لرسم قوس من دائرة مركزها النقطة (250,250) ونصف قطرها 100 ابتداءً من الزاوية 90 وانتهاءً بالزاوية 65 درجة نكتب

```
arc(250,250,90,65,100);
```

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int midx, midy;
  int stangle = 0, endangle = 180;
  int radius = 100;

  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");

  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
  }
  midx = getmaxx() / 2;
  midy = getmaxy() / 2;
  setcolor(getmaxcolor());
  /* draw arc */
  arc(midx, midy, stangle, endangle, radius);
  /* clean up */
  getch();
```

```
  closegraph();
  return 0;
}
```



شكل (12-29)

9- رسم قطاع دائرة Drawing piesl ce

الدالة pielice( )

صيغتها العامة كما يلي:

```
void far pieslice(int x, int y, int stangle, int endangle, int radius);
```

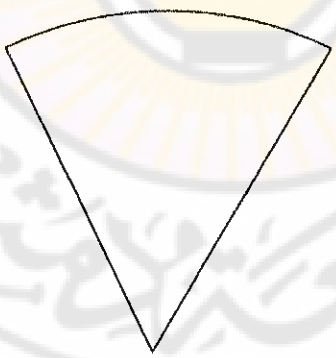
في البرنامج التالي يستخدم طريقة التمثيل التزايدي لرسم الدائرة ، حيث تمّ رسم قوس دائرة محصور بين زاوية البداية start وزاوية النهاية ending واللتين أعطيتا القيمتان 45 و 135ثم تمّ إيصال نهايتي القوس بمركز الدائرة لرسم قطاع دائري.

مثال:

يرسم البرنامج التالي قطاعاً دائرياً بين الزاويتين 45 و 135 وبنصف قطر مقداره 150 ومركزه في وسط الشاشة المستخدمة.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
```

```
int gdriver = DETECT, gmode, errorcode;
int midx, midy;
int stangle = 45, endangle = 135, radius = 150;
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");
/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk)   /* an error occurred */
{
  printf("Graphics error: %s\n", grapherrormsg(errorcode));
  printf("Press any key to halt:");
  getch();
  exit(1);            /* terminate with an error code */
}
midx = getmaxx() / 2;
midy = getmaxy() / 2;
/* set fill style and draw a pie slice */
      //  setfillstyle(EMPTY_FILL, getmaxcolor());
pieslice(midx, midy, stangle, endangle, radius);
/* clean up */
getch();
closegraph();
return 0;

  }
```



شكل (12-30)

10- رسم القطوع الناقصة وقطاعاتها Drawing ellipse and pieslice

لرسم شكل القطاع الناقص نستخدم العلاقات التالية:

$$x = x_0 + a\cos\theta$$
$$y = y_0 + b\sin\theta \qquad (12-12)$$

حيث:

x_0, y_0 هي إحداثيات نقطة مركز القطع الناقص.

a : نصف القطر الأفقي (السيني).

b : نصف القطر العمودي (الصادي).

إن حساب عدة نقاط لشكل القطع الناقص بهذه الطريقة يستغرق وقتاً طويلاً نسبياً بسبب حسابات $\sin\theta$ و $\cos\theta$ حيث θ تتغير من الصفر إلى 2π فإذا كان التغير بالزاوية صغير ولكل زاوية يتمّ حساب نقطة على محيط الشكل القطع الناقص، وعليه يمكن استخدام أسلوب التمثيل التزايدي الذي استخدمناه سابقاً لرسم الدائرة. ولإيجاد أي نقطة على المحيط بدلالة النقطة السابقة لها يتمّ من خلال العلاقات التالية:

$$x_2 = x_1 + a\cos(\theta + d\theta)$$
$$y_2 = y_0 + b\sin(\theta + d\theta) \qquad (13-12)$$

شكل (12-31)

مثال:

البرنامج التالي لرسم القطع الناقص باستخدام العلاقات (12-12).

```
#include <graphics.h>
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <dos.h>
int main(void)
{
        clrscr();
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int x0,y0,r,a,b,color;
  float x1,y1,pi = 3.14;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
  }

  cout << "Please enter the A radius :\n";
  cin >> a;
  cout << "Please enter the B radius :\n";
  cin >> b;
        cout << "\n\n Please enter the center : \n x0 =";
  cin >> x0;
  cout << "y0 =";
  cin >> y0;
        cout << "Please Enter the color that you want:";
  cin >> color;
```

```
    r = 360;
    cleardevice();
    for (float r = 0; r <= 360; r++)
    {
          x1 = x0 + a*cos(r*pi/180);
          y1 = y0 + b*sin(r*pi/180);
      putpixel(x1,y1,color);
    }
    outtextxy(getmaxx()/2,getmaxy()/2,"Press any key to quit");

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

وبتطبيق قواعد المثلثات تصبح العلاقات كما يلي:

$$x_2 = x_1 \cos(d\theta) - (a/b) y_1 \sin(d\theta)$$
$$y_2 = y_1 \cos(d\theta) - (b/a) x_1 \sin(d\theta) \qquad (14-12)$$

حيث تستخدم $d\theta$ في الحسابات والتي يتمّ حسابها مرة واحدة فقط بدلاً من θ.

شكل (12-32)

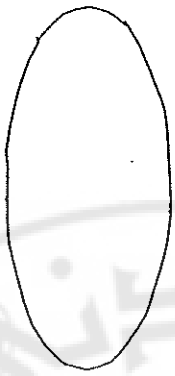
الدالة ellipse ( )

البرنامج التالي يرسم قطع ناقص باستخدام الدالة ellipse( ).

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
```

```
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int midx, midy;
  int stangle = 0, endangle = 360;
  int xradius = 150, yradius = 80;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);              /* terminate with an error code */
  }
  midx = getmaxx() / 2;
  midy = getmaxy() / 2;
  setcolor(getmaxcolor());
  /* draw ellipse */
  ellipse(midx, midy, stangle, endangle, xradius, yradius);
  /* clean up */
  getch();
  closegraph();
  return 0;
}
```

بادل بين قيمتي xradius,yradius فينتج شكل قطع ناقص عمودي كما في الشكل.



شكل (12-33)

يمكن استخدام طرق رسم شكل القطع الناقص الأنفة الذكر لرسم جزء من شكل القطع الناقص فقط أي قطاع قطع ناقص.

☜ الدالة sector( )

لرسم قطاع شكل قطع ناقص صيغتها العامة:

void sector (x,y : integer; stangle, enangel,Xradius,yradius:word);

ترسم الدالة sector( ) قطاع شكل قطع ناقص مطلي باللون والتشكيلة قيد الاستخدام حيث:

x,y : تمثل إحداثيات نقطة المركز لشكل القطع الناقص.

stangel : زاوية البداية بالدرجات.

endangel : زاوية النهاية بالدرجات.

Xradius : نصف القطر الأفقي لشكل القطع الناقص.

Yradius : نصف قطر العمودي لشكل القطع الناقص.

مثال:

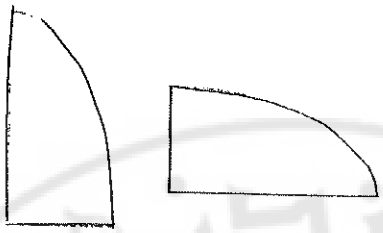
sector (350,250,90,0,60,100);

البرنامج التالي يرسم قطاع شكل قطع ناقص بصورة أفقية.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int midx, midy, i;
  int stangle = 45, endangle = 90;
  int xrad = 200, yrad = 100;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);              /* terminate with an error code */
  }
  midx = getmaxx() / 2;
  midy = getmaxy() / 2 + 150;
  /* loop through the fill patterns */
  for (i=EMPTY_FILL; i<USER_FILL; i++) {
    /* set the fill style */
    setfillstyle(i, getmaxcolor());
    /* draw the sector slice */
    sector(midx, midy, stangle, endangle, xrad, yrad);
    getch();
  }
  /* clean up */
  closegraph();
  return 0; }
```

ويمكن استبدال أنصاف الأقطار مع بعضها للحصول على القطاع نفسه ولكن

بصورة عمودية كما في الشكل (31.3).



شكل (12-34)

تحتوي وحدة الرسم البياني على أكثر من عشرين دالة قياسية لتغير اللون والتشكيلة. الجدول التالي يبين التشكيلات المتاحة في النظام وهي 13 تشكيلة، حيث يوضح الجدول رقم التشكيلة وأسمها.

EMPTY_FILL	0	Fill with background color
SOLID_FILL	1	Solid fill
LINE_FILL	2	Fill with ---
LTSLASH_FILL	3	Fill with ///
SLASH_FILL	4	Fill with ///, thick lines
BKSLASH_FILL	5	Fill with \\\, thick lines
LTBKSLASH_FILL	6	Fill with \\\
HATCH_FILL	7	Light hatch fill
XHATCH_FILL	8	Heavy crosshatch fill
INTERLEAVE_FILL	9	Interleaving line fill
WIDE_DOT_FILL	10	Widely spaced dot fill
CLOSE_DOT_FILL	11	Closely spaced dot fill
USER_FILL	12	User-defined fill pattern

الجدول (12-3)

الجدول التالي يبين الألوان المتوفرة في النظام والكود المقابل لكل منها.

0	BLACK	8	DARKGRAY
1	BLUE	9	LIGHTBLUE
2	GREEN	10	LIGHTGREEN
3	CYAN	11	LIGHTCYAN
4	RED	12	LIGHTRED
5	MAGENTA	13	LIGHTMAGENTA
6	BROWN	14	YELLOW
7	LIGHTGRAY	15	WHITE

الجدول (12-4)

11- روتينات التشكيل والتلوين

☜ الدالة :setfillstyle( )

صيغتها العامة هي:

```
void far setfillstyle(int pattern, int color);
```

تعيد هذه الدالة التشكيلة واللون لملء الأشكال المغلقة بالتشكيلات التي تستخدم من قبل الدوال fillpoly( ) و bar( ) و bar3d( ) و pieslice( ): يتمّ إعطاء اسم التشكيلة أو رقمها واسم اللون أو رقمه في البارامترات attern و color على التوالي. إذا تمّ الاستدعاء بقيم مقبولة سيعيد الدالة القيمة 11- وتبقى القيمة الجارية للتشكيلة واللون بدون تغيير.

القيمة الاعتباطية للتشكيلة هي solidfill واللون الاعتباطي هو maxcolor أي أعلى قيمة للألوان.

☜ الدالة :getfillsettings( )

صيغتها العامة هي:

void far getfillsettings(struct fillsettingstype far *fillinfo);

تعيد هذه الدالة تشكيلة الملء الجارية واللون كما تمّ إعدادهما بوساطة الدالة setfillpattern( ). إذا كانت قيمة التشكيلة تساوي userfill فاستخدم الدالة getfillpattern( ) للحصول على التشكيلة المعرفة من قبل المستخدم والتي تمّ اختيارها.

مثال:

البرنامج التالي يستدعي الدالة setfillstyle( ) والدالة bar3d( ) لملئه بالتشكيلات المتاحة والمعرفة في البرنامج.

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
/* the names of the fill styles supported */
char *fname[] = { "EMPTY_FILL", "SOLID_FILL",
"LINE_FILL", "LTSLASH_FILL", "SLASH_FILL",
"BKSLASH_FILL", "LTBKSLASH_FILL", "HATCH_FILL",
"XHATCH_FILL", "INTERLEAVE_FILL",
"WIDE_DOT_FILL", "CLOSE_DOT_FILL", "USER_FILL" };
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int style, midx, midy;
  char stylestr[40];
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
```

```
    getch();
    exit(1);            /* terminate with an error code */
  }

  midx = getmaxx() / 2;
  midy = getmaxy() / 2;
  for (style = EMPTY_FILL; style < USER_FILL; style++) {
    /* select the fill style */
    setfillstyle(style, getmaxcolor());
    /* convert style into a string */
    strcpy(stylestr, fname[style]);
    /* fill a bar */
    bar3d(0, 0, midx-10, midy, 0, 0);
    /* output a message */
    outtextxy(midx, midy, stylestr);
    /* wait for a key */
    getch();
    cleardevice();
  }
  /* clean up */
  getch();
  closegraph();
  return 0;
}
```

: floodfill( ) الدالة ←

صيغتها العامة وهي

```
void far floodfill(int x, int y, int border);
```

تُملأ مساحة الشكل المغلق باللون والتشكيلة الجاريين حيث تمثل x,y إحداثيات أي نقطة تقع ضمن الشكل المغلق وتسمى البذرة أما border فيمثل لون الخط المحيط بالشكل. يتمّ تحديد اللون والتشكيلة المطلوبة بوساطة الدالة setfillStyle( ) أو setfillpattern( ). إذا كانت البذرة تقع خارج حدود الشكل المغلق أو إن الشكل لم يكن مغلقاً تماماً فإن اللون والتشكيلة المحددان سينتشر خارج الشكل.

➪ الدالة setfillpattern( ):

صيغته العامة هي:

```
void far setfillpattern(char far *upattern, int color);
```

حيث upattern* تشير إلى سلسلة من 8 بايت، كل بايت يخص 8 نقاط في التشكيلة المعرفة من قبل المستخدم. كلما يعطى البت قيمة 1 في التشكيلة، يتمّ رسم النقطة المقابلة. فمثلاً يمكن تعريف التشكيلة التالية والتي تمثل لوحة الشطرنج.

```
char chekerborder[8]={
        0xAA, /* 10101010 = - - - - */
        0x55, /*01010101 = - - - -*/
        0xAA, /* 10101010 = - - - - */
        0x55, /*01010101 = - - - -*/
        0xAA, /* 10101010 = - - - - */
        0x55, /*01010101 = - - - -*/
        0xAA, /* 10101010 = - - - - */
        0x55, /*01010101 = - - - -*/
};
```

➪ الدالة getfillpattern( ):

صيغته العامة هي:

```
void far getfillpattern(char far *pattern);
```

تعيد الدالة getfillpattern( ) اللون والتشكيلة الجارية المستخدمة في ملء الأشكال المغلقة والتي تمّ إعدادها بالدالة setfillpattern( ). إذا لم يتمّ استدعاؤها setfillpattern( ) فستعيد getfillpattern( ) مصفوفة مملوءة بالقيمة $FF. حيث fillpattern( ) معرفة مسبقاً في النظام.

مثال:

البرنامج التالي يستدعي الدالتين getfillpattern( ) و setfillpattern( ) بعد أن يعرف تشكيلة خاصة به في المتغير Pattern والذي هو عبارة عن مصفوفة من ثمانية رموز، حيث تمَّ استدعاء الدالة لملئه بهذه التشكيلة ثم تمَّ تغيير بعض رموز التشكيلة واستدعاء الدالة bar( ) مرة أخرى لملئه بالتشكيلة الجديدة.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int maxx, maxy;
  char pattern[8] = {0x00, 0x70, 0x20, 0x27, 0x25, 0x27, 0x04, 0x04};
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);              /* terminate with an error code */
  }
  maxx = getmaxx();
  maxy = getmaxy();
  setcolor(getmaxcolor());
  /* select a user-defined fill pattern */
  setfillpattern(pattern, getmaxcolor());
  /* fill the screen with the pattern */
  bar(0, 0, maxx, maxy);
  getch();
  /* get the current user-defined fill pattern */
  getfillpattern(pattern);
  /* alter the pattern we grabbed */
```

```
    pattern[4] -= 1;
    pattern[5] -= 3;
    pattern[6] += 3;
    pattern[7] -= 4;
    /* select our new pattern */
    setfillpattern(pattern, getmaxcolor());
    /* fill the screen with the new pattern */
    bar(0, 0, maxx, maxy);
    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

الدالة fillpoly( ):

ترسم الدالة **fillpoly( )** شكل متعدد أضلاع باستخدام شكل الخط ولونه الجاريين ثم يلونه بالتشكيلة واللون المحدد بواسطة **setfillstyle( )** أو **setfillpattern( )**، أما إذا لم يتمّ استدعاء أي من هذين الدالتين قبله فسيتمّ ملء متعدد الأضلاع باللون والتشكيلة الاعتباطية وهما اللون الأبيض والتشكيلة solidfill.

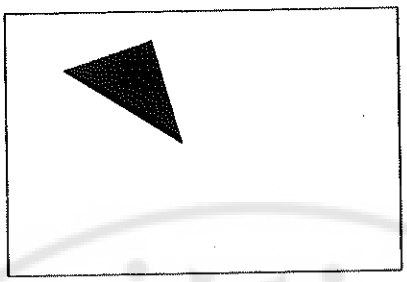
صيغتها العامة كما يلي:

```
void far fillpoly(int numpoints, int far *polypoints);
```

حيث:

numpoints : تمثل عدد نقاط رؤوس المضلع.

PolyPoints* : تمثل سلسلة من النقاط المخزنة في مصفوفة حجمها (numPoints x 2) من القيم الصحيحة كل زوج من القيم يمثل إحداثيات x و y لكل نقطة من نقاط رؤوس المضلع. لرسم شكل مغلق ذي N من الرؤوس يجب تمرير N من النقاط أي (N*2) من القيم، حيث تقوم الدالة بغلق المضلع أوتوماتيكياً.



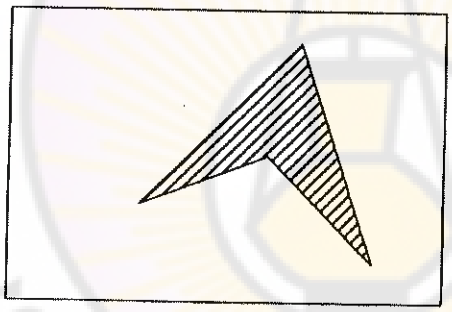
شكل (12-35)

مثال:

البرنامج التالي يرسم متعدد الأضلاع الموضح في شكل (12-35) ويملأه بالتشكيلات المتاحة في النظام حيث نحصل على تشكيلة مختلفة كلما ضغطنا المفتاح Enter.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int i, maxx, maxy;
  /* our polygon array */
  int poly[8];
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
  }
  maxx = getmaxx();
  maxy = getmaxy();
  poly[0] = 20;          /* first vertex */
  poly[1] = maxy / 2;
  poly[2] = maxx - 20;     /* second vertex */
```

```
  poly[3] = 20;
  poly[4] = maxx - 50;     /* third vertex */
  poly[5] = maxy - 20;
  poly[6] = maxx / 2;      /* fourth, fillpoly automatically */
  poly[7] = maxy / 2;      /* closes the polygon */
  /* loop through the fill patterns */
  for (i=EMPTY_FILL; i<USER_FILL; i++) {
    /* set fill pattern */
    setfillstyle(i, getmaxcolor());
    /* draw a filled polygon */
    fillpoly(4, poly);
    getch();
  }
  /* clean up */
  closegraph();
  return 0;
}
```



شكل (12-36)

☜ الدالة fillellipse( ):

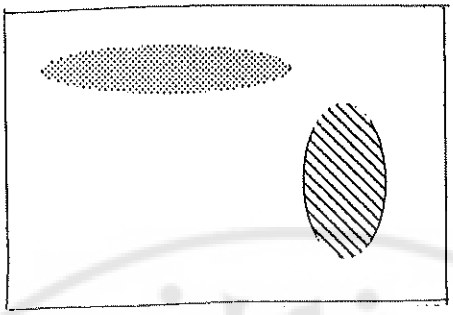
صيغتها العامة هي:

```
void far fillellipse(int x, int y, int xradius, int yradius);
```

ترسم شكل قطع ناقص وملئه باللون والتشكيلة الجارية، حيث:

x,y : هي إحداثيات نقطة المركز.

xradius, yradius : هما القطران الأفقي والعمودي للقطع الناقص.



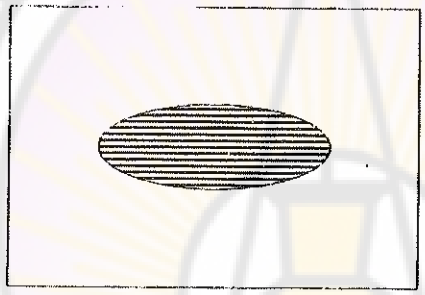
شكل (12-37)

مثال:

البرنامج التالي يستدعي الدالة **fillellipse( )** ليرسم شكل القطع الناقص الموضح في الشكل السابق ويستخدم التشكيلات والألوان المتاحة لملئه حيث يتمّ ملأه بلون وتشكيلة مختلفة عند كل ضغطة على أي مفتاح من لوحة المفاتيح.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int midx, midy, i;
  int xradius = 100, yradius = 50;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);              /* terminate with an error code */
  }
```

```
    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    /* loop through the fill patterns */
    for (i = EMPTY_FILL; i < USER_FILL; i++) {
      /* set fill pattern */
      setfillstyle(i, getmaxcolor());
      /* draw a filled ellipse */
      fillellipse(midx, midy, xradius, yradius);
      getch();
    }
    /* clean up */
    closegraph();
    return 0;
}
```



شكل (12-38)

- الدالة getpalette( ):

صيغتها العامة هي:

```
void far getpalette(struct palettetype far *palette);
```

تعيد هذه الدالة القيمة الحالية لحجم الألوان والقيم الرقمية لألوانها في حقلي المتغير palette حيث هو من نوع السجل (struct في لغة C++) وهو معرف مسبقاً في النظام.

- الدالة setcolor( ):

صيغتها العامة هي:

void far setcolor(int color);

تحدد لون الرسم البياني الحالي بالقيمة المعطاة في البارامتر color على شرط أن يكون هذا اللون هو أحد الألوان الموجودة في لوحة الألوان، حيث يمكن استدعاء هذه الدالة.

- الدالة **getmaxcolor( )** :

صيغتها العامة كما يلي:

int far getmaxcolor(void);

تعيد هذه الدالة أعلى كود للألوان المتوفرة في لوحة الألوان المستخدمة.

color := getmaxcolor;

- الدالة **getbkcolor( )**:

صيغتها العامة كما يلي:

int far getbkcolor(void);

تعيد هذه الدالة لون خلفية الشاشة الجاري، حيث تتراوح ألوان الخلفية في المدى من 0 إلى 15 اعتماداً على سواقة الرسم قيد الاستخدام ونمط الرسم.

- الدالة **setplaette( )**:

صيغتها العامة كما يلي:

void far setpalette(int colornum, int color);

يغير اللون colornum في لوحة الألوان إلى color.

مثال:

البرنامج التالي يستدعي الدالة **setpalette( )**.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
```

```
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int color, maxcolor, ht;
  int y = 10;
  char msg[80];
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);              /* terminate with an error code */
  }
  maxcolor = getmaxcolor();
  ht = 2 * textheight("W");
  /* display the default colors */
  for (color=1; color<=maxcolor; color++)
  {
    setcolor(color);
    sprintf(msg, "Color: %d", color);
    outtextxy(1, y, msg);
    y += ht;
  }
  /* wait for a key */
  getch();
  /* black out the colors one by one */
  for (color=1; color<=maxcolor; color++)
  {
    setpalette(color, BLACK);
    getch();
  }
  /* clean up */
  closegraph();
  return 0; }
```

☜ الدالة getdefaultpalette( ):

صيغتها العامة كما يلي:

```
Procedure getDefualtpallet (var palette : paletteType);
void far getpalette(struct palettetype far *palette);
```

تعيد هذه الدالة بيانات تخص الحجم واللون للوحة الألوان المتاحة، تخزن النتائج في المتغير Palette الذي هو من النوع PaletteType المعرف مسبقاً في النظام.

مثال:

البرنامج التالي يستدعي الدالة getdefualtPalette( ).

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  /* far pointer to palette structure */
  struct palettetype far *pal = NULL;
  int i;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);              /* terminate with an error code */
  }
  /* return a pointer to the default palette */
  pal = getdefaultpalette();
  for (i=0; i<pal->size; i++) {
    printf("colors[%d] = %d\n", i, pal->colors[i]);
```

```
    getch();
  }
  /* clean up */
  getch();
  closegraph();
  return 0;
}
```

- الدالة setlinestyle( ):

صيغتها العامة كما يلي:

void far setlinestyle(int linestyle, unsigned upattern, int thickness);

يستخدم هذا الدالة لتحديد شكل الخط المطلوب لرسم الأشكال بواسطة الدالة line( )، drawpoly( )، rectangle( )، lineto( ) وذلك بإعطاء قيم للبارامترات:

lincstylc: ويمثل أسلوب رسم الخط ويمكن أن يأخذ أي قيمة صحيحة بين 0 و 4 كما موضح لأدناه:

Name	Value	Description
SOLID_LINE	0	Solid line
DOTTED_LINE	1	Dotted line
CENTER_LINE	2	Centered line
DASHED_LINE	3	Dashed line
USERBIT_LINE	4	User-defined line style

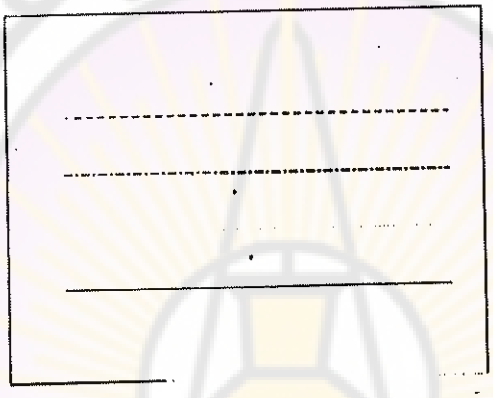
Pattern: يمثل شكل الخط ويمكن أن يأخذ أي قيمة صحيحة. يظهر تأثيرها هذا البارمتر فقط عندما تسند القيمة الثابتة UserBitLn إلى البارامتر lineSty;e عدا ذلك فليس له أي تأثير.

Thickness : يمثل سمك الخط المستقيم ويمكن أن يأخذ أحد القيمتين 1، 3 حيث:

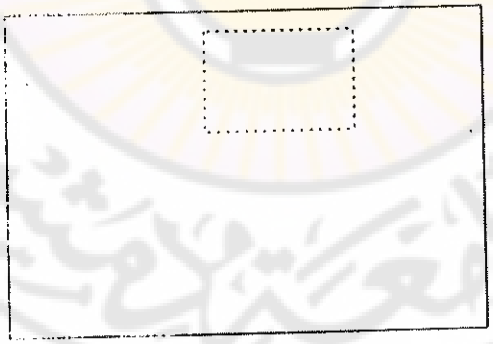
Name	Value	Description
NORM_WIDTH	1	1pixel wide
THICK_WIDTH	3	3pixels wide

1 يمثل السمك العادي، و 3 يمثل الخط السميك.

تستخدم هذه الدوال مع الدوال **line( ), lineto( ), rectangle( ), drawpoly( )** و **fillpoly( )**. الدوال pieslice, arc , circle تستخدم دائماً الخط الصلد solidLn ولا تتأثر بشكل الخط الناتج عن هذا الدالة .



شكل (12-39)



شكل (12-40)

مثال:

البرنامج التالي يستخدم الدالة **setLinestyle( )**.

```
#include <graphics.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
/* the names of the line styles supported */
char *lname[] = { "SOLID_LINE",
                  "DOTTED_LINE",
                 "CENTER_LINE",
                 "DASHED_LINE",
                 "USERBIT_LINE"
               };
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int style, midx, midy, userpat;
  char stylestr[40];
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
  }
  midx = getmaxx() / 2;
  midy = getmaxy() / 2;
  /* a user-defined line pattern */
  /* binary: "0000000000000001" */
  userpat = 1;
  for (style=SOLID_LINE; style<=USERBIT_LINE; style++)
  {
    /* select the line style */
    setlinestyle(style, userpat, 1);
    /* convert style into a string */
    strcpy(stylestr, lname[style]);
```

```
    /* draw a line */
    line(0, 0, midx-10, midy);
    /* draw a rectangle */
    rectangle(0, 0, getmaxx(), getmaxy());
    /* output a message */
    outtextxy(midx, midy, stylestr);
    /* wait for a key */
    getch();
    cleardevice();
  }
  /* clean up */
  closegraph();
  return 0;
}
```

- الدالة **setalpalette( ):**

صيغتها العامة كما يلي:

void far setallpalette(struct palettetype far *palette);

تغير كل الألوان في لوحة الألوان في وقت واحد، لاحظ أن المتغير Palette لم يحدد نوعه لكي تتمكن الدالة من استيعاب أنماط رسوم مختلفة. إن أي تغييرات تمت على اللوحة ستتمكن من مشاهدتها على الفور على الشاشة. عند تمرير أي قيم غير صحيحة لهذا الدالة ، ستعيد graphresult القيمة 11 (geerror).

مثال:

البرنامج التالي يستدعي الدالة ويعرض الألوان الموجودة في لوحة الألوان الأصلية على شكل نص يتكرر داخل دورة حيث يكتب النص باللون المعني والمبين رقمه في النص. ثم يغير ألوان اللوحة إلى اللون الأسود بالتدريج واحداً عند كل كبسة لأحد المفاتيح ثم يسترجع اللوحة القديمة.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
```

```
int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  struct palettetype pal;
  int color, maxcolor, ht;
  int y = 10;
  char msg[80];
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)  /* an error occurred */
  {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);              /* terminate with an error code */
  }
  maxcolor = getmaxcolor();
  ht = 2 * texthcight("W");
  /* grab a copy of the palette */
  getpalette(&pal);
  /* display the default palette colors */
  for (color=1; color<=maxcolor; color++) {
    setcolor(color);
    sprintf(msg, "Color: %d", color);
    outtextxy(1, y, msg);
    y += ht;
  }
  /* wait for a key */
  getch();
  /* black out the colors one by one */
  for (color=1; color<=maxcolor; color++) {
    setpalette(color, BLACK);
    getch();
  }
  /* restore the palette colors */
  setallpalette(&pal);
  /* clean up */
```

```
  getch();
  closegraph();
  return 0;
}
```

الدالة setrgbpalette( ):

صيغتها العامة كما يلي:

void far setrgbpalette(int colornum, int red, int green, int blue);

وهو تعرف ألوان كرت الرسم لجهاز IBM-8514، البارامتر colonnum يحدد أحد مدخلات لوحة الألوان المراد تحميله أما **red ,green ,blue** في تعرف مركبات الألوان للبارامتر colornum.

يحرر من قيود لوحة الألوان، ويعطي الحرية للمستخدم كي يصنع الألوان بنفسه (أي خلق لوحة ألوان خاصة به).

هذه الدالة تعمل فقط على IBM-8514 وسواقات VGA. المتغير colorNum، يكون في المدى 0.. 255 للسواقة IBM-8514. وللسواقة VGA في نمط اللون 256k يكون في الحدود 0..15.

مثال:

البرنامج التالي يستدعي هذه لدالة ويقسم شاشة الرسم إلى مستطيلات عرضية باستخدام الدالة **bar( )** ويلونه بالألوان التي يحصل عليها من الاستدعاء.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* select driver and mode that supports use of setrgbpalette */
  int gdriver = VGA, gmode = VGAHI, errorcode;
  struct palettetype pal;
  int i, ht, y, xmax;
```

```
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  {  /* an error occurred */
     printf("Graphics error: %s\n", grapherrormsg(errorcode));
     printf("Press any key to halt:");
     getch();
     exit(1);             /* terminate with an error code */
  }
  /* grab a copy of the palette */
  getpalette(&pal);
  /* create gray scale */
  for (i=0; i<pal.size; i++)
     setrgbpalette(pal.colors[i], i*4, i*4, i*4);
  /* display the gray scale */
  ht = getmaxy() / 16;
  xmax = getmaxx();
  y = 0;
  for (i=0; i<pal.size; i++)
  {

     setfillstyle(SOLID_FILL, i);
     bar(0, y, xmax, y+ht);
     y += ht;
  }
  /* clean up */
  getch();
  closegraph();
  return 0;
}
```

أن الأشكال الهندسية كالخطوط، متعددات الأضلاع، الأقواس والقطوع المخروطية والتي جميعها تتكون من عدد غير محدود من أشكال النقط، يمكن أن تعرف بمجموعة محددة من النقاط. يمكن إجراء عملية التحويل على أي شكل هندسي بتطبيق دالة مثل F على كل نقطة من نقاطه المحددة، ذلك لأن تطبيق مثل هذه الدالة على عدد غير

محدود من النقاط ليس عملياً (غير ممكن)، ومن ثم استخدام هذه النقاط بعد عملية التحويل لتكوين هيئة أخرى للشكل الهندسي.

12- التحويلات الخطية هي أي تركيب من التحويلات التالية:

- الإزاحة translation.

1. التقييس scaling.
2. الدوران rotation.
3. الانعكاس reflection.
4. القص shears.

- إزاحة الأشكال الرسومية Translation of Graphics Objects

لإزاحة أي شكل رسومي، يجب أن تزاح كل نقطة (x,y) من النقاط التي تمثل الشكل إلى الموقع الجديد $(\bar{x},\bar{y})$

$$(x,y)\rightarrow(\bar{x},\bar{y}) \qquad (15-12)$$

حيث:

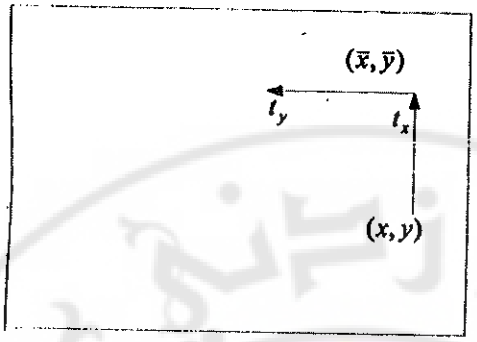
$$\bar{x}=x+t_s \qquad (16-12)$$

$$\bar{y}=y+t_y$$

إذا كانت قيمة t_x موجبة فيتمّ إزاحة النقطة إلى اليمين أما إذا كانت سالبة فيم إزاحة النقطة إلى اليسار، عندما تكون $t_x=0$ فيعني أن النقاط تتحرك موازية المحور الصادي.

إذا كانت t_y موجبة فيتمّ إزاحة النقطة إلى الأسفل أما إذا كانت سالبة فيتمّ إزاحة النقطة نحو الأعلى، وعندما تكون $t_y=0$ ف فهذا يعني أن النقطة تتحرك بموازاة

المحور السيني.



شكل (12-41)

لإزاحة النقطة $p(x,y)$ إلى نقطة الأصل (0,0) فإن ذلك يتمّ بطرح إحداثيات تلك النقطة من نفسها لتكون $p(x-x,y-y)$ ، الدالة (translatepoint) :

مثال:

البرنامج التالي يزيح المربع إلى نقطة الأصل ومن ثم إزاحته من نقطة الأصل إلى موقع جديد بإضافة قيم إحداثيات الموقع الجديد إلى قيم إحداثيات كل نقطة من نقطه.

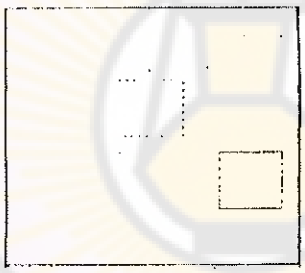
حيث تتمّ الإزاحة من خلال الدالة **translate( )** الذي يتمّ استدعاؤها عدة مرات:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
# include <math.h>
//int px1,px2,px3,px4, py1,py2,py3,py4;
//{=============================================}
void Drawsqure(int px1,int py1,int px2,int py2,int px3,int py3,
                                                    int px4,int py4)
{
   line(px1,py1,px2,py2);
   line(px2,py2,px3,py3);
   line(px3,py3,px4,py4);
   line(px4,py4,px1,py1);
}
```

```
//{=====================================}
void Translat (int flag, int x11,int y11,int & px1,int &py1,int
&px2,int &py2, int &px3,int &py3,int &px4,int &py4)
{
    px2 = px2 - (x11 * flag);
    py2 = py2 - (y11 * flag);
    px3 = px3 - (x11 * flag);
    py3 = py3 - (y11 * flag);
    px4 = px4 - (x11 * flag);
    py4 = py4 - (y11 * flag);
    px1 = px1 - (x11 * flag);
    py1 = py1 - (y11 * flag);
}
//{=====================================}
int main(void)
{
  /* select driver and mode that supports use of setrgbpalette */
  int gdriver = DETECT, gmode , errorcode;
  int x,x1,x2,x3,x4,y, y1,y2,y3,y4;
  float alpha;
  //int px1,px2,px3,px4, py1,py2,py3,py4;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
  }
  x1 = 300;   y1 = 200;
  x2 = 420;   y2 = 200;
  x3 = 420;   y3 = 320;
  x4 = 300;   y4 = 320;
  // { Draw the squre in its orignal position}
    Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
  // {translate the squre to the origion Point (0,0)}
    Translat (1,x1,y1,x1,y1,x2,y2,x3,y3,x4,y4);
```

```
// {redraw the squre inthe origion point(0,0)}
  setlinestyle(1,1,1);
  Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
x = 150;    y = 120;
// {translate the squre to the origion Point (150,120)}
  Translat (-1,x,y,x1,y1,x2,y2,x3,y3,x4,y4);
// {redraw the squre in the new postion}
  setlinestyle(1,1,1);
  Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
/* clean up */
getch();
closegraph();
return 0;
}
```

المتغير flag الوارد في الدالة **translate( )** يأخذ أحد القيمتين 1 أو 1- فعندما تكون قيمته 1 يتمّ إزاحة الشكل من نقطة الأصل، وعندما تكون قيمته 1- فيتمّ إزاحته إلى الموقع الجديد. تمّ إزاحة المربع بدون أي تغيير في الحجم أو الشكل.



شكل (12-42)

- تقييس الأشكال الرسومية Scaling of Graph Objects

نعني بعملية التقييس هي تغير حجم الشكل الرسومي، من ناحية التكبير أو التصغير. كل نقطة $p(x,y)$ من النقاط التي تمثل الشكل يتمّ تحريكها إلى موقع جديد $\bar{p}(\bar{x},\bar{y})$ طبقاً لما يلي:

$$\bar{x} = S_x * x$$
$$\bar{y} = S_y * y \qquad (17-12)$$

حيث S_x, S_y تسمى معامل التقييس Scaling factor، فإذا كان معامل التقييس 1 يعني الحفظ على نفس الحجم وإذا كان أكبر من الواحد يعني زيادة حجم الشكل أما إذا كان هذا العامل أصغر من الواحد فيعني تصغير حجم الشكل.

إذا كانت S_x لا تساوي S_y فهذا يؤدي إلى تشويه الشكل الناتج عن الشكل الأصلي. هذه التحويلات سواء أكانت تكبير أم تصغير للشكل تسمى التحريفات بالنسبة لنقطة الأصل. أما إذا كان التقييس بالنسبة لنقطة أخرى تسمى نقطة الارتكاز (x_p, y_p) غير نقطة الأصل (0,0) فيتمّ التقييس حسب الخطوات التالية:

i. إزاحة الشكل إلى نقطة الأصل نسبة إلى تلك النقطة، حيث تصبح إحداثيات كل نقطة (x, y) من نقاط الشكل $(\bar{x}, \bar{y})$ وكما يلي:

$$\bar{x} = x - x_p$$
$$\bar{y} = y - y_p \qquad (18-12)$$

ii. تقييس النقطة المزاحة حسب العلاقات التالية:

$$\bar{\bar{x}} = \bar{x} \cdot S_x \Rightarrow (x - x_p)S_x$$
$$\bar{\bar{y}} = \bar{y} \cdot S_y \Rightarrow (y - y_p)S_y \qquad (19-12)$$

iii. إعادة كل نقطة تمّ تقييسها إلى موقعها الأصلي بإعادة إزاحتها المقدار نفسه الذي تم إزاحتها به في الخطوة I ولكن بعكس الاتجاه:

$$\bar{\bar{\bar{x}}} = \bar{\bar{x}} + x_p$$
$$\bar{\bar{\bar{y}}} = \bar{\bar{y}} + y_p \qquad (20-12)$$

ومن الخطوات الثلاث السابقة نحصل على ما يلي:

$$\overline{\overline{x}} = (x - x_p)S_x + x_p$$
$$\overline{\overline{y}} = (y - y_p)S_y + y_p \qquad (21-12)$$

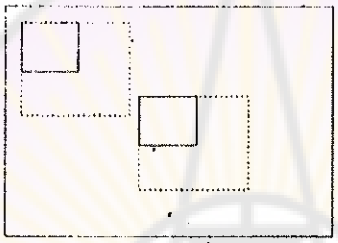
مثال:

البرنامج التالي يقيس المربع الذي رؤوسه النقاط $(x1, y1)$ ، $(x2, y2)$ ، $(x3, y3)$ ، $(x4, y4)$ بالنسبة للنقطة $(x1, y1)$.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
# include <math.h>
//int px1,px2,px3,px4, py1,py2,py3,py4;
//{=====================================}
void Drawsqure(int px1,int py1,int px2,int py2,int px3,int py3,
                                int px4,int py4)
{
   line(px1,py1,px2,py2);
   line(px2,py2,px3,py3);
   line(px3,py3,px4,py4);
   line(px4,py4,px1,py1);
}
//{=====================================}
void Translat (int flag, int x11,int y11,int & px1,int &py1,int
&px2,int &py2,int &px3,int &py3,int &px4,int &py4)
{
   px2 = px2 - (x11 * flag);
   py2 = py2 - (y11 * flag);
   px3 = px3 - (x11 * flag);
   py3 = py3 - (y11 * flag);
   px4 = px4 - (x11 * flag);
   py4 = py4 - (y11 * flag);
   px1 = px1 - (x11 * flag);
   py1 = py1 - (y11 * flag);
}
//{=====================================}
void scale (int sx,int sy, int &x,int & y)
```

```
{
        x = x * sx;
  y = y * sy;
}
//{=====================================}

int main(void)
{
  /* select driver and mode that supports use of setrgbpalette */
  int gdriver = DETECT, gmode , errorcode;
  int x,x1,x2,x3,x4,y, y1,y2,y3,y4;
  //int px1,px2,px3,px4, py1,py2,py3,py4;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);            /* terminate with an error code */
  }
  x1 = 300;   y1 = 200;
  x2 = 420;   y2 = 200;
  x3 = 420;   y3 = 320;
  x4 = 300;   y4 = 320;
  x = 300;              y = 200;
  // { Draw the squre in its orignal position}
    Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
  // {translate the squre to the origion Point (0,0)}
    Translat (1,x,y,x1,y1,x2,y2,x3,y3,x4,y4);
  // {redraw the squre inthe origion point(0,0)}
    Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
  /* {Scale the squre by multiplaying each of its points
    whith the scaling factor of 2}*/
    scale (2,2,x1,y1);
    scale (2,2,x3,y3);
    scale (2,2,x2,y2);
    scale (2,2,x4,y4);
```

```
    setlinestyle(1,1,2);
    Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
  // {translate the squre to the its orignal positon}
    Translat (-1,x,y,x1,y1,x2,y2,x3,y3,x4,y4);
  // {redraw the squre in the new postion}
    setlinestyle(1,1,1);
    Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
  /* clean up */
  getch();
  closegraph();
  return 0;
}
```



شكل (12-43)

- التدوير **Rotation**

تدوير شكل رسومي حول نقطة معينة تسمى نقطة الارتكاز أو محور معين يسمى المركز بزاوية معينة θ ولأجل أن يتمّ التدوير حول هذه النقطة يجب أن يرجع الشكل إلى نقطة الأصل أي أن يزاح الشكل إلى نقطة الأصل بالنسبة لنقطة الارتكاز. ثم يتمّ تدوير الشكل بالنسبة إلى نقطة الأصل وذلك بتدوير كل نقطة من النقاط التي تمثله.

فلو فرضنا أن إحدى نقاطه هي $p(x,y)$ يراد تدويرها بزاوية θ فإن:

$$(x,y) \rightarrow (\bar{x},\bar{y}) \qquad (22\text{-}12)$$

بعد إزاحتها إلى نقطة الأصل حيث:

$$\bar{x} = x - x_p$$
$$\bar{y} = y - y_p \qquad (23-12)$$

ثم يتمّ تدوير $p(\bar{x},\bar{y})$ لتعطينا نقطة جديدة $p(\bar{\bar{x}},\bar{\bar{y}})$ حيث:

$$\bar{\bar{x}} = (x - x_p)\cos\theta - (y - y_\theta)\sin\theta$$
$$\bar{\bar{y}} = (y - y_p)\cos\theta + (x - x_p)\sin\theta \qquad (24-12)$$

حيث θ يجب أن تكون بالزوايا النصف قطرية وتكون مضاعفاتها $2\pi/2$.

ثم يتمّ إعادة إزاحته إلى موقعه الأصلي بعد التدوير لتعطينا النقطة $p(\bar{\bar{\bar{x}}},\bar{\bar{\bar{y}}})$ حيث:

$$\bar{\bar{\bar{x}}} = (x - x_p)\cos\theta - (y - y_\theta)\sin\theta + x_p$$
$$\bar{\bar{\bar{y}}} = (y - y_p)\cos\theta + (x - x_p)\sin\theta + y_p \qquad (25-12)$$

حيث يكون التدوير باتجاه عقرب الساعة ولتدويره عكس عقرب الساعة يجب فقط تغيير الزاوية إلى $-\theta$ بدلاً من θ.

الدالة **rotatpoint( )** في البرنامج لتدوير نقطة حيث يتمّ استدعاؤها بعدد النقاط التي تعرف الشكل المطلوب تدويره.

مثال:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
# include <math.h>
//int px1,px2,px3,px4, py1,py2,py3,py4;
```

```
//{=========================================}
void Drawsqure(int px1,int py1,int px2,int py2,int px3,int py3,
                        int px4,int py4)
{
   line(px1,py1,px2,py2);
   line(px2,py2,px3,py3);
   line(px3,py3,px4,py4);
   line(px4,py4,px1,py1);
}
//{=========================================}
void Translat (int flag, int x11,int y11,int & px1,int &py1,int
&px2,int &py2,      int &px3,int &py3,int &px4,int &py4)
{
   px2 = px2 - (x11 * flag);
   py2 = py2 - (y11 * flag);
   px3 = px3 - (x11 * flag);
   py3 = py3 - (y11 * flag);
   px4 = px4 - (x11 * flag);
   py4 = py4 - (y11 * flag);
   px1 = px1 - (x11 * flag);
   py1 = py1 - (y11 * flag);
}
//{=========================================}
void rotatepoint (int alpha,int &x,int &y)
{
  float xtemp,ytemp;
  float sine,cosine;
  float pi = 22/7;
  xtemp = x;
  ytemp = y;
  cosine = cos(alpha * pi / 180);
  sine = sin (alpha * pi / 180);
  x = floor (cosine * xtemp - sine * ytemp);
  y = floor (sine * xtemp + cosine * ytemp);
}
//{=========================================}
int main(void)
{
  /* select driver and mode that supports use of setrgbpalette */
  int gdriver = DETECT, gmode , errorcode;
```

```
int x,x1,x2,x3,x4,y, y1,y2,y3,y4;
float alpha;
//int px1,px2,px3,px4, py1,py2,py3,py4;
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");
/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk)
{ /* an error occurred */
  printf("Graphics error: %s\n", grapherrormsg(errorcode));
  printf("Press any key to halt:");
  getch();
  exit(1);          /* terminate with an error code */
}
x1 = 300;   y1 = 200;
x2 = 420;   y2 = 200;
x3 = 420;   y3 = 320;
x4 = 300;   y4 = 320;
x = 300;              y = 200;
alpha = 30;
// { Draw the squre in its orignal position}
  Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
// {translate the squre to the origion Point (0,0)}
  Translat (1,x,y,x1,y1,x2,y2,x3,y3,x4,y4);
// {redraw the squre inthe origion point(0,0)}
  Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
/* {Scale the squre by multiplaying each of its points
  whith the scaling factor of 2}*/
  rotatepoint (alpha,x1,y1);
  rotatepoint (alpha,x2,y2);
  rotatepoint (alpha,x3,y3);
  rotatepoint (alpha,x4,y4);
  setlinestyle (1,1,1);
  Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
// {translate the squre to the its orignal positon}
  Translat (-1,x,y,x1,y1,x2,y2,x3,y3,x4,y4);
// {redraw the squre in the new postion}
  Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
/* clean up */
getch();
```

```
    closegraph();
    return 0;
}
```

- تحويلات القص shear Transformation

ينتج عن هذا النوع من التحويلات انحرافات بالشكل تمثل حالة برم للشكل، أو أن تأثير shear يكون كما لو أن الشيء متكون من طبقات حيث يتسبب بانزلاقها فوق بعضها. ومن تحويلات shear الشائعة هي باتجاه x وباتجاه y، حيث ينتج shear باتجاه x باستخدام مصفوفة التحويل التالية:

$$\begin{bmatrix} 1 & 0 & 0 \\ shx & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (26-12)$$

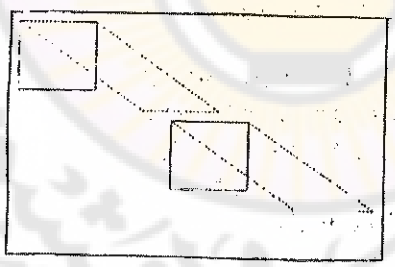
shx يمكن أن تأخذ أي قيمة حقيقية، هذا التحويل سيؤثر في قيم إحداثيات x. أما قيم إحداثيات y فستبقى كما هي. قيم shx السالبة ستؤدي إلى إزاحة الشكل أفقياً إلى اليسار.

مثال:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
# include <math.h>
//int px1,px2,px3,px4, py1,py2,py3,py4;
//{=====================================}
void Drawsqure(int px1,int py1,int px2,int py2,int px3,int py3,
                    int px4,int py4)
{
   line(px1,py1,px2,py2);
   line(px2,py2,px3,py3);
   line(px3,py3,px4,py4);
   line(px4,py4,px1,py1);
```

```
}
//{=========================================}
void Translat (int flag, int x11,int y11,int & px1,int &py1,int
&px2,int &py2, int &px3,int &py3,int &px4,int &py4)
{
    px2 = px2 - (x11 * flag);
    py2 = py2 - (y11 * flag);
    px3 = px3 - (x11 * flag);
    py3 = py3 - (y11 * flag);
    px4 = px4 - (x11 * flag);
    py4 = py4 - (y11 * flag);
    px1 = px1 - (x11 * flag);
    py1 = py1 - (y11 * flag);
}
//{=========================================}
void shearpointx(float shx, int & x,int&y)
{
        x = floor(x+shx*y);
  y = y;
}
int main(void)
{
  /* select driver and mode that supports use of setrgbpalette */
  int gdriver = DETECT, gmode , errorcode;
  int x,x1,x2,x3,x4,y, y1,y2,y3,y4;
  float sx;
  //int px1,px2,px3,px4, py1,py2,py3,py4;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);          /* terminate with an error code */
  }

  x1 = 300;   y1 = 200;
```

```
    x2 = 420;   y2 = 200;
    x3 = 420;   y3 = 320;
    x4 = 300;   y4 = 320;
    // { Draw the squre in its orignal position}
      Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
    // {translate the squre to the origion Point (0,0)}
      Translat (1,x1,y1,x1,y1,x2,y2,x3,y3,x4,y4);
    // {redraw the squre inthe origion point(0,0)}
      Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
                sx = 2.0;
      shearpointx (sx,x3,y3);
      shearpointx (sx,x4,y4);
      setlinestyle(1,1,1);
      Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
      x = 300;    y = 200;
    // {translate the squre to the origion Point (150,120)}
      Translat (-1,x,y,x1,y1,x2,y2,x3,y3,x4,y4);
    // {redraw the squre in the new postion}
      Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
    /* clean up */
    getch();
    closegraph();
    return 0;
}
```



شكل (12-44)

تحويلات shear باتجاه المحور العمودي y كما يلي:

$$\begin{bmatrix} 1 & shy & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (27-12)$$

يولد هذا التحويل إزاحة أفقية في موقع الإحداثيات، shy يمكن أن تأخذ أي قيمة حقيقية، والتي تغير مواقع مركبة y من إحداثيات نقاط الشكل بقيمة تتناسب مع قيمة x.

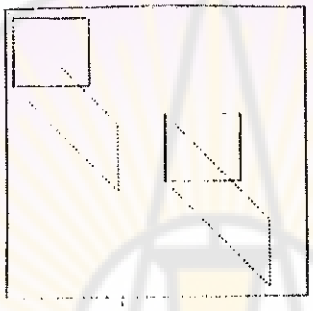
مثال:

البرنامج التالي يجري تحويلات **shear** للمربع باتجاه y.

```
#include <grapnics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
# include <math.h>
//{=====================================}
void Drawsqure(int px1,int py1,int px2,int py2,int px3,int py3,
                    int px4,int py4)
{
   line(px1,py1,px2,py2);
   line(px2,py2,px3,py3);
   line(px3,py3,px4,py4);
   line(px4,py4,px1,py1);
}
//{=====================================}
void Translat (int flag, int x11,int y11,int & px1,int &py1,int
&px2,int &py2,      int &px3,int &py3,int &px4,int &py4)
{
   px2 = px2 - (x11 * flag);
   py2 = py2 - (y11 * flag);
   px3 = px3 - (x11 * flag);
   py3 = py3 - (y11 * flag);
   px4 = px4 - (x11 * flag);
   py4 = py4 - (y11 * flag);
```

```
    px1 = px1 - (x11 * flag);
    py1 = py1 - (y11 * flag);
}
//{=====================================}
void shearpointy(float shy, int & x,int&y)
{
        y = floor(y+shy*x);
  x = x;
}
int main(void)
{
  /* select driver and mode that supports use of setrgbpalette */
  int gdriver = DETECT, gmode , errorcode;
  int x,x1,x2,x3,x4,y, y1,y2,y3,y4;
  float sy;
  //int px1,px2,px3,px4, py1,py2,py3,py4;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
}
 x1 = 300;   y1 = 200;
 x2 = 420;   y2 = 200;
 x3 = 420;   y3 = 320;
 x4 = 300;   y4 = 320;
// { Draw the squre in its orignal position}
   Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
// {translate the squre to the origion Point (0,0)}
   Translat (1,x1,y1,x1,y1,x2,y2,x3,y3,x4,y4);
// {redraw the squre inthe origion point(0,0)}
   Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
                sy = 2.0;
  shearpointy (sy,x2,y2);
  shearpointy (sy,x3,y3);
```

```
  setlinestyle(1,1,1);
  Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
  x = 300;    y = 200;
// {translate the squre to the origion Point (150,120)}
  Translat (-1,x,y,x1,y1,x2,y2,x3,y3,x4,y4);
// {redraw the squre in the new postion}
  Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
/* clean up */
getch();
closegraph();
return 0;
}
```



شكل (12-45)

- التحويلات المركبة Composite transformation

بصورة عامة يمكن تمثيل أي تركيب من تحويلات الإزاحة والتقييس والتدوير والقص كما يلي:

$$[\bar{x}\ \bar{y}1] = [x\ y1]\begin{bmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{bmatrix} \qquad (28-12)$$

لذلك فإن المعادلات التي يمكن أن تستخدم لحساب الإحداثيات بعد التحويل كما يلي:

$$\bar{x} = ax + by + c$$
$$\bar{y} = dx + ey + f \qquad (29\text{-}12)$$

هذه الحسابات تتضمن أربع عمليات ضرب وأربع عمليات جمع لكلّ نقطة من النقاط التي تمثل الشكل، وهذا هو أكبر عدد من الحسابات المطلوبة لحساب زوج من الإحداثيات لأي سلسلة من التحويلات، وذلك بعد أن يتمّ تسلسل المصفوفات الانفرادية وإيجاد المصفوفة المركبة الناتجة من حاصل ضربها مع بعضها.

إذا فالتطبيق الكفء لعمليات التحويل، يتمّ بصياغة التحويلات المركبة، لأي سلسلة من التحويلات وحساب قيم الإحداثيات المحمولة من العلاقتين. فمثلاً لتقييس شكل معين حول نقطة ارتكاز محددة يمكننا الحصول على المصفوفة المركبة لعمل هذا التحويل والناتجة من حاصل ضرب مصفوفات التحويل الثلاثة مع بعضها وهي:

أولاً. مصفوفة الإزاحة إلى نقطة الأصل بالنسبة لنقطة الارتكاز المحددة.

ثانياً. مصفوفة التقييس

ثالثاً. مصفوفة الإزاحة إلى الموقع الأصلي وهي كما يلي حسب الترتيب مع المصفوفة المركبة:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -xF & -y_F & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_F & y_F & 1 \end{bmatrix} =$$

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ (1-sx)x_F & (1-sy)y_F & 1 \end{bmatrix} \qquad (30\text{-}12)$$

ومن الطبيعي أنّ عدد الحسابات باستخدام المصفوفة المركبة (12-30) سيكون أقل بكثير مما لو استخدمنا المصفوفات الانفرادية كلٌّ على حدة، حيث يمكن حساب

إحداثيات أي نقطة من نقاط الشكل (xi,yi) بعد إجراء هذا التحويل عليها من العلاقتين التاليتين:

$$xi = xi(sx + (1 - sx)xf)$$

$$yi = yi(sy + (1 - sy)yf)$$

حيث xf, yf تمثلان إحداثيات نقطة الارتكاز.

مثال:

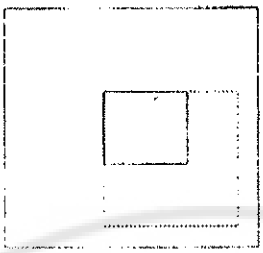
البرنامج التالي يجري تقييساً للمربع الذي رؤوسه النقاط $(x1, y1)$، $(x2, y2)$، $(x3, y3)$، $(x4, y4)$ والتي قيمها على التوالي:

$$(300,290),(420,200),(420,420),(320,320)$$

(باستخدام المصفوفة المركبة (12-30))

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
# include <math.h>
//{=====================================}
void Drawsqure(int px1,int py1,int px2,int py2,int px3,int py3,
                                    int px4,int py4)
{
   line(px1,py1,px2,py2);
   line(px2,py2,px3,py3);
   line(px3,py3,px4,py4);
   line(px4,py4,px1,py1);
}
//{=====================================}
void compscale(int xf, int yf, float sx,float sy, int &x,int & y)
{
        x = floor(x * sx + (1-sx)*xf);
  y = floor(y * sy + (1-sy)*yf);
}
//{=====================================}
```

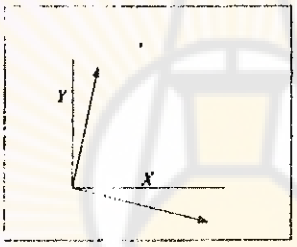
```
int main(void)
{
  /* select driver and mode that supports use of setrgbpalette */
  int gdriver = DETECT, gmode , errorcode;
  int x,x1,x2,x3,x4,y, y1,y2,y3,y4;
  float sy;
  //int px1,px2,px3,px4, py1,py2,py3,py4;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
  }
  x1 = 300;   y1 = 200;
  x2 = 420;   y2 = 200;
  x3 = 420;   y3 = 320;
  x4 = 300;   y4 = 320;
        x = 300;                    y = 200;
 // { Draw the squre in its orignal position}
    Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
 // {make compiste scale for each point of square}
        compscale(x,y,2,2,x1,y1);
    compscale(x,y,2,2,x2,y2);
    compscale(x,y,2,2,x3,y3);
    compscale(x,y,2,2,x4,y4);
    setlinestyle(1,1,1);
    Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
    x = 300;     y = 200;
 /* clean up */
 getch();
 closegraph();
 return 0;
}
```



شكل (12-46)

التقييس الذي تمّ حسب المصفوفة المركبة (12–30) كان بالاتجاهين x,y فقط. يمكننا إجراء التقييس للأشكال باتجاهات أخرى وذلك بعمل تركيبة من تحويلات التدوير والتقييس.

فلو فرضنا بأننا نريد تطبيق معاملات تقييس S1 و S2 في الاتجاهات الموضحة في الشكل (12.5).



شكل (12-47) معاملات التقييس S1 و S2 يطبقان في الاتجاهات المحسوبة من إزاحة زاوية مقدارها θ عن المحور x.

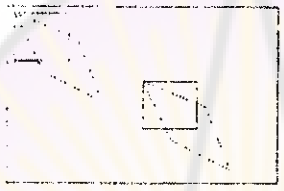
لإنجاز ذلك يجب أولاً عمل تدوير بحيث تتطابق اتجاهات S1 و S2 مع المحورين x,y على التوالي. ثم تطبيق تحويلات التقييس يليها تدوير بالاتجاه المعاكس لإعادة النقاط إلى مواقعها الأصلية.

مصفوفة التحويل المركبة الناتجة من حاصل ضرب مصفوفات التحويل الثلاثة السابقة هي:

(31-12)

$$\begin{bmatrix} S1\cdot\cos^2\theta + S2\cdot\sin^2\theta & (-S1+S2)\cdot\sin\theta\cos\theta & 0 \\ (-S1+S2)\cdot\sin\theta\cos\theta & S1\cdot\sin^2\theta + S2\cdot\cos^2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

وكمثال تطبيق على هذا النوع من تحويلات التقييس، يمكننا تجربته على الشكل المربع المستخدم في البرامج السابقة مع اعتبار أن قيم كل من $\theta = 45^\circ$ و $S1 = 1$ و $S2 = 2$.



شكل (12-48)

ويمكن استخدام طريقة التحويلات المركبة لحساب المصفوفة المركبة لتدوير أي شكل رسومي وذلك بضرب المصفوفة الثلاثية التالية مع بعضها وهي: أولاً مصفوفة إزاحة الشكل بحيث تنطبق نقطة الارتكاز على نقطة الأصل، مصفوفة التدوير حول نقطة الأصل، والثالثة هي مصفوفة إزاحة الشكل إلى موقعه الأصلي بعد التدوير. هذه السلسلة من المصفوفات مع مصفوفة التدوير الناتجة عن ضربها مع بعضها موضحة أدناه على التوالي:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ -x_R & -y_R & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_R & y_R & 1 \end{bmatrix}$$

(32-12)

$$= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ (1-\cos\theta)x_R + y_r \sin\theta & (1-\cos\theta)y_R - x_R \sin\theta & 1 \end{bmatrix}$$

المصفوفة السابقة تؤدي إلى تدوير الشكل بزاوية مقدارها θ مقاسة بالزوايا النصف قطرية وباتجاه عقرب الساعة، وإذا كان المطلوب تدوير الشكل بعكس عقرب الساعة فإن مصفوفة التدوير ستكون كما يلي:

(33-12)

$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ (1-\cos\theta)x_R - y_r \sin\theta & (1-\cos\theta)y_R + x_R \sin\theta & 1 \end{bmatrix}$$

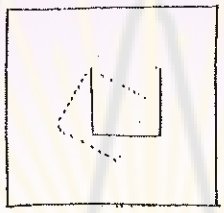
مثال:

البرنامج يُدوِّر المربع باستخدام المصفوفة المركبة (25.5):

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
# include <math.h>
//{=============================================}
void Drawsqure(int px1,int py1,int px2,int py2,int px3,int py3,
                                    int px4,int py4)
{
   line(px1,py1,px2,py2);
   line(px2,py2,px3,py3);
   line(px3,py3,px4,py4);
   line(px4,py4,px1,py1);
}
//{=============================================}
```

```
void comprotatepoint(int xr, int yr, float theta, int &x,int & y)
{
        float xtemp,ytemp, sine,cosine;
  float pi = 22/7;
  xtemp = x;
  ytemp = y;
  cosine = cos (theta * pi / 180);
  sine = sin (theta * pi / 180);
  x = floor (cosine * xtemp - sine * ytemp + (1-cosine) * xr + yr *
sine);
  y = floor (sine * xtemp + cosine * ytemp + (1-cosine) * yr - xr *
sine);
}
//{=============================================}
int main(void)
{
  /* select driver and mode that supports use of setrgbpalette */
  int gdriver = DETECT, gmode , errorcode;
  int x,x1,x2,x3,x4,y, y1,y2,y3,y4;
  float alpha;
  //int px1,px2,px3,px4, py1,py2,py3,py4;
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);             /* terminate with an error code */
  }
  x1 = 300;   y1 = 200;
  x2 = 420;   y2 = 200;
  x3 = 420;   y3 = 320;
  x4 = 300;   y4 = 320;
        x = 300;                    y = 200;
  alpha = 30;
  // { Draw the squre in its orignal position}
    Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
```

```
// {make compiste rotation for each point of square}
      comprotatepoint(x,y,alpha,x1,y1);
   comprotatepoint(x,y,alpha,x2,y2);
   comprotatepoint(x,y,alpha,x3,y3);
   comprotatepoint(x,y,alpha,x4,y4);
   setlinestyle(1,1,1);
   Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
 /* clean up */
 getch();
 closegraph();
 return 0;
}
```



شكل (12-49)


وقل رب زدني علما
جامعة دمشق
Damascus University

المصطلحات العلمية

Abstract	تجريدي
Address	عنوان
Algorithm	خوارزمية
Append	إلحاق
Argument	وسيط
Arithmetic	حسابي
Array	مصفوفة
Assignment	تعيين
Base	أساس
Bath file	ملف دفعي
Block	كتلة
Brace	{ }
Brackets	[ ]
Browser	استعراض
Buffer	ذاكرة مؤقتة
Build	بناء
Call	استدعاء (دالة)
Cast	تحويل قسري
Catch	التقاط
Catch block	كتلة التقاط
Class	صف
Code	رمز

Comment	تعليق
Compilation	ترجمة
Compiler	مترجم
Component	مكوّن
Condition	شرط
Configuration	تكوين عام
Constant	ثابت
Constructor	باني
Container	يتضمن
Coordinates	إحداثيات
Cursor	مؤشر
Data	بيانات
Data member	عضو بياني
Declaration	تصريح
Decrement	تناقص
Definition	تعريف
Denominator	مقام
Derived	مشتق
Destructor	مهدّم
Dialog box	مربع حوار
Digit	رقمي
Directive	مرشد
Double	مضاعف

Edit	تحرير
Element	عنصر
Entity	كينونة
Enumerated	تعداد
Environment	بيئة
Escape sequence	دالة هروب
Exception	استثناء
Executable file	ملف تنفيذي
Execute	تنفيذ
Exponent	أس
Expression	تعبير
Extend	توسيع
Extraction	استخراج
Field	حقل
Flag	علَم
Float	عائم
Function	دالة
Function member	عضو داليّ
Generate	توليد
Global	عام (شامل)
Header	ترويسة
Heap	ذاكرة كومة
Hyperlink	ارتباط فائق

Increment	تزايد
Index, subscript	فهرس
Inheritance	وراثة
Initial	قيمة ابتدائية
Initialization	تهيئة
Inline	مباشر
Input	دخل
Insert	إدراج
Install	تنصيب
Integer	عدد صحيح
Interaction	تفاعل
Interface	واجهة
Item	حد
Iterator	مُكرِّر
Label	وسم
Link	ارتباط
Linked list	قائمة مرتبطة
Linker	رابط
List	قائمة
List box	مربع سرد
Local	محلي (موضعي)
Long	طويل
Loop	حلقة

Manipulator	مناور
Mathematical	رياضية
Member	عضو
Mode	نمط
Module	وحدة
Multimedia	وسائط متعددة
Negation	عكس العلامة
Negative	سالب
Nested	متداخل
Null	خامل (صفري)
Numerator	بسط
Object	غرض (كائن)
Object oriented	غرضي التوجه
OOP	برمجة غرضي التوجه
Operating system	نظام التشغيل
Operator	معامل
Output	خرج
Overflow	فائض
Overload	تحميل زائد
Paradigm	منهج
Parameter	بارامتر
Pointer	مؤشر
Pop	سحب

Positive	موجب
Postfix	التالي
Precedence	أولوية
Precision	دقة
Prefix	السابق
Private	خاص
Processor	معالج
Prompt	موجّه
Properties	خصائص
Protected	محمي
Public	عام
push	دفع
Queue	طابور
Quotation	علامة اقتباس
Redirection	تغيير الوجهة
Reference	مرجع
Register	مسجّل
Relational	علائقي
Resource	مصدّر (مورد)
Return	إعادة
Root directory	دليل جذري
Run	تشغيل
Scope	مدى (متغيّر)

Settings	إعدادات
Setup	تنصيب
Single	أحادي
Slash	\
Sort	فرز
Source	مصدر
Space	مسافة
Square root	جذر تربيعي
Stack	مكدس
Standard	قياسي
State	حالة
Statement	عبارة
Static	ساكن
Stream	دفق
String	سلسلة محارف
Structure	بنية
Syntax	تركيب نحوي
Tab	حرف جدولة
Template	قالب
Update	تحديثي
Variable	متغيّر
Vector	متّجه
Version	إصدار

View	معاينة
Void	عقيم
Window	نافذة

المراجع العلمية

Bibliography

1. Ellis, Margaret A. and Bjarne Stroustrup. *The Annotated C++ Reference Manual*, Addison-Wesley Publishing Company, Reading, MA, 1991.
2. Kernighan, Brian W. and Dennis M. Ritchie. *The C Programming Language*, Second Edition, Prentice Hall, Englewood Cliffs, NJ, 1988.
3. Microsoft Corporation, *Microsoft Developer Network Developer Library*, July, 1995.
4. Microsoft Corporation, *Windows 95 Game SDK*, 1995.
5. Press, William H., Saul A. Teukolsky, William T. Vettering, and Brian P. Flannery. *Numerical Recipes in C*, Second Edition. Cambridge University Press, 1992.
6. SAMS Publishing, multiple authors, Programming Windows 95 Unleashed, Sams Publishing, Indianapolis, IN. 1995.
7. Shirley, John and Ward Rosenberry. *Microsoft RPC Programming Guide*, O'Reilly & Associates, Sebastopol, CA, 1995.
8. Stevens, W. Richard. *UNIX Network Programming*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1990.
9. Stroustrup, Bjarne. *The C++ Programming Language*, Second Edition, Addison-Wesley Publishing Company, Reading, MA, 1991.
10. Leendert Ammereal, "Programming Principles In computer Graphics ", Second Edition, John Wiley & Sons, New York, 1986.
11. Scotts Valley; CA :"Borland C++", Borland International, 1991.
12. Salmon .R and M .Slater; "computer Graphics system & concepts", Wokingham, Addison Wesley,1987.

13. ترجمة مركز التعريب :"الدليل الكامل C++ " الدار العربية للعلوم ، 1997.

14. ترجمة د. صلاح الدوه جي: "كيف تبرمج بلغة C++ " دار شعاع للنشر والعلوم، 2000.

وقل رب زدني علما
جامعة دمشق
Damascus University