جامعة دمشق

الكلية التطبيقية

محاضرات البرمجة المتقدمة

قسم الميكاترونيكس

السنة الثانية

م. آلاء خسارة

alaa

المحاضرة الأولى

ايعازات القرار والتكرار
وبنى التحكم

تعتبر القواعد الأكثر اهمية في البرمجة. وهي ايعازات القرار (**if statement**) وكذلك الإيعاز المرافق لها (**else**) ، وعبارات التكرار التي هي (**while loop -do.. while loop -for loop**)، غالبا تعد هذه الأوامر من الأوامر الكثيرة الاستخدام في البرمجة .

1. عبارة اذا (if) Statement

يستخدم هذا الأمر لاتخاذ قرار من قبل المترجم بناءا على بعض المعطيات التي ترد في البرنامج، هناك العديد من الحالات التي لايمكن التنبؤ بها من قبل المبرمج أثناء كتابة البرنامج.

ان استخدام عبارة (if) يكون بحيث إذا تحقق الشرط الذي يرافق الأمر if فيتم تنفيذ العبارة التي بعده ، أما إذا لم يتحقق فتهمل العبارة.

يمكن صياغة الشروط الخصة بالبنى التي تعتمد على تعليمة if باستخدام عمليات المساواة والمقارنة كما في المثال التالي:

Expression ? Expression1 : Expression2

(a>b)? c : d

أي أنه إذا كان الشرط صحيح فاختر القيمة c ، وإذا كان غير صحيح فاختر القيمة d .

مثال:

مثال :

إذا كان المسبح ممتلئا افتح الصنبور A ثم B

```
if   pool_ full
  {
  open A ;  }
   open B ;
```

هنا يتم تنفيذ التعليمة الأولى فقط

```
if   pool_ full
{
open A ;
open B ;      }
```

هنا يتم تنفيذ التعليمتين ثم يخرج من الحلقة ، أي أن في حالة عدم وضع { } فان أول عبارة ستأتي بعد الشرط الذي بعد الأمر if هي التي ستعامل على أنها تعود الى الأمر if وتنفذ في حالة تحقق الشرط.

2. الجملة الشرطية أو بنية الاختيار if – else:

تقوم بفحص تعبيرمعين، ويعيد قيمة معينة اذا كان ذلك التعبير صح، ويعيد قيمة مختلفة اذا كان ذلك التعبير خطأ، هذا العامل هو اختصار لعامل الاختيار if. else الصيغة العامة له:

condition ? result1: result2

فاذا كان الشرط condition صح فان التعبير سيعيد القيمة 1result اما اذا كان خطأ فانه سيعيد القيمة 2result .

لنرى المثال التالي:

```
if (pin1 == digit )
light high;
else
light low;
```

دائما تستخدم (if) عندما تحتاج أن تختار بين أكثر من حالة أي أختيار عمل أو حالة واحدة من بين أثنين أو أكثر

مثال:

طباعة عددين بمختلف الحالات :

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

```
void main()
{
int z,y;
cout<< "enter first number"<<endl;
cin>> z;
cout<<"enter scanned number"<<endl;
cin>>y;
if(z>y)
cout<<"the large number is first"<<z<<endl;
else
if(z==y)
cout<<"z=y";
else
cout<<"the large number is  scaned";
}
```

مثال :

اكتب برنامج لخط سير سيارة مؤتمتة تقوم بنقل مواد ضمن معمل بحيث إذا كانت المواد المحملة ذات وزن أكبر من 50 كغ تسير على المسار ذو الأسود وإلا تسير على المسار ذو اللون الأبيض ، ثم تعود إلى نقطة البداية.

Nested for loop

مثال:

اكتب برنامج بلغة ++c باستخدام البنية for من أجل حساب مجموع الأعداد الزوجية الصحيحة من 2 وحتى 100 :

```
#include<iostream>
using namespace std;
void main()
{
Int sum=0;
for (int number = 2; number <= 100; number +=2)
sum+= number;
cout<< “sum =”<<sum<<endl;
}
```

مثال :

اكتب برنامج يقوم بطباعة الشكل التالي :

```
*  *  *  *        *  *  *  *
*        *        *  *  *  *
*        *        *  *  *  *
*  *  *  *        *  *  *  *
```

الشكل الأول عبارة عن مربع

```
int main()
{ int x;
 cout<<”enter number\n”;
 cin>>v;
```

```
for(int b=1;b<=v;b++)
{ for(int c=1;c<=v;c++)
{cout<<"*";}
Cout<<endl; }
return 0;}
```

الشكل الثاني :

```
int main()
{ int x;
 cout<<"enter number\n";
 cin>>v;
for(int b=1;b<=v;b++)
{ for(int c=1;c<=v;c++)
{  if (b==1||b==v)
cout<<"*";}
else if (c==1||c==v)
cout<<"*";}
else { cout<<"  "; }
cout<<endl; }
return 0;}
```

بنية التكرار Do-While:

تشبه بنية التكرار do-while البنية while حيث تقوم البنية while بالتحقق من صحة شرط الاستمرار بالتكرار في بداية الحلقة قبل تنفيذها، أما في حالة البنية do-while فيتم ذلك بعد تنفيذ جسم الحلقة أولا ، أي يتم تنفيذ جسم البنية do-while مرة واحدة على الأقل . عند الانتهاء من تنفيذها يتم الانتقال إلى التعليمة التي تأتي مباشرة بعد جزئها while.

شكلها العام:

```
do { Statement }while (condition);
```

ويجب بنهاية سطر الwhile وضع فاصلة منقوطة

فمثلا في التمرين التالي نجد ان البرنامج يقوم بطباعة الرقم 12 ثم ينظر إلى الشرط ويتوقف :

```
int counter =12;
do{
cout<<"counter="<<counter<<endl;
counter++;
}while(counter<=10);
cout<<"the End";
```

مثال:

اكتب برنامج يقوم بطباعة الأعداد من 1 إلى 10 باستخدام البنية do-while .

```
#include<iostream>
using main space std;
void main()
{  int counter=1;
do {
cout<<counter<<" ";
} while (counter<=10);
counter++;
}
or
While(++counter<=10);
```

مثال:

اكتب برنامج يقوم بجمع الأعداد المدخلة ومن ثم يتوقف عن العمل ويقوم بطباعتها بمجرد إدخال الرقم صفر:

```
#include <iostream>
using namespace std;
void main()
{
int x, sum;
sum = 0;
do
{
cout<< “x=”<<endl;
cin>>x;
sum+=x;
}while (x!=0);
cout<<"sum of number is ="<<sum;
}
```

إن العبارة sum+=x تكافئ العبارة sum = sum+x

حلقة الإيقاف break والاستمرار continue:

تستخدم تعليمتي الإيقاف والاستمرار من أجل تغيير مجرى تدفق التحكم ضمن البرنامج .

حلقة break : تعمل على إيقاف البنية أوحلقة تكرار عند تحقق شرط معين ، حيث يتم الخروج مباشرة من هذه البنية ويتابع البرنامج بعدها مع أول تعليمة تليها مباشرة ، ويضاف إلى ذلك تخطي ماتبقى من البنية والخروج منها .

مثال :

```
#include <iostream>
void main()
```

```
{
for(int i=1 ; i<= 100 ; i++ )
{ cout<<i;
if(i = = 10)
break; } }
```

حلقة continue :

تعمل على تجاوز ماتبقى من التعليمات في حلقة التكرار خلال الدورة الحالية والانتقال إلى الدورة التالية ، أي المتابعة مع المرور الثاني للحلقة ، في البنيتين while و do-while يتم القيام بالتحقق من شرط استمرار التكرار مباشرة بعد تنفيذ التعليمة continue ، اما مع البنية for يتم القيام بعملية الزيادة أولا ثم يبدأ بعدها تقييم شرط التكرار .

مثال:

```
#include <iostream>
using namespace std;
void main()
{
for(int i=1; i<=100; i++)
{
if(i==10)
continue;
cout<< i <<endl;
}
}
```

مثال آخر:

اكتب برنامج يقوم بطباعة جميع الأرقام بين (1 – 100) والتي تقبل القسمة على الأعداد (2- 4 – 6) :

```
#include <iostream>
using namespace std;
```

```
void main()
{
for(int i=1; i<=100; i++)
{
   if(i%2!=0 )
   continue;
else  if(i%4!=0 )
   continue;
else  if(i%6!=0 )
   continue;
cout<<i<<endl;
} }
```

حل آخر:

```
#include <iostream>
using namespace std;
void main()
{
for(int i=1; i<=100; i++)
{
if(i%6=0 && i%4=0 && i%2=0)
cout<< i<<endl;}}
```

المصفوفات Arrays

المصفوفة هي مجموعة من البيانات (العناصر)، بحيث أن جميع هذه العناصر من نوع واحد double , int , char , float ويتم تخزينها في الذاكرة في مواقع متجاورة، وتعرف المصفوفة من خلال الاسم الذي يسند لها ويتم اختيار اسم المصفوفة وفقا لقواعد اختيار اسماء المتغيرات ويستخدم هذا الأسم للاشارة الى المصفوفة وليس الى عناصر المصفوفة اذ ان عناصرالمصفوفة يتم الاشارة الى كل واحد منها باستخدام أسم المصفوفة متبوعا برقم يشير الى موقع العنصر فيها.

تستخدم المصفوفة كبديل للإعلان عن عدد من المتغيرات المتشابهة النوع.

التصريح عن المصفوفة:

هناك ثلاثة أشياء يجب أن تؤخذ بعين الاعتبار عند التصريح عن المصفوفة :

1. اسم المصفوفة : ويتم اختياره مثلما يتم اختيار المتغيرات العادية .
2. عدد العناصر التي بداخلها.
3. نوع البيانات المستخدمة فيها.

المصفوفات الأحادية:

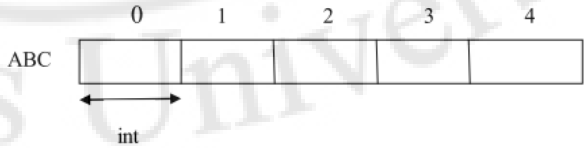
المصفوفة الاحادية هي عبارة عن سلسلة من العناصر المتشابهة والتي تخزن في الذاكرة في مواقع متجاورة والتي من الممكن الاشارة لكل واحد من هذه العناصربشكل منفرد من خلال إضافة رقم الدلالة (index) الى الاسم التعريفي لها، ويكون الاعلان عن المصفوفة الاحادية بكتابة النوع اولا متبوع باسم المصفوفة كما في المتغيرات، مع إضافة قوسين مربعين بعد اسم المصفوفة يحتويان على عدد عناصر المصفوفة (يشار له بحجم المصفوفة)، والصيغة العامة للإعلان عن المصفوفة هي:

```
Type   ArrayName [ number of elements ] ;
```

مثال :

```
int ABC [ 5 ] ;
```

في هذا المثال سيتم تحديد خمس مواقع متجاورة في الذاكرة من نوع الأعداد الصحيحة كما في الشكل



العنصر ٠	العنصر ١	العنصر ٢	العنصر ٣	العنصر ٤	العنصر ٥
قيمة	قيمة	قيمة	قيمة	قيمة	قيمة

إنشاء المصفوفة **Array Initialization**

عند الاعلان عن المصفوفة فانها ستنشأ كمصفوفة خالية من القيم، حيث ان عناصرها لاتحتوي على قيم مالم يتم خزن قيم فيها أي اسناد قيم ابتدائية لهذه العناصر وتسمى هذ العملية ابتداء المصفوفة ، كما هو الحال مع المتغيرات الاعتيادية.

المصفوفات ايضا يمكن ان تعرف على انها محلية او عامة مثل المتغيرات الاعتيادية، وذلك من خلال وضع قيم بين قوسين متوسطين تفصل بين قيمة واخرى فاصلة ، كما يأتي:

ABC [5] = { 5, -234, 45, 0, 123 } ;

ويجب ان تنتبه الى ان عدد القيم بين القوسين المتوسطين يجب ان لاتزيد عن عدد عناصر المصفوفة التي تم الاعلان عنها في اعلان المصفوفة.

ملاحظة://

عندما يتم ابتداء القيم سوف تسند لعناصر المصفوفة، C++ يسمح بامكانية ترك الاقواس المربعه فارغة []. في هذه الحالة، فان المترجم سيفرض حجم للمصفوفة يطابق عدد القيم الموجودة بين الاقواس المتوسطة. مثال

ABC [] = { 3, 4, 5} ;

هنا سيحدد المترجم عدد العناصر بثلاث.

مثال عن كيفية إدخال عناصر المصفوفة:

يتم ذكر اسم المصفوفة ثم رقم العنصر الذي نريد التعامل معه بين قوسين ، فإذا أردنا مثلا ان نضع القيمة 75 في العنصر الثالث من المصفوفة يتم ذلك كما يلي Mark[2] = 75

كما في الشكل التالي

رقم العنصر	0	1	2	3	4
قيمة العنصر	454	11	1	258	852

```
#include<iostream>
 using namespace std;
void main (void) {
int a[7];
int   i ;
for ( i=0 ; i<=6 ;i++ )
```

```
  cin >> a[i] ;
}
```

شرح مثال عن كيفية كتابة مصفوفة لعلامات طالب:

عندما نريد طباعة مجموعة علامات طالب ما وفق الطريقة التقليدية تكون على الشكل التالي:

```
# include<iostream>
#include<string>
 using namespace std;
void main () {
string name “Omer”;
float mark1;
float mark2;
float mark3;
float mark4;
mark1=90;
mark2=91.5;
mark3=92;
mark4=93.5;
cout<<”student name:”<<endl;
cout<<”mark1=”<<mark1<<endl;
cout<<”mark2=”<<mark2<<endl;
cout<<”mark3=”<<mark3<<endl;
cout<<”mark4=”<<mark4<<endl;
```

تعتبر من الطريق الطويلة وقد تسبب أخطاء وخاصة في حال كان هناك أكثر من طالب لأنه وبهذه الحالة تم تعريف أربعة متغيرات وتخزين قيمة في كل منها وجميعها لنفس الطالب، لذلك نقوم باستخدام مفهوم المصفوفة

وذلك عن طريق تعريف متغير واحد بداخله متغيرات أخرى متشابهة كما يلي :

يتم وضع حجم المصفوفة بين قوسين للدلالة على عدد العناصر فيها وهذا الرقم يعتبر ثابت وغير قابل للتعديل ...
يمكن ان نعرف قيمة المصفوفة قبل سطر كما في المثال المجاور... وبالتالي نضع بين قوسين size ذات القيمة الثابتة.
 	
المتغير mark فيه عدة متغيرات أي أننا بهذه الحالة لانحجز لكل علامة متغير مستقل ، ويمكن التمييز بينهم حسب المكان index .
أي ترتيب العناصر ضمن المصفوفة ويبدأ الترتيب من الرقم 0 لذلك الترتيب يكون أقل من حجم المصفوفة بمقدار 1.
	const int size = 4;
	float mark [4];
	cout<<"Array";
	marks[0]=90;
	marks[1]=91.5;
	marks[2]=92;
	marks[3]=93.5;
	cout<<"marks[0]="<<marks[0];
	cout<<"marks[1]="<<marks[1];
	cout<<"marks[2]="<<marks[2];
	cout<<"marks[3]="<<marks[3];
	}

ويوجد طريقة أخرى يمكن من خلالها إدخال البيانات وذلك عن طريق الادخال بشكل مباشر ، والذي يمكن ان يتم على نفس السطر .
 	
إذا كانت عدد القيم المعطاة لمصفوفة أقل من حجمها فإن c++ تعتبر القيم الناقصة مساوية للصفر ، وهذه الميزة موجودة فقط عند كتبة المصفوفة بشكل مباشر أي بين أقواس {}
	const int size=4;
	cout<<"Array"<<endl;
	float marks[size]= {90,91.5,92,93.5};
	cout<<"marks[0]="<<marks[0];
	cout<<"marks[1]="<<marks[1];
	cout<<"marks[2]="<<marks[2];
	cout<<"marks[3]="<<marks[3];
	}

مثال : احسب معدل الطالب وفق المصفوفة:

90 91 92 93 94

```
# include<iostream>
#include<string>
 using namespace std;
void main () {
stirng name;
cout<<"Enter the name:";
```

```
cin>>name;
float marks[5];
cout<<"Enter mark1=";
cin>>marks[0];
cout<<"Enter mark2=";
cin>>marks[1];
cout<<"Enter mark3=";
cin>>marks[2];
cout<<"Enter mark4=";
cin>>marks[3];
double sum;
sum=marks[0]+marks[1]+marks[2]+marks[3]+marks[4];
double total= sum/5;
cout<<"Total="<<total<<endl;
```

مثال

1. اكتب برنامج يقوم بإدخال سلسلة مؤلفة من عشرة رموز ثم يقوم بطباعتها وفق ترتيب الادخال وبعكسه:

```
#include<iostream>
void main()
{
char s[10];      حددنا عناصر المصفوفة بانها عشرة عناصر
for (int i=0;i<10;i++)   حلقة لإدخال العناصر
   cin>>s[i];
for(i=0;i<10;i++)   حلقة الإخراج التصاعدية
  cout<<s[i];
for(i=9;i>=0;i--)    حلقة الإخراج التنازلية
  cout<<s[i];
}
```

المصفوفات المتعددة الأبعاد

المصفوفات ثنائية البعد:

المصفوفات المتعددة الأبعاد ممكن ان تعرف على انها مصفوفة المصفوفات،

المصفوفات الثنائية لها استخدامات كثيرة وهي تساعد على تسهيل التعامل مع بعض المسائل المعقدة . فمثلا لدينا عدد من المعامل (ثلاثة معامل: معمل1، معمل2، معمل3) والتي تنتج مواد كهربائية متشابهة مثل (تلفزيون، ثلاجة، غسالة، مجمدة، مكيف) فيمكن تمثيل انتاجها بمصفوفة ثنائية والتعامل مع قيمها على هذا المبدأ كمايأتي:

	مكيف	مجمدة	غسالة	ثلاجة	تلفزيون
معمل1	21	12	56	34	23
معمل2	21	34	44	43	22
معمل3	12	15	23	31	42

الان لو سألنا كم غسالة انتجت في المعمل1.؟؟ بالتأكيد سيكون الجواب 56، واذا كان السؤال كم مكيف انتج في المعمل 3 فسيكون الجواب 12 وهكذا....حجم المصفوفة هو (3×5).

الاعلان عن المصفوفة الثنائية:

يتم الاعلان عن المصفوفة الثنائية بنفس الطريقة التي يتم فيها الاعلان عن المصفوفة الاحادية وذلك بكتابة نوع المصفوفة متبوعا باسم المصفوفة ثم عدد العناصر في المصفوفة وهنا يكون عدد العناصر موزعا على اربع اقواس مربعة (لأنها ثنائية)، القوسين المربعين الاولين يمثل عدد الصفوف في المصفوفة الثنائية والقوسين المربعين التاليين يمثلان عدد الاعمدة في المصفوفة، وكما يأتي:

```
int Test Array [3][5] ;
```

عدد عناصر هذه المصفوفة الكلي يساوي حاصل ضرب عدد الصفوف في عدد الاعمدة وسيكون مساوي 5=15×3).

مثال :

$$\begin{vmatrix} 6 & 5 \\ 1 & 4 \\ 3 & 7 \end{vmatrix}$$

تتالف هذه المصفوفة من عمودين وثلاثة أسطر ولإدخال هذه المصفوفة يجب علينا بداية ان نعرف البرنامج بسطر ثم بعمود العنصر المراد إدخاله وتكون بالطريقة التالية:

```
cin>> a[i] [j];
```

حيث: a اسم المصفوفة i رقم السطر j رقم العمود. ويكون الترقيم كما يلي :

	0 عامود	1 عامود
سطر 0	6	5
سطر 1	1	4
سطر 2	3	7

حيث يعبر الكود a[1][0]=1 ان القيمة 1 موجودة في السطر 1 والعمود 0 .

وهنا يمكننا أن نستخدم حلقتي for لإدخال المصفوفة ثنائية البعد كما يلي :

```
for(i=0 ; i<3 ; i++)
for(j=0 ; j<2; j++)
cin>>a[i][j];
```

الوصول لعناصر المصفوفة الثنائية:

عند الوصول إلى أي عنصر من عناصر المصفوفة الثنائية فيمكننا التعامل معه وإجراء كافة العمليات التي تتناسب مع نوعة كأي متغير اعتيادي، مثلا لغرض إسناد القيمة 56 لعنصر من عناصر مصفوفة ثنائية (نفرض ان العنصر في الموقع 5×3) فيتم ذلك كمايأتي:

```
TestArray [3][5] = 56 ;
```

لاحظ عند العمل على عناصر المصفوفة لاتحتاج لتحديد النوع لانه تم تحديدة عند الاعلان عن المصفوفة.

الآن لو أردنا طباعة قيمة هذا العنصر على الشاشة فسيكون كمايأتي:

```
cout << TestArray[3][5] ;
```

ويمكن مساواته لاي متغير اعتيادي مثل

```
X = TestArray[3][5] ;
```

وستكون قيمة المتغير x تساوي 56.

ابتداء المصفوفة الثنائية:

يقصد بالإبتداء هو اسناد قيم ابتدائية للمصفوفة ويكون بعدة طرق:

* يمكن ان تبدأ المصفوفة الثنائية بنفس الطريقة التي تم فيها بداية المصفوفة الأحادية وذلك من خلال كتابة اسم المصفوفة مع الاقواس التوي تمثل الابعاد ومساواتها الى مجموعة من القيم وتحدد القيم بين قوسين متوسطين مع ملاحظة ان عدد القيم يجب ان لايزيد عن عدد عناصر المصفوفة وكمايلي:

```
int  theArray[5][3] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 } ;
```

في هذ الحالة فان اول ثلاث قيم يتم اسنادها إلى المواقع الثلاث في الصف 0 وثاني ثلاث عناصر تسند الى المواقع الثلاث في الصف الاول وهكذا.

* ويمكن ان تكون مجاميع ثلاثية ضمن المجموعة الرئيسة وتسندها للمصوفوفة وكمايأتي:

```
int theArray[5][3]={{1,2,3}, {4,5,6}, {7,8,9}, {10,11,12}, {13,14,15}};
```

المترجم سيهمل الاقواس الداخلية التي ستساعد على فهم توزيع القيم بشكل سهل

بالامكان اسناد قيم الى عناصر المصفوفة باستخدام لوحة المفاتيح أثناء تنفيذ البرنامج وذلك باستخدام حلقتي تكرار متداخلتين الحلقة الخارجية تعمل كعداد للصفوف (تضع مؤشر على الصفوف) بينما الحلقة الداخلية تعمل كعداد للأعمدة (تضع مؤشر على الاعمدة<

مثال : اكتب برنامج لقراءة مصفوفة ثنائية بإدخال قيم عناصر من لوحة المفاتيح:

```
#include <iostream>
 using namespace std;
 int main()
 {
int SomeArray[5][4] ;
for (int i =0;<5 ; i++)
 for (int j=0; j<4 ; j++)
cin >> SomeArray [i][j] ;
```

```
 return 0;
for (int i = 0; i<5; i++)
 for (int j=0; j<4; j++)
   cout << SomeArray [i][j]<< '\t' ;

 return 0;
 }
```

لاحظ في المثال لايمكن استخدام اوامر الاخراج مالم يتم اسناد قيم لعناصر المصفوفة باحدى طرق اسناد القيم المبينة .

مثال:

- برنامج لطباعة عناصر مصفوفة على شكل صفوف واعمدة

```
#include <iostream>
using namespace std;
int main()
 {
 int SomeArray[5][4] ;
 for (int i = 0; i<5; i++)
for (int j=0; j<4; j++)
  cin >> SomeArray[i][j] ;
 for (int i = 0; i<5; i++)
 {
   for (int j=0; j<4; j++)
     cout << SomeArray[i][j]<< endl ;
   cout << endl ;
 }
 return 0; }
```

مثال:

اكتب برنامج يقوم بإدخال مصفوفة ثنائية البعد تتالف من سطرين وثلاثة أعمدة بحيث يقوم البرنامج باستبدال عناصر المصفوفة ذات القيمة الفردية بالقيمة (10) ثم يقوم البرنامج بطباعة شكل المصفوفة الجديد.

```
#include <iostream. >
void main()
{
int s[2][3],i,j;
for(i=0;i<2;i++)
for(j=0;j<3;j++)
  {
 cin>>s[i][j];
 if(s[i][j]%2!=0)
   s[i][j]=10;
 }
for(i=0;i<2;i++)
 {
 for(j=0;j<3;j++)
 cout<<s[i][j]<<"  ";
 cout<<endl;

}}

   }}
```

مصفوفات الأحرف Character Arrays

بلغة ++c يعبر عنها بـ strings والتي هي مجموعة من الأحرف يعبر عنها بالرموز char
مثلا: لتخزين جملة "السلام عليكم" تكون كما يلي:

```
Void main(){
String str= "السلام عليكم";
Cout<<str<<endl;
}
```

وبما ان النص عبارة عن مجموعة من الأحرف char ، فإذا أردنا عمل مصفوفة من characters يعني اننا نقوم بكتابة نص.
مثلا:

```
Char [c]= "c++";
Cout<< c <<endl;
```

يمكننا كتابتها دون الحاجة إلى أقواس كبيرة بشكل مباشر مابين علامتي التنصيص ، وذلك فقط في المصفوفة نوع char ، ويمكن معرفة حجم المصفوفة من خلال عدد الأحرف مع زيادة واحد

```
Char [4]= "c++";
```

ويمكننا التعبير عنها أيضا مثل cout << "hello world.\n";

ففي لغة ++C فإن السلاسل الرمزية هي عبارة عن مصفوفة لحرف تنتهي بالحرف null حرف النهاية(حيث يمثل نهاية السلسلة الرمزية، بالأمكان ان تعلن وتبتدأ السلاسل الرمزية كما تفعل بالضبط مع مصفوفات البيانات من الأعداد الصحيحة والحقيقية، مثال

char Greeting[ ] = { `H'، `e'، `l'، `l'، `o'، `` ، `W','o','r','l','d'; ، `\0' }

لاحظ ان الحرف الاخير مابعد الشرطة المعكوسة هو صفر.

القاعدة :

عندما نريد إنشاء متغير من النوع string من خلال مصفوفة من الأحرف char فإن البرنامج يقوم بحجز مساحة الذاكرة بأكبر من حجم النص بمقدار واحد وفي هذه المساحة الزائدة يضع البرنامج رمز خاص وهو null دلالة على انتهاء النص ويعبر عنه ب \0

لغة ++C توفر امكانية اختصار الطريقة اعلاه والتي تعتمد على ادخال حرف بعد الأخر، وكما يأتي:

char Greeting[ ] = "Hello World";

حيث ان هذه القاعدة توفرشيئين:

- فبدلا من استخدام الحاصرات المفردة المفصولة بالفواصل والمحاطة بالأقواس فاننا نستخدم الحاصرات المزدوجة بدون فواصل واقواس.

- عدم الحاجة لأضافة حرف النهاية لان المترجم سيضيفة عوضا عنك.

هنا حجم المصفوفة يساوي 12 byte وذلك لان كلمة Hello تحتاج الى خمس بايتات، فراغ واحد يحتاج بايت واحد، وكلمة World تحتاج الى خمس بايتات، واخيرا بايت واحد لحرف النهاية.

لذلك فعندما تعلن عن مصفوفة حرفية وتكتب حجمها فيجب ان يكون حجمها بعدد الاحرف زائدا واحد (الفراغ بين الأحرف يعامل معاملة الحروف)، مثال ; char Colour [4] = “RED”

سيتم اسناد الأحرف لكل موقع في مصفوفة الاحرف كما يأتي:

Colour[0] =’R’ ; Colour[1] =’E’ ; Colour[2] =’D’ ; Colour [3] = '\0';

مثال:

اكتب برنامج لقراءة مصفوفة احرف وطباعتها:

```
#include <iostream>
using namespace std;

 int main()
 {
 char buffer[80];
 cout << "Enter the string: ";
 cin >> buffer;
 cout << "Here's the buffer: " << buffer << endl;
 return 0;
 }
```

مخرجات البرنامج :

```
Enter the string: Hello World
Here's the buffer: Hello
```

لاحظ مدخلات ومخرجات البرنامج ، والذي يجب ان تتأكد بعدم وضع احرف اكثر من الحجم المحدد (حيث ان المصفوفة معرفة بحجم 81 حرف اي بامكانك ان تضع 79 حرف لان الحرف الاخير يمثل حرف النهاية.

المحاضرة الثالثة

مثال عن المصفوفات:

اكتب برنامج لطباعة عناصر القطر الرئيسي في مصفوفة 5×5:

```
#include <iostream>

#define row 5

#define col 5

using name space std;

void main()

{

Int D[row][col];

for(int i=0 ; i<5 ; i++)

for(int j=0 ; j<5 ; j++)

cin>>D[i][j];

for ( i =0 ; i < 5 ; i++ )

cout << D[i][i] << endl ;

}
```

مثال : أوجد العدد الأكبر وموقعه ضمن مصفوفة ثنائية (6×3) :

```
#include < iostream>
```

```
using namespace std;

int main() {

int max , A [3][6];

for ( int i=0 ; i< 3 ; i++ )

for ( int j =0 ; j< 6 ; j++ )

cin>>A [i][j] ;

max = A [0][0] ;

int row = 0 ;

int col = 0 ;

for ( int i=0 ; i< 3 ; i++ )

for ( int j =0 ; j< 6 ; j++ )

  if ( A [i][j] > max )

{  max = A [i][j] ;

row = i ;

col = j ;

}

cout << " Max element in array \n " << max ;

cout << "location of max element in array: \n" ;

cout << "row = " << row << " col=" << col ;
```

```
return 0;
```

السجلات (التركيبات) structures

تعريف السجل : هو مجموعة من البيانات (المتغيرات) المختلفة في النوع مع بعضها البعض بحيث يمكن التعامل معها كوحدة واحدة . أي هو نوع متغير جديد نقوم نحن بإدخاله ويكون خليط من أكثر من متغير يحتوي متغيرات متعددة.

وهو يشبه المصفوفة لكن مع فروقات:

1. لايمكن تحديد المكونات index له.
2. داله مجموعة عناصر ليس شرط ان تكون جميعها من نفس النوع كما في المصفوفات.

نوعها: يمكن أن يكون من نوع int وبعضها نوع char وبعضها float.....إلخ

الصيغة العامة:

```
Struct اسم السجل

{عدد المتغيرات ونوعها };
```

مثال:

```
Struct student {

String name;

Int age;

Int id;

};
```

يجب مراعاة ان تكون القيم بنفس الترتيب وان يتم ذكرها قبل ()main

مثلا:

لكتابة برنامج لتسجيل بيانات موظفين في الشركة نحتاج إلى تخزين:

1. اسم الموظف وهو من نوع مصفوفة حرفية char name [40]
2. عنوانه وهو من نوع مصفوفة حرفية char address [40]
3. عمره وهو متحول من نوع عدد صحيح int age
4. راتبه متحول من نوع عدد حقيقي float salary

وكما نلاحظ فإن جميع هذه البيانات يجب التعامل معها كوحدة واحدة لأنها لموظف واحد ولذلك نحتاج لسجل لهذا الموظف.

كيفية الإعلام عن السجل:

للتصريح عن السجل نقوم باستخدام الكلمة المحجوزة struct وهي اختصار لكلمة structure ومعناها تركيب.

ونضع جميع مكونات هذا التركيب ضمن قوسين ونختم التصريح بفاصلة منقوطة بعد القوس الثاني، ثم يتم التصريح عن متحول خاص لهذا التركيب (السجل) ضمن القائمة الرئيسية للبرنامج.

ويتم التصريح عن السجل بإحدى الطرق التالية:

```
#include <iostream.h>
struct  employee        اسم السجل
{
---------;
---------;
---------;
---------;
};                      لا ننسى الفاصلة المنقوطة هنا
Void main ( )
{
struct  employee  emp;        هنا نصرح عن متحول خاص بالسجل
```

كما نلاحظ:

فإننا كتبنا السجل وكتبنا بداخله جميع المتحولات الموجودة بداخله ، وثم بدأنا بكتابة البرنامج بالعبارة المعتادة ()void main ثم صرحنا عن متغير خاص بهذا السجل ، ثم نتابع كتابة البرنامج.

كيفية كتابة المتحولات داخل السجل :

```
#include <iostream.h>
struct  employee
{
char name[40];          مصفوفة لكتابة اسم الموظف
char address[40];       مصفوفة لكتابة عنوانه
int age;                متحول لكتابة عمره
float salary;           متحول لكتابة راتبه
};
```

ونصرح طبعا عن متحول خاص بهذا السجل كالتالي:

```
Void main ( )
{
struct  employee emp;
```

كيفية إدخال وإخراج المعلومات داخل السجل :

عندما نريد مثلا إدخال عمر الموظف فإننا نكتب cin>>emp . age;

وعندما نريد إدخال اسم الموظف نكتب cin>>emp . name;

ولانكتب الأقواس عندما نريد إدخال الاسم كاملا كما تعلمنا في المصفوفات

اما عندما نريد إخراج أي قيمة فإننا نستبدل cin بـ cout كما هو معروف

ويمكننا عمل التصريح بطريقة أخرى:

```
Void main ( )
{
Struct employee
{
-----;
-----;
-----;
-----;
};
ونكمل البرنامج
```

مثال عملي :

```
#include<iostream>

#include<string>

Using namespace std;

Struct student

{

String name;

Int age;

Float id;

};

Void main() {

Student
s1……
……
```

اكتب برنامج يقوم بإدخال معلومات عن :

1. اسم الموظف
2. عنوانه
3. عمره
4. راتبه

```
#include<iostream>
using namespace std;
struct employee    هنا لانضع فاصلة منقوطة
{
char name[40];
char address[40];
int age;
float salary;
};
void main()
{
struct employee emp;
cout<<"enter name"<<endl;
cin>>emp.name;
cout<<"enter address"<<endl;
cin>>emp.address;
```

```
cout<<"enter age"<<endl;

cin>>emp.age;

cout<<"enter salary"<<endl;

cin>>emp.salary;

}
```

كيفية التصريح عن مصفوفة السجلات:

بنفس الطريقة السابقة ولكن نجعل المتحول عبارة عن مصفوفة ونضيف سطر برمجي وهو حلقة for كما في المصفوفات.

مثال:

إعادة كتابة البرنامج السابق ولكن لثلاثة عشر موظفا ندخل المعلومات عنهم ثم نخرجها:

```
#include<iostream>

Using namespace std;

struct employee

{

char name[40];

char address[40];

int age;

float salary; };

void main() {

struct employee emp[13];
```

```
for(int i=0;i<13;i++)
  {
    cout<<"enter name"<<endl;
    cin>>emp[i].name;
cout<<"enter address"<<endl;
    cin>>emp[i].address;
    cout<<"enter age"<<endl;
    cin>>emp[i].age;
    cout<<"enter salary"<<endl;
    cin>>emp[i].salary;
  }
for(i=0;i<13;i++)
cout<<emp[i].name<<emp[i].address<<emp[i].age<<emp[i].salary<<endl;
 }
```

مثال آخر :

الإعلان لمصفوفة تراكيب لبيانات شاملة لطلاب مدرسة

```
struct School  {
int rollno ;
```

```
int age ;

char sex ;

float height ;

float weight; }  ;

School student; [ 300 ]
```

هنا ([300] student) هو متغير تركيب وهو يستوعب تركيب طلبة لغاية (300) طالب (أي إن كل طالب ستكون له كامل المعلومات المبينة بالتركيب).

وفي هذه الحالة فإن كل قيد ممكن إن تصل له وتتعامل معه بشكل منفصل مثل أي عنصر مفرد في المصفوفة.

ولكن إذا اردنا توسيع هذه المسألة ، فنحن بحاجة إلى تعريف سجل داخل سجل:

تعريف سجل داخل سجل :

من اجل تعريف سجل داخل سجل من

اجل ارتباط السجل الثانوي بالسجل الرئيسي مثل ان نكتب برنامج يدخل اسم الموظف وعنوانه و..... و..... ويتم إسناد هذا السجل إلى سجل آخر وهو القسم او المؤسسة التي يوجد فيها هذا الموظف .

فيكون سجل المؤسسة هو السجل الرئيسي اما سجل الموظف فهو السجل الثانوي وليس من الضروري في كتابة الكود كتابة أي سجل قبل الآخر.

مثال :

```
Struct employee        سجل الموظف ثانوي
{
Char name[20];
Char address[40];
Int age;
Float salary;

};
Struct dept            سجل القسم رئيسي
{
Int deptno;            نعرف متحول رقم القسم
Int projct;            نعرف متحول المشروع
Struct employee emp;   نعرف هنا متحول السجل الثانوي داخل الرئيسي
};
```

أما عن كيفية الإدحال والإخراج كما يلي:

إذا أردنا مثلا إدخال عمر الموظف فإننا نكتب اسم السجل الرئيسي ثم المتحول للسجل الثانوي ثم مكان الإدخال ، اما إذا أردنا إضافة معلومة ضمن السجل الرئيسي فيكفي كتابة اسم السجل الرئيسي ثم كتابة اسم مكان الإدخال:

Dept.emp.age;

Dept.deptno

استدعاء مصفوفة ضمن سجل – struct and array:

نوع المصفوفة من نوع السجل:

اكتب برنامج بلغة c++ يقوم بطباعة مصفوفة طالب (اسم الطالب – عمره- العلامة) وذلك عن طريق دمج المصفوفات ضمن سجل واحد

```
#include<iostream>
#include<string>
using name space std;
struct student {
```

```
        string name;
         int age , id;
        float degree; };
int main() {
      const s=5;
      student [s];
      for (int i=0; i<5 ; i++)
     { cout<< "Enter name:";
       cin>>student[i] . name;
       cout<< "Enter age:";
       cin>>student[i] . age;
      cout<< "Enter degree:";
       cin>>student[i] . degree; }
      for ( int i=0 ; i<s ; i++)
     { cout<< "Name is" << student[i] . name << "age is" << student[i] . name
<< "degree is" << student[i] . degree << endl;
      return 0;    }
```

في طريقة كتابة البرامج العادية نحتاج لعمل ثلاث أو أربع مصفوفات لكل نوع (الاسم – العمر – المعدل .....) او اكثر حسب المطلوب . ولكن يمكن دمجها من خلال سجل واحد كما رأينا في المثال.

المحاضرة الرابعة

تمرين :

اكتب سجل طالب يحتوي :

الاسم الأول – الاسم الثاني – العنوان – تاريخ الميلاد – الدورات التي سجل فيها.

```
#include<iostream>
#include<string>
using name space std;
struct student {
string f name , l name , SSN;
Date birthday;
Course courses[2]; };
struct Date {
int day , month , year; };
struct course {
string title;
float mark; };
void main ()
{student s1;
S1.name = “Ahmad”;
S1.SSN= “54874”;
S1. Birthdate.day=5;
S1. Birthdate.month=2;
S1. Birthdate.year=1999;
```

```
S1.courses[0].title=”cpp”;
S1.courses[0].mark=90;
S1.courses[1].title=”java”;
S1.courses[1].title=95;
Cout<<”enter name”;
Cin>>s1.name;
Cout<<”enter course[1] title”;
Cin>> S1.courses[1].title
```

التوابع functions

تقسيم البرنامج الى دوال هي احدى المبادئ الرئيسية للبرامج المهيكلة باتباع اسلوب من الاعلى الى الاسفل (Top Down)، وهي مفيدة نظرا لإمكانية استدعائها واستخدامها في اماكن مختلفة في البرنامج.

الدوال (التوابع):

الدوال هي واحدة من كتل البناء الاساسية في لغة ++C، فهي مجموعة من الخطوات (الايعازات) تحت اسم واحد..والدالة تسمح لك بخلق مجاميع منطقية من الشفرات، فهي جزء من برنامج يعمل على البيانات ويعيد قيمة، وكل دالة لها اسمها الخاص وعندما يتم تمييز الاسم في البرنامج اثناء التنفيذ فان البرنامج سيولد تفرع الى الدالة التي تحمل هذا الاسم ليقوم بتنفيذها، وبعد الانتهاء يعود المسيطر الى ذات المكان الذي تفرع منه في البرنامج لاكمال تنفيذ باقي الايعازات.

فوائد استخدام الدوال:

- تساعد الدوال المخزنة في ذاكرة الحاسوب أو التي يكتبها المبرمج على تلافي عملية التكرار في خطوات البرنامج التي تتطلب عملا مشابها لعمل تلك الدوال.
- تساعد الدوال الجاهزة على تسهيل عملية البرمجة نفسها.
- من شأن أستعمال الدوال التوفير في المساحات المطلوبة في الذاكرة.
- ومن شأنها أيضا اختصار زمن البرمجة وزمن تنفيذ البرنامج.
- إمكانية استخدام الدوال مع برامج أخرى تتطلب تنفيذ أو انجاز ذات المهمة.

- عندما يكون برنامج c++ مكون من أجزاء (دوال) مستقلة واضحة المعالم، فإن البرنامج نفسه يكون واضحا لكل من المبرمج والقارئ والمستخدم على حد سواء.

فالتابع هو عبارة عن برنامج فرعي مهمته تنفيذ مهمة معينة حين يتم استدعاؤه.

مم يتألف البرنامج الفرعي :

يتألف من ثلاثة أقسام :

1. التصريح عن البرنامج ، وينتهي بفاصلة منقوطة ولكن إذا كان التصريح خارج (قبل) التعليمة void main() فإن البرنامج الفرعي يكون مشاع (لكل البرامج المتاحة في البرنامج) ، اما إذا تمت كتابته داخل جسم void main() فإن البرنامج الفرعي يكون حكرا على هذه التعليمة .
2. كود طلب البرنامج وينتهي بفاصلة منقوطة.
3. جسم البرنامج وهو يشابه طريقة كتابة البرنامج الرئيسي وطبعا بدون فاصلة منقوطة ويكتب جسم التابع بعد نهاية جسم البرنامج الرئيسي .

كيفية التصريح عن هذا التابع:

الصيغة العامة هي :

```
type  function-name (argument-list)
{ //  codes to execute inside function  }
```

وهو بأن نكتب اسم خرج التابع ثم اسم التابع ومن ثم نكتب نوع البارامترات أي نوع الخرج :

```
خرج التابع   اسم التابع   البارامترات
Void         positive     (int ) ;
```

ويمكن ان يكون خرج التابع :

فارغ void ()

عدد صحيح int

بدون قيمة void

عدد حقيقي float

..... إلخ

ملاحظة://

في الأعلان عن الدوال فان أسماء الوسائط اختياري لانها لاتمثل الأسماء الحقيقية، لذلك يمكن حذف هذه الاسماء والاكتفاء بنوعها فقط مثل

```
float volume ( int ،float ،float ) ;
```

بينما في تعريف الدالة فان الاسماء ضرورية لامكانية الاشارة لها او استخدامها داخل الدالة، مثل

```
float volume ( int a ،float b ،float c )
{
float v = a * b * c ;
return v;
}
```

إعادة القيم return values:

في كل مرة يتم استدعاء الدالة فإن هناك نتائج او مخرجات يجب ان تخرج نتيجة تنفيذ الدالة، فكل الدوال عدا التي من نوع (void) تعيد قيم، وهذه القيم تحدد بواسطة عبارة الارجاع (return).

في ++c إن أي دالة يجب أن تحتوي عبارة الإرجاع التي يجب أن تعيد قيمة، عدا طبعا تلك من نوع void.

أمثلة:

```
int square ( int x   ،int y )
      {   int s = x * y ;
          return s;      }

int square ( int x   ،int y )
      {    return ( x * y ) }

int square ( int x   ،int y )  {
          s = x * y ;
          square = s ;   }
```

ومن الممكن استخدام أكثر من عبارة إعادة في الدالة الواحدة ولكن واحدة منها فقط سوف تنفذ وتنهي البرنامج والأخرى سوف يتم إهمالها:

```
int max (int x   ،int y)
  { if (x > y)
        return x ;
      else
        return y ; }
```

<u>عبارة return لها وظيفتان:</u>

1. تعد مخرجا طبيعيا في نهاية الدالة، وتعيد نتيجة الدالة الى العبارة التي استدعت الدالة في البرنامج.
2. تستعمل لعمليات حساب واستخراج قيم تعابير بداخلها.

مثال بدون الحاجة إلى إرجاع:

بدون الحاجة لكتابة خرج (أي بدون عملية إرجاع)، اكتب برنامج يقوم بمعرفة العدد المدخل من قبل المستخدم ، موجب أم سالب.

```
#include<iostream.h>
void positive (int);   لاتنسى الفاصلة المنقوطة
void main() {
int x;
do
{ cin>>x;
positive (x);   استدعاء البرنامج الفرعي  كود طلب البرنامج
}
while(x !=0);
}
void positive (int a)   البرنامج الفرعي
{ if(a<0)
```

```
cout<<a<<"is a negative number";
else
cout<<a<<"is the positive number ";
}
```

شرح البرنامج:

تم التصريح عن البرنامج بالكود التالي void positive (int); ، ومعناه بأن البرنامج غير مطالب بإرجاع (خرج البرنامج الفرعي) إلى البرنامج الأساسي من خلال التعليمة void ، ثم من خلال الحلقة do-while طالبنا البرنامج بالعمل طالما أن المتحول x لايساوي الصفر، وتم استدعاء البرنامج الفرعي من خلال الكود التالي positive (x);

فبواسطة هذه التعليمة فإن البرنامج سيقفز مباشرة إلى البرنامج الفرعي والذي هو :

```
void positive (int a)
{
if(a<0)
cout<<a<<"is a negative number";
else
cout<<a<<"is the positive number ";
}
```

لقد تم تعريف المتحول a بشكل شكلي وتسند قيمة المتحول الفعلي x المدخلة عليه ومن ثم تم تنفيذ عليه البرنامج الفرعي ، وبعد الانتهاء من البرنامج الفرعي يعود البرنامج لمتابعة البرنامج الرئيسي والمتمثل بإعادة إدخال القيمة x .

ملاحظة :

يوجد طريقتين للتصريح عن التابع :

1. أن نقوم بالتصريح عن التابع ثم كتابة التابع بشكل كامل بعد نهاية main بالكامل أي بعد نهاية القوس الثاني ، ثم نقوم بكتابة جسم البرنامج ويتم استدعاء الدالة ضمن البرنامج .
2. أو نقوم مباشرة بالتصريح عن التابع ويوضع بعد القوسين الصغيرين فاصلة منقوطة ومن ثم كتابة جسم التابع الرئيسي (البرنامج) مباشرة ولانكتب التصريح عن الدالة، ونكتبها بعد نهاية جسم التابع (البرنامج الرئيسي).

عند تنفيذ مثل هذا البرنامج فإن البرنامج سينفذ void main أول شيء

أمثلة :

```
#include <iostream>
using namespace std;
```

```
int square (int i);
int main() {
int i=10;
cout<<"\n"<<(square(i))<<" is the quare value of "<<i<<endl;
return 0; }
int square(int i) { i *=i;
 return i; }
```

مثال : مع الحاجة إلى إرجاع

اكتب برنامج يكتشف اكبر عدد من ثلاثة أعداد مدخلة عليه:

```
#include<iostream>
using name space std;
int great (int,int,int);
void main()
{
int a,b,c,d;
cout<<”enter the first number:”
cin>>a;
cout<<”enter the second number:”
cin>>b;
cout<<”enter the third number:”
cin>>c;
d=great(a,b,c);
cout<<”the great number is:”<<d;
}
```

```
int great(int x,int y,int z)
{
int max;
max=x;
if(y>max)  max=y;
if(z>max)  max=z;
return max;
}
```

مثال:

اكتب برنامج من أجل حساب وطباعة قيمة التعبير التالي:

Sum = 1 + x^1 + x^2 + x^3 + x^4 + x^5

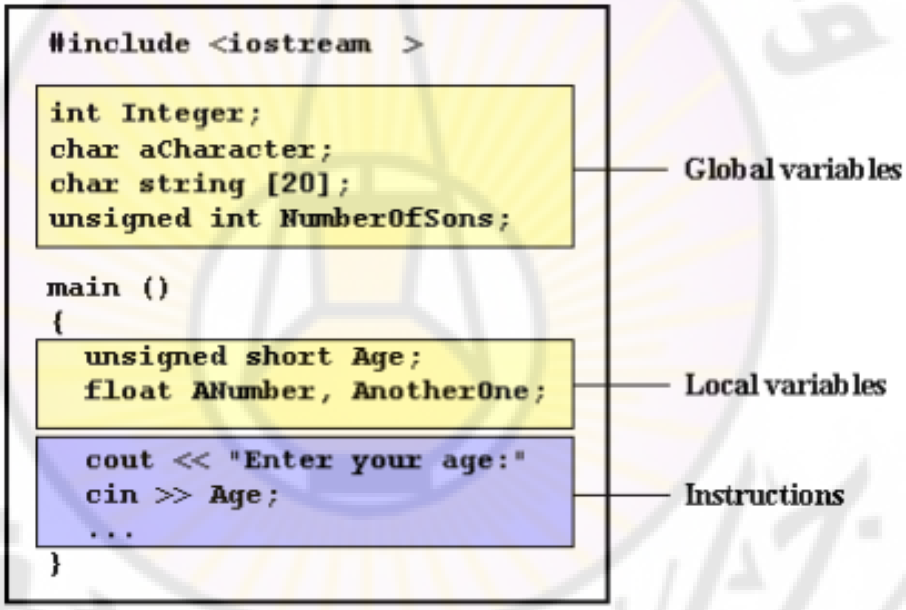
مع العلم ان (x^n) تعني الرقم x مرفوع إلى القوة n ، باستخدام الأمر pow(x,i)

```
#include<iostream>
#include<math.h>
Usin namespace std;
void main()
{
int sum=0,x,i;
 cin>>x;
for(i=5;i>=1;i--)
sum+=pow(x,i);
cout<<sum;
}
```

المحاضرة الخامسة

المتغيرات Scope(Local- Global):

ليس بالامكان تمرير متغيرات الى الدالة فقط ولكن بالامكان الاعلان عن متغيرات داخل جسم الدالة ايضا، وهذا يتم باستخدام المتغيرات المحلية (Local variables) وسميت كذلك لتواجدها محليا في الدالة نفسها فقط، اذ ان هذه المتغيرات سوف لاتستمر فعاليتها بعد الانتهاء من تنفيذ الدالة (اي بعد اعادة القيمة من الدالة). المتغيرات المحلية يتم تعريفها مثل المتغيرات الأخرى.
اما المتغيرات التي تعرف خارج جميع الدوال فلها تاثير عام على كامل البرنامج بكل دالة وتسمى المتغيرات العامة (global variables) وبالرغم من كون المتغيرات العامة متغيرات مقبولة في C++ لكنها غالبا لاتستخدم..



شكل يوضح مدى عمل المتغيرات المحلية والعامة

كما هو واضح فإن المتغيرات العامة تكتب بعد الموجهات وقبل الدالة الرئيسية .

المتغيرات المحلية التي لها نفس الاسم للمتغيرات العامة لاتغير المتغيرات العامة، فاذا كانت هناك دالة فيها متغيرات لها نفس اسم المتغيرات العامة فإن الاسم يشير الى متغيرات محلية وليس العامة عند استخدامها داخل الدالة .

* برنامج يوضح مدى عمل المتغيرات المحلية والعامة وذلك بطباعة المتغيرات العامة والمحلية التي لها نفس التسمية.

```
#include <iostream>

using namespace std;
```

```
 void myFunction() // prototype
{  int y = 10;
 cout << "x from myFunction: " << x << "\n";
 cout << "y from myFunction: " << y << "\n\n";  }
  int x = 5 , y = 7; // global variables
 int main()
 {
 cout << "x from main: " << x << "\n";
 cout << "y from main: " << y << "\n\n";
 myFunction();
 cout << "Back from myFunction!\n\n";
 cout << "x from main: " << x << "\n";
 cout << "y from main: " << y << "\n";
 return 0; }
```

ويكون الخرج على الشكل التالي:

```
x from main: 5
y from main: 7
x from my Function: 5
y from my Function: 10
Back from my Function!
x from main: 5
```

y from main: 7

استدعاء الدالة (التابع):

يقصد باستدعاء الدالة، هي العملية التي يتم فيها الطلب من الدالة لتنفيذ الشفرة الخاصة بها، ويتم ذلك من خلال كتابة اسم الدالة مع القوسين اللذين يحملان الوسائط الواجب تمريرها الى الدالة لتستخدمهما بانجاز عملها.. ويجب ان تلاحظ ان اسم الدالة عند الاستدعاء لايسبق بتعريف النوع .

* برنامج لجمع عددين باستخدام الدوال، يوضح كيفية خزن نتائج الدالة :

```
#include <iostream>

using namespace std;

int addition (int a,int b)

{ int r;

  r=a+b;

  return r; }

int main ()

{

  int z;

  z = addition (5,3);

  cout << "The result is " << z;

  return 0; }
```

وبذلك يكون الخرج 8

- الاستدعاء بواسطة القيمة by value
- الاستدعاء بواسطة المرجعية by reference

الاستدعاء بواسطة القيمة :

في لغة ++C عند استدعاء دالة معينة تحتوي على وسائط فإن عبارة الاستدعاء تمرر متغيرات ذات قيم ، اي ان كل متغير له قيمة وبالتالي فإن المترجم سيعوض قيم هذه المتغيرات في عبارة الاستدعاء كما هو الحال عند التعامل مع اي متغير في البرنامج حيث تعوض قيمتة ويتم التعامل مع القيمة. الدالة المستدعاة تخلق مجموعة جديدة من المتغيرات وبأسماء ليس من الضروري ان تكون ذات الأسماء في عبارة الاستدعاء لان الدالة تأخذ نسخة من قيم المتغيرات في عبارة الاستدعاء وتحملها على المتغيرات التي تقابلها في الدالة المستدعاة. فهنا لا تصل الدالة الى المتغيرات الحقيقية في برنامج الاستدعاء وبامكانها العمل فقط على القيم التي تم نسخها، وليس على القيم الموجودة ضمن الذاكرة ، لذلك عندما تغير قيمة المتغير فإنها لاتؤثر على القيمة الحقيقية للمتغير في الذاكرة.

الاستدعاء بواسطة المرجعية:

استخدام المتغيرات المرجعية في ++C تسمح لنا بتمرير وسائط إلى الدوال بالمرجعية أو الاشارة. اي عندما نمرر وسائط بهذه الطريقة فإن المتغير في الدالة سيسبق بعلامة (&)، هذه العلامة تعني الاشارة الى عنوان الذاكرة الخاصة بهذا المتغير وبالتالي فإن العمل يتم على الموقع الحقيقي للمتغير في الذاكرة لذلك فإن التغيير سيكون دائما في الذاكرة وينسحب الى المتغير في دالة الاستدعاء،

مثال:

برنامج لإخراج الاسم باستخدام المرجعية ومن دون استخدامها:

```
#include <iostream>

#include<string>

using namespace std;

void p(string n)
{ cout<< “name :”<<n<<endl;
        n= “Omar”;
cout<< “name :”<<n<<endl; }
```

```
void main ()
{ string name;
Cout<<”Enter the name:”;
Cin>>name;
P(name);
Cout<<name<<endl; }
```

المخرجات:

إذا قمنا بإدخال اسم Ahmad يطبع

Ahmad

Ahmad

Omar

في حال استخدام المرجعية by refernce يكون الكود على الشكل التالي:

```
#include <iostream>

#include<string>

using namespace std;

void p(string &n)
{ cout<< “name :”<<n<<endl;
        n= “Omar”;
cout<< “name :”<<n<<endl; }
void name ()
{ string name;
Cout<<”Enter the name:”;
Cin>>name;
P(name);
Cout<<name<<endl; }
```

تغيير n يؤثر على name وتكون المخرجات :

Ahmad

Omar

Omar

مثال تطبيقي هام:

اكتب برنامج بلغة ++c بأربع دوال لآلة حاسبة والاستدعاء لكل دالة ضمن البرنامج الرئيسي:

```
#include <iostream>
#include<string>
#include<math.h>
using namespace std;
int sum(int c1,int c2) { return c1+c2 ; }
int sub(int c1,int c2) { return c1−c2 ; }
int mul(int c1,int c2) { return c1*c2 ; }
int div1(int c1,int c2) {
if (c2!=0) { return c1/c2 ; }
else { cout<<"error";
return 1;}}
int main()
{ int x1, x2 ;
 char ch;
cout<<"Enter first number:";
cin>>x1 ;
```

```
cout<<"Enter second number:";
cin>>x2 ;
cout<<"Enter the operator:";
cin>>ch ;
switch(ch) {
case '+':
cout<<"sum="<<sum(x1,x2)<<endl;
break;
case '-':
cout<<"sub="<<sub(x1,x2)<<endl;
break;
case '*':
cout<<"mul="<<mul(x1,x2)<<endl;
break;
case '/':
cout<<"division="<<div1(x1,x2)<<endl;
break;
default:
cout<<"Errorr opration"<<endl;
```

ملاحظة:

من الممكن استخدام حلقة if –else وأيضا يفضل استخدام حلقة do – while من اجل اختيار أكثر من عملية حسابية في خرج واحد وعدم الخروج من البرنامج عند الانتهاء من عملية حسابية واحدة والحاجة لتشغيل البرنامج عند كل عملية حسابية ، وكما يمكن وضع شرط القسمة ضمن البرنامج الرئيسي.

المعاكسة بين المتغيرات في الدوال swap in function

وهي عملية معاكسة متغيرين ضمن الدوال وجعل قيمة المتغير الأول للثاني والعكس والتي نستخدمها كثيرا في كتل البرامج الضخمة عند الحاجة إلى استخدام الدالة أكثر من مرة ولكن بترتيب مختلف للمتغيرات داخلها ، ولحل ذلك نلجأ إلى متغير ثالث بحيث مثلا نقوم بإسناد قيمة المتغير الأول إليه وبذلك يصبح المتغير الجديد يملك قيمة المتغير الأول والمتغير الأول يصبح فارغا(شاغرا) ومن ثم نقوم بإسناد قيمة المتغير الثاني إلى الأول وبذلك يصبح المتغير الثاني فارغ لنعود ونقوم بإسناد قيمة المتغير الجديد (الثالث) إليه ... وبذلك نكون قد قمنا بالتبديل قيمة المتغيرين.

مثال:

```
#include <iostream>
using namespace std;
void sw(int x1, intx2) {
int x3= x1;
x1= x2;
x2= x3; }
int main()
{ int c1 =10, c2=20;
```

```
cout<<c1<<"  "<<c2<<endl;

return 0; }
```

من خلال هذا الحل يتم التغيير ضمن الدالة فقط ولكننا إذا كنا بحاجة التغيير ضمن جسم البرنامج كامل نقوم باستخدام القيم المرجعية :

أي سطر التابع void sw(int& x_1, int&x_2)

التحميل الزائد في الدوال أو التطابق في الدوال Over load functions

تطابق الدوال يشير إلى استخدام نفس الكيان لأغراض مختلفة ، لغة c++ تسمح بتطابق الدوال ، هذا يعني أنه بالإمكان استخدام نفس الدالة لخلق دوال تقوم بإنجاز مهام مختلفة ، وهذا يدعى في البرمجة الكيانية over loaded .

توفر C++ للمبرمج وسيلة جيدة في استعمال أسماء الدوال والأدوات لأغراض متعددة كل منها يرؤدي دورا معينا. وبهذه الطريقة سيكون في البرنامج عدة دوال بنفس الاسم فمثلا الكثير منا يستعمل الكلمة (أقرأ) وهذا الفعل يدل على عمل معين هو فعل القراءة مثل (يقرأ القرأن، يقرأ الكتاب، يقرأ الرسالة، يقرأ المجلة)، وهنا عدة أسماء يمكن تكوينها من الفعل أقرأ لذا نقول أن الفعل اقرأ يدل على قراءة مجموعة من الحالات أو الوسائل . ومن مفهوم البرمجة كلما كانت المعلومات المطلوب من المبرمج معرفتها حول دالة معينة محدودة كلما كان أسلوب البرمجة أفضل. فمثلا لا توجد ضرورة للمبرمج ان يعرف أو يخبر عن ماهية الدالة التي تؤدي الى طباعة نص او عدد صحيح او حقيقي وأنما المترجم يجب أن يميز نوع المادة المراد طباعتها، مثال

```
cout << " This is test " ;
cout <<  12345 ;
```

لذا فأن الدالة)>> cout(تسمى دالة متعددة الأغراض. ويستطيع المترجم أن يميز الدالة من خلال متغيراتها، ان استخدام مفهوم تطابق الدوال سيمكنك من استخدام عائلة من الدوال التي لها نفس الاسم ولكن لكل واحدة منها قائمة وسائط مختلفة كأن تختلف بالعدد او تتشابه بالعدد وتختلف بالنوع جميعا او قسم منها او تختلف بالعدد والنوع.

الأعلان عن الدوال:

* الأعلان عن الدوال:

```
1. int  add (int  a ،int  b) ;
2. int  add (int  a ،int  b ،int c) ;
3. double  add  ( double  x ،double  y) ;
4. double  add (int  p ،double q) ;
5. double  add (double p ،int  q) ;
```

*** استدعاء الدالة:**

يستخدم النموذج 1	cout << add (5, 12) ;
يستخدم النموذج 2	cout << add (15, 5, 10) ;
يستخدم النموذج 3	cout << add (12.5, 2.8) ;
يستخدم النوذج 4	cout << add (54, 12.7) ;
يستخدم النموذج 5	cout << add (0.7611 , 34) ;

أي شروط التحميل الزائد للدوال :

- تغيير ال data type للدالة
- تغيير قيم البارامترات
- تغيير ال data type للبارامترات

مثال:

```
#include<iostream>
using name space std;
void shop ();
void shop (int c);
void shop (int c, char ch);
int main ()
{
shop ();
shop (10);
shop (10, '+');
return 0;}
```

```
void shop () {for (i=0; i<=5; i++)
            {cout<<'*';} }
 void shop (int c) {for (i=0; i<=c; i++)
            {cout<<'*'; } }
void shop (int c, char ch) {for (i=0 ; i<=c ; i++)
            {cout<<ch ; } }
```

شرح البرنامج:

تم إعطاء ثلاث دوال نفس الاسم shop ولكل منها آلية عمل مختلفة والاختلاف بين البارامترات من حيث العدد والنوعية ، تعريف الدوال بانتهاء البرنامج الرئيسي .

خرج البرنامج كما يلي :

*****

****

++++++++++

استدعاء الدالة لنفسها Recursion

في لغة C++ الدالة ممكن ان تستدعي نفسها، مثل هذه الدالة تسمى دالة الاستدعاء الذاتي، بحيث يتم استدعاء الدالة من داخل جسم الدالة أي ان الدالة تستدعي نفسها.

لنأخذ المثال التالي:

```
 #include<iostream>
using name space std;
int f(int c) { if (c>1) // c=5 //5*f(4) //5*4*f (3) // 5*4*3*f (2) // 5*4*3*2*f (1)
            return c* f(c-1);
            else
            return 1; }
int main () {
cout << f (5);
          return 0;  }
```

مخرجات البرنامج = 120

مثال لاستخدام دالة القوة pow :

```
#include<iostream>
using name space std;
int p(int c, int b){
   if (b>=1)
   return c*f(c , b-1) // 5*5*5
    else
  return 1;   }
int main() {
cout <<p(5,3);
        return 0;     }
```

الجواب=125.

يتم استدعاء الدالة على الشكل التالي :

C=5 , b=3

return 5*f(5 , 3-1)

return 5*5*f(5 , 2-1)

return 5*5*5*f(5 , 1-1)

يتوقف استمرار عمل الدالة عند b=1 ويكون الجواب النهائي 125

أمثلة عامة

استدعاء المصفوفة ضمن الدالة array and function :

أي ان المصفوفة كاملة ستكون على شكل بارامتر ضمن الدالة

مثال : اكتب برنامج بلغة c++ لإيجاد مجموع مصفوفة أحادية ومعدلها :

```
#include<iostream>
using name space std;
void add ( int a[] , int size)
   { int s=0 ;
    for ( int i=0 ; i< size ; i++)
      {     s+= s[i];   }
    cout<< "sum is :"<< s << end l;
     cout<< "ave is :"<< s/ size << end l;    }
int main ()
  { int a [10]
for ( int i=0 ; i< 10 ; i++)
{ cin>> (a , 10);     }          نكتب اسم المصفوفة فقط عند الاستدعاء
 add ( a, 10) ;
 return 0;       }
```

سجل ضمن دالة struct and function

أي أن الدالة تأخذ متغير السجل .

مثال : اكتب برنامج يقوم بأخذ قيم الدقائق والساعات وجمعها وإخراج القيمتين بواسطة الدوال :

```
#include<iostream>
using name space std;
struct time {
    int h ; // قيم الساعات
    float m;// قيم الدقائق    } ;
 time add ( time x , time y )
    {   time z;  // عرفنا متحول نوع time
     z . h = x . h + y . h ;
     z . m = x . m + y . m ;
     return z ;    }   // وفق لدقائق والساعات z سوف يعيد قيمة
int main ()  {
     time s1 , s2 ;
   cout << "Enter first time hours"<<end l;
   cin >>  s1 . h ;
   cout << "Enter first time minutes"<<end l;
   cin >>  s1 . m;
   cout << "Enter second time hours"<<end l;
   cin >>  s2 . h ;
   cout << "Enter second time minutes"<<end l;
   cin >>  s2 . m ;
    s3 = add (s1 , s2); //  نقوم هنا باستدعاء الدالة
   cout << "h="<< s3.h<< end l;
   cout << "m="<< s3.m<< end l;
```

```
return 0;     }
```

الدالة تقوم بإرجاع متغير واحد ولكن مع وجود السجل يمكنها ان ترجع السجل وبالتالي أكثر من متغيرلأن السجل بحد ذاته يحتوي عدد من المتغيرات . لان الدالة ترى السجل على أنه متغير واحد.

المحاضرة السادسة

الصفوف Classes

تكتب البرامج عادة لحل مشاكل العالم الحقيقي، مثل متابعة سجلات الموظفين او محاكاة عمل انظمة التسخين وغيرها من الامور الكثيرة، وبالرغم من احتمالية حل المشاكل المعقدة باستخدام البرامج التي تكتب مع الاعداد الصحيحة والحروف فقط، فهي اكثر سهولة للتعامل مع المشاكل الكبيرة والمعقدة اذا ماتمكنت من خلق تمثيل للكيانات التي تتحدث عنها، بكلام اخر محاكاة عمل انظمة التسخين اسهل اذا خلقت متغيرات تمثل الغرف، متحسسات الحرارة، منظمات الحرارة، والمراجل. فكلما كانت هذه المتغيرات اكثر قربا لما يقابلها في الحقيقة كلما كان كتابة البرنامج اسهل.

تعريف الصفوف :

يوجد عدد غير محدد من التعاريف والتفسيرات لمصطلح البرمجة الكيانية ، ولكن من الممكن أن تصف برمجة الكيان على أنها البرمجة المتعلقة بالبيانات والوسائل (الدوال) التي تستخدم تلك البيانات، حيث يتم تسمية البيانات والوسائل بأسم معين هو الكيان ويكون هذا الكيان مكتفيا ذاتيا أي يشكل وحدة برمجية متكاملة.

وحسب هذا التعريف فأن الكثير من البرامج التي تقوم بمهمة معينة وتحتوي على البيانات التي تحتاجها لأداء عملها تسمى كيانا، فمثلا عند رسم صورة أو شكل على شاشة الحاسب فان هذه الصورة تسمى كيانا وبالتالي تستطيع وصف هذ الصورة باللون والحجم والتظليل ومواصفات أخرى .

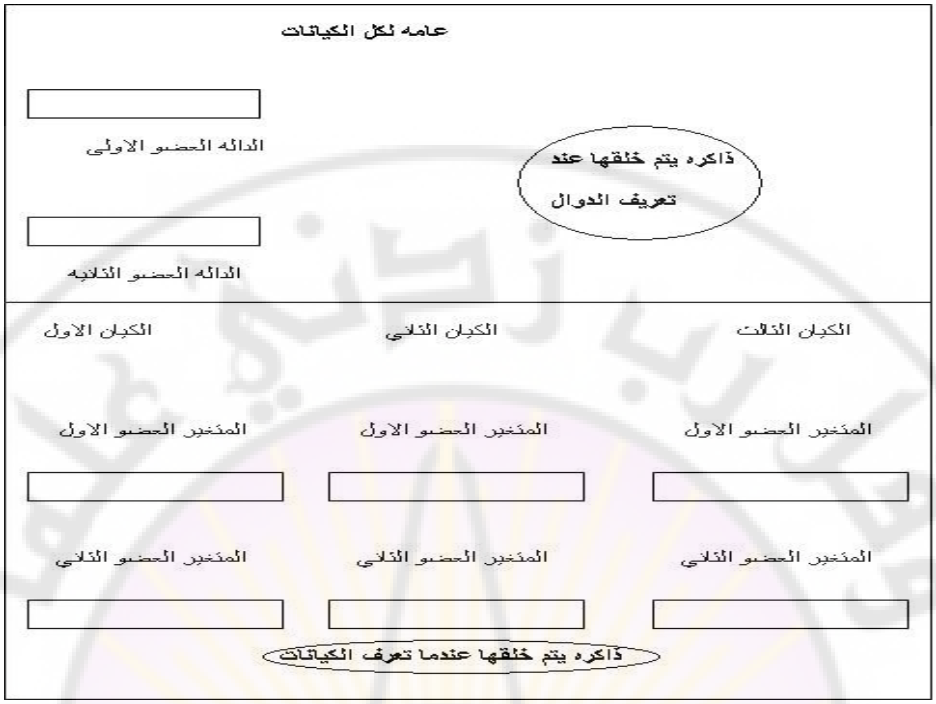
أن وسيلة الصفوف classes تؤدي الى الكيان وهي مشابهة لعملية هيكلة البيانات ولكنها تتصرف بصفات أخرى ، حيث تحتوي هذه الوسيلة على برامج بذاتها تسمى الدوال .

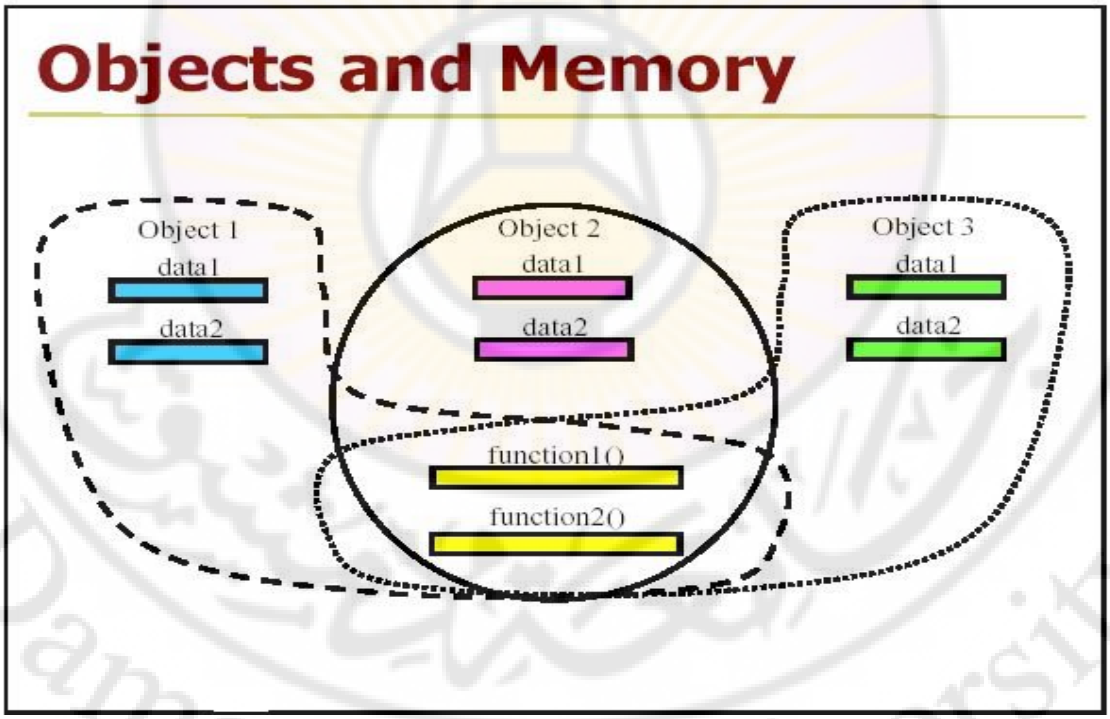
مفهوم الكيان Object Concept

يتكون العالم الحقيقي من كيانات:

- بعضها يكون ملموس- مثلا انت كشخص، هذا الكتاب، سياراتك ، القلم
- وبعضها غير ملموس- مثلا حساب في البنك، المحاضرة... ال

ويتصرف بطريقة معينة . فالكيان هو تجميع للبيانات والطرق او السلوكيات -طرق العمل- على هذه البيانات.. لذلك فان الكيان يغلف او يحزم البيانات والطرق في النموذج البرمجي، وبهذا فان الكيان البرمجي يوفر تمثيلا او تجريدا لكيان العالم الحقيقي.





شكل يوضح كيفية تخصيص مساحات الذاكره للدوال والبيانات للكيانات المختلفة

الصفوف والكيانات Classes and Objects

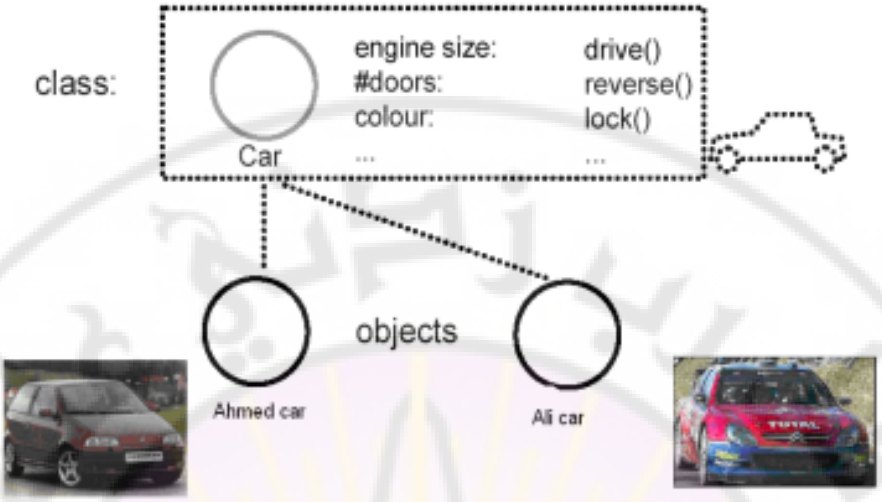
الصنف هو نوع له متغيرات وهي كيانات. الصيغة القواعدية لتعريف الصف كما يأتي:

```
class class_name
{
public :
    member_specification_1
    member_specification_2
        ...
        ...
    member_specification_n
private :
    member_specification_n+1
    member_specification_n+2
        ...
        ...
};
```

الاعضاء العامه

الاعضاء الخاصه

لاتنسى الفارزه المنقوطه



الشكل (9.2): يوضح فكرة انشاء كيانات مختلفة من صنف معين

بالرغم من ان السيارات متشابهة، عندما يتم تصنيعها، ولكنها عناصر مفردة. فكل واحدة من هذه السيارات لها ماكنة خاصة بها، نظام وقود، نظام توقف وهكذا.. فاذا سأل شخص لتشغيل سيارة فانه لايستطيع تشغيل السيارة اذا لم يعرف اي سيارة من السيارات يجب تشغيلها، وفقا لذلك فان كلمة ماكنة مثلا لاتعرف او تحدد ماكنة وحيدة، ولكي تتمكن من عمل ذلك فيجب ان تحدد الماكنة لاي سيارة اي يتم ربط الماكنة بسيارة معينة.

ان الماكنة هي كيان متطور حتى في النماذج الاولية التي صنعت في وقت مبكر، فهي تحتوي على اجزاء عديده ليس من المفروض ان يصلها المستخدم (الذي يقود السيارة)، كمثال، نظام حقن الوقود فان هذا النظام لا يتم الوصول له بشكل مباشرمن قبل المستخدم، وعلى الرغم من ذلك فان هذا النظام يقوم بوظائفة بشكل غير منظور، ولكن من جانب اخر، فان السائق يصل بشكل مباشرة الى المقود، دواسة البريك، او دواسة البنزين وغيرها.. هذه الامور من الممكن ان تشير

اليها على انها واجهات عامة للكيان (سيارة) (Car) وبنفس الطريقة في البرمجة الكيانية، كل كيان يجب ان تكون له واجهة علنية لتكون مفيدة. الكيان بدون واجهة علنية مثل السيارة المصنعة بشكل نموذجي، والتي تكون مغلفة ومغلقة بالكامل بحيث لايمكن لاي شخص ان يدخل بها ويقودها.

الكيانات البرمجية ايضا سوف يكون لها دوال خاصة لها بعض الاغراض التصميمية والتي لايتم الوصول لها بشكل مباشر بواسطة مستخدم الكيان (مثل نظام حقن الوقود الى محرك السيارة). لذلك ففي غالبية الحالات العامة، فان كيان ما هو كيان مغلفا مع واجهات علنية التي لها وصول محدد الى التفاصيل المخفية.

الإعلان عن الصف :

الصف يتكون من أربع أشياء:

1. المتغيرات data type.
2. التوابع functions .
3. بناء كل جزء constructor .
4. هدم كل جزء destructor .

كما سبق وذكر ان الصف هو مجموعة الأجزاء المجردة فمثلا الإنسان يتكون من مجموعة من الأعضاء (data type) وحركة كل عضو (function) وبداية تعلم الحركة (constructor) ونهاية عمل العضو (destructor) أو الذي يحدد نهاية عمل هذا العضو.

مثال : ناخذ Class باسم human كما يلي :

```
Class human {    };
```

اما

```
human. boy;
human. girl;
```

تكون العناصر objects وهي تعريف لكل نوع من الصف او sub class

أنواع العناصر datatype إما ان تكون

عامة public خاصة private مابين العام والخاصprotected

```
#include<iostream>
#include<string>
using namespace std;
Class human
{
Public:
        int b;
        string n;
        void show() { cout<<"hello oop"<<endl; }
private:
        int x, y;
                        };
int main ()
{       human h; // وهو العنصر object
        h.show();
        h.v=12;
        cout<<h.v<<endl;
        erturn0;   }
```

المتغيرات الخاصة يتم استخدامها بطرق غير مباشرة أي إما عن طريق السجل أو الدالة يكون استدعاؤها ، وتكون محدودة الوصول .

ينصح بأن تكون جميع المتغيرات private وجميع الدوال public .

تمرين :

```
        #include<iostream>
#include<string>
```

```
using namespace std;
Class human
{
        private:
                int x , y;
        public:
                void take_x(int v) { x=v ; }
                int get_x() { return x ; }
                void take_y(int v) { y=v; }
                int get_y() { return y ; }
        void main()
        {       human h;
                h.take_x(15);
                h.take_y(20);
                cout<<h.get_x()+h.get_y()<<endl;        }
```

ويمكن إجراء أي عملية حسابية أخرى ... نجد هنا ان الـ x و y لايمكن استخدامها دون الدوال أبدا.

ملاحظة :

الفرق بين الصفوف والسجلات هو أن الصفوف تكون محتوياتها خاصة بشكل افتراضي أما السجلات فتكون عامة وغالبا مانتعامل مع السجل على انه نوع من ال data type لأنه لايمكن ان نجري من خلاله الوراثة وكثير من الأشياء المتعلقة بالبرمجة الكائنية.

مثال :

```
class point:  {
                float x1 , x2 ;
                char op;
        public:
```

```
point() { x1=0.0f ;
          x2=1.0f ;
          op= '+'; }   //constructor هذا التعريف يعتبر الـ
void show() {
        cout<<x1<<"   "<<x2<<"   "<<op<<endl;  }
void set_x1(float v1) { x1=v1 ;}
float get_x1() {return x1 ;}
void set_x2(float v2) { x2=v2 ;}
float get_x2() {return x2 ;}
                 };
Void main () {
        point p1 ; // سوف يعطي القيم الابتدائية
        p1.show() ;
        point p2 ;
        p2.show() ;
        p1. set_x1(1.2);
        p2. set_x2(1.4);
        p2.show() ; }
```

نلاحظ انه سوف يطبع القيم الموجودة في الـ constructor ومن ثم سيقوم بطباعة القيم التي تم إدخالها

(Cat) بحيث بامكانك الوصول الى البيانات الاعضاء هي:

```
class Cat
{
public:
unsigned int itsAge;
unsigned int itsWeight;
Meow();
};
```

هنا فان (itsAge, itsWeight, Meow()) جميعها عامة

يجب عدم الاشتباة ان اخفاء البيانات باستخدام التقنيات الامنية تستخدم لحماية قواعد بيانات الحاسوب، لتوفير مقاييس امنية، ربما على سبيل المثال يحتاج المستخدم الى توفير كلمة مرور قبل ان يوفر لها قاعدة البيانات، كلمة المرور تمنع الاشخاص غير المخولين او المتطفلين من تغيير البيانات او حتى قراءتها احيانا. من جانب اخر، اخفاء البيانات مصممة لحماية المبرمجين ذو القصد الحسن من الوقوع باخطاء المبرمجين. اما الراغبون بشكل حقيقي من الوصول الى البيانات الخاصة فيمكنهم من ايجاد طريقة للوصول الى البيانات الخاصة، ولكن من الصعب عمل ذلك بالصدفة.

9.9 تعريف الكيان Object Definition

ان تعريف الكيان للنوع الجديد هو مشابهة الى تعريف متغير من نوع الاعداد الصحيحة.

```
unsigned int GrossWeight;      // تعريف عدد صحيح بدون اشارة
Cat Nono;                      // تعريف كيان من نوع قطة Cat
```

العبارة الأولى تعرف متغير يدعى (GrossWeight) والذي هو من نوع (unsigned integer) وهذا ماتعودنا عليه في كتابة البرامج المختلفة، كذلك فأن العبارة الثانية عرفت قطة باسم (Nono) والذي هو كيان من الصنف او النوع (Cat), وهنا لاتجد فرق بين الاعلانين فكلاهما يعرف متغير من صنف محدد الاولى حددت متغير من الصنف unsigned int وهو صنف مبني داخليا في لغة C++ وطبعا هذا الصنف له مواصفاته وطريقة العمل عليه سبق وان تعلمناها, اما الحالة الثانية فاننا اعلنا عن صنف من نوع Cat وهذا الصنف يتم كتابتة وتحديدة من قبل المبرمج بحيث تحدد طريقة العمل عليه وصفاته العامة.

المحاضرة السابعة

الصفوف Classes

دوال البناء (constructor) والهدم (diconstructor) في الصفوف

اذا لم يتم الاعلان عن دوال بناء او هدم، فان المترجم يعمل واحدة وهي ماتسمى بدالة البناء او دالة الهدم الافتراضية. دالة البناء والهدم الافتراضية لاتاخذ

اي وسائط ولاتعمل اي شيء. ماالجيد بدالة البناء التي لاتعمل شيئا ؟ جزئيا، هي مسألة شكلية. فكل الكيانات يجب ان تبنى وتهدم، هذه الدوال التي لاتعمل شيئا تستدعى في الوقت المناسب. لذلك، للاعلان عن كيان ما دون ان نمرر وسائط، مثل

```
Cat Rags;        // Rags gets no parameters
```

فانه يجب ان يكون لك بناء على شكل

```
Cat();
```

فعندما تعرف كيان لصنف معين، فان دالة البناء تستدعى. فاذا دالة بناء (Cat) اخذت اثنين من الوسائط، فانه من الممكن ان تعرف كيان (Cat) وذلك بكتابة.

```
Cat Nono (5,7);
```

اما اذا دالة البناء اخذت وسيطا واحدا، فانك تكتب

```
Cat Nono (3);
```

اما في حالة ان دالة البناء لاتاخذ اي وسيط اطلاقا، فاننا نترك او لانكتب الاقواس ونكتب

```
Cat Nono ;
```

وهذا استثناء للقاعدة التي تقول ان الدوال تحتاج الى اقواس، حتى اذا لم يكن هناك وسائط. هذا هو السبب الذي يجعلنا قادرين على كتابة

حتى نستطيع استخدام المتغيرات الخاصة خارج نطاق الصف يكون عن طريق تابع أو كمايلي :

اسم الصف : : اسم الدالة

• برنامج يوضح الاعلان عن دوال البناء ودوال الهدم للصنف Cat

```
// Example 9.6
#include <iostream>
class Cat                    // بداية الاعلان عن الصنف
{
public:                      // بداية المقطع العام
Cat (int initialAge);        // دالة بناء
~Cat();                      // دالة هدم
int GetAge();                // دالة وصول
void SetAge(int age);        // دالة وصول
void Meow();
private:                     // بداية المقطع الخاص
int itsAge;                  // متغير عضو
};
                             // بناء Cat
Cat::Cat (int initialAge)
{   itsAge = initialAge;     }
```

مثال آخر :

```
Class part {
        int x , y ;
public:
        void show () { cout<<x<<y; }
        void add ( part a , part b );
        part sub ( part a);  };
void part : : add (part a , part b ) {
        x= a.x + b.x ;
```

```
        y= a.y + b.y ;  }
part part : : sub (part a) {
        part c;
       c.x = a.x – b;
       c.y = a.y – b ;
        return c ; }
void main () {
part a (12,13) , part b (15,16 ) , c;
c. add (a,b );
c. show () ;  }
```

```
Cat::~Cat()                    // الهدم، لاياخذ اي فعل
{    }
Cat::GetAge()                  // دالة وصول عامة
{      return itsAge;     }
void Cat::SetAge(int age)
{
   itsAge = age;        }
void Cat::Meow()
{   cout << "Meow.\n";    }
        // البرنامج الرئيس يخلق قطة اسمها نونو يحدد موانها ،
        // يعلمنا عن عمرها، وبعدها يجعلها تموء ثانية
int main(){
Cat Nono(5);
Nono.Meow();
cout << "Nono is a cat who is " ;
cout << Nono.GetAge() << " years old.\n";
Nono.Meow();
Nono.SetAge(7);
cout << "Now Nono is " ;
cout << Nono.GetAge() << " years old.\n";
return 0;
}
```

مخرجات البرنامج 9.6:

```
Meow.
Nono is a cat who is 5 years old.
Meow.
Now Nono is 7 years old.
```

إذن الدالة يجب ان تعرف بانها (void او int او float .....) اما دالة البناء لايوضع تعريف أمامها .

مواصفات دالة البناء :

1. تعرف في القسم العام .
2. تستدعى آليا عند خلق كيان جديد .
3. ليس لها أنواع إعادة عند نهاية الدالة return ولاحتى void أي لايمكنها إعادة قيمة .

دوال البناء المتعددة – التحميل الزائد لدوال البناء :

الى الان استخدمت نوعين من دوال البناء، في النوع الاول فان دالة البناء هي التي توفر اسناد البيانات (integer ())، ولاتوجد بيانات تمرر بواسطة البرنامج المستدعي اما الحالة الثانية فان استدعاء دالة البناء يرافقة تمرير قيم مناسبة من داخل الدالة (main ()).

C++ يسمح لك باستخدام النوعين داخل الصنف الواحد.

مثال: من الممكن تعريف صنف كما يأتي:

```
class Integer {
   int m ، n ;
public :
  Integer ( )  {  m = 0 ;   n = 0 ;  }        // constructor 1
  Integer ( int a ، int b ) {  m = a ;   n = b ;   } // constructor 2
  Integer  ( Integer &I )  { m = I .m  ;   n = I.n  ;  }  // constructor 3
} ;
```

هنا تم الاعلان عن ثلاث دوال بناء لبناء الكيان (Integer)، دالة البناء الاولى لاتستلم أي من الوسائط، بينما الدالة الثانية تستلم اثنين من الوسائط من نوع الاعداد الصحيحة، اما الدالة الثالثة تستلم كيانا واحدا من نوع الاعداد الصحيحة

تمرين :

ليكن لدينا الصف التالي :

```
Class Point
{
```

```
        Float x1 , x2 ;
Public:
        Part() { x1=0 ;
                X2=0 ;  }
        Part(int v1 , int v2 ) { x1= v1 , x2=v2 ; }
Void show() { cout<<x1<<x2 ;  }
Void add (part p1 , part p2)  {
                        X1= p1 . x1 + p2 . x1 ;
                        X2 = p1 . x2 + p2 . x2 ;   }  };
Void main ()
        { part p1(12,1) , p2(15,16) , p3 ;
          P3 . add (p1 , p2);
          P3 . show () ;  }
```

المحاضرة الثامنة

البيانات الساكنة

Static Data members

كما لاحظنا سابقا يتم تعريف الدالة العضو بكتابة اسم الصف متبوع بأربع نقاط متعامدة ثم اسم الدالة ووسائطها .

أما الاستدعاء يكون أولا بكتابة اسم الكيان ثم نقطة ثم اسم الدالة . وذلك لأن الدالة المستدعاة تعمل دائما على كيان محدد وليس على الصف بشكل عام .

ملاحظة://

ان الدوال الاعضاء لصنف ممكن ان يتم الوصول اليها فقط بواسطة كيان ذلك الصنف.

ملاحظة://

العامل (::) والذي يوضع بين العضو وصنفة يدعى عامل تحديد المدى (scope resolution operator)، وسمي كذلك لانه يبين المدى او الصنف الذي يعود اليه العضو. ان وضع اسم الصنف قبل النقاط المتعامدة يشبهة اسم الاب، بينما اسم الدالة الذي بعد النقاط المتعامدة يشبه اسم الشخص (الابن)- وسيكون الترتيب مشابهة لاسم الشخص واسم ابيه (اسم الشخص+اسم الاب)

```
// Example 9.1
#include <iostream >
class Cat //          بداية الاعلان عن الصنف
{
public:                 // بداية القسم العام
int GetAge();           //      دالة وصول عامة
void SetAge (int age);  //      دالة وصول عامة
void Meow();            // دالة عامة
private:                // بداية القسم الخاص
int itsAge;             //  متغير عضو
};

int Cat::GetAge()
{    return itsAge;    }
void Cat::SetAge (int age )  // itsAge تعيد القيمة التي يضبط عليها العضو
{
```

```
itsAge = age;              //        itsAge تضبط قيمة المتغير العضو
 }                                   // age الى قيمة تمرر بواسطة الوسيط
void Cat::Meow   ()  //  "Meow"  عملهاالطباعةعلى الشاشة  كلمة
      {     cout << "Meow.\n";    }
                    خلق قطة ضبط عمرها، لها مواء إخبارناعن عمرها، مواء ثانية//
int main()
{
 Cat Nono;
 Nono.SetAge(5);
 Nono.Meow();
 cout << "Nono is a cat who is " ;
 cout << Nono.GetAge() << " years old.\n";
 Nono.Meow();
 return 0;
}
```

مخرجات البرنامج 9.1

```
Meow.
Nono is a cat who is 5 years old.
Meow.
```

البيانات الأعضاء الساكنة :

صفات الأعضاء الساكنة مشابهة لصفات المتغيرات الساكنة ، لها مواصفات خاصة :

1. تنشأ وهي مساوية للصفر .

2. يتم إنشاء نسخة واحدة من ذلك العضو فقط لكامل الصف ويكون مشتركا بين كل البيانات لذلك الصف .

مثال

```
Class part {
    Int x ,y ;
    Static int c=0 ;
    Public :
        Part (x,y) { c++ ; } };
    Int part : : c=0 ;
Void main () {
```

الدوال الأعضاء الساكنة Static member function :

كما في المتغيرات الساكنة يوجد دوال ساكنة لها المواصفات التالية :

1. الدوال الساكنة يمكنها الوصول إلى الأعضاء الساكنة الأخرى فقط (دوال او متغيرات) والمعلن عنها في نفس الصف .
2. الدوال الأعضاء الساكنة تستدعى باستخدام اسم الصف كما يلي :

```
#include<iostream>
class Test    {
int code ;
static int count ;
public :
  void SetCode ( void )
{     code = ++ count ;    }
void ShowCode ( void )
{     cout << " object number : " << code << "\n " ;    }
static void ShowCount ( void )
{   cout << " count : " << count << "\n " ;   }
};
```

```
int Test :: count ;
main () {
Test t1،
t2 ;
t1.SetCode() ;

t2.SetCode() ;
Test . ShowCount () ;

Test t3 ;     t3.SetCode () ;
Test :: ShowCount () ;
T1.ShowCode() ;    t2.ShowCode() ;   t3.ShowCode() ;
return 0;
  }
```

المخرجات :

```
Count : 2
Count : 3
Object number : 1
Object number : 2
Object number : 3
```

الدوال الأعضاء الثابتة :

إذا تم تعريف دالة صف على أنها ثابتة فإن ذلك يفيد بأن الدالة سوف لاتغير قيمة أي من أعضاء الصف . ولغرض تعريف دالة صف أنها ثابتة يجب استخدام الكلمة المفتاحية const بعد الأقواس وقبل الفاصلة المنقوطة وهي لا تأخذ وسائط وتعيد void . مثل

```
void SomeFunction() const;
```

استخدام المصفوفات مع الصفوف

عنصر من الصف على شكل مصفوفة :

كما يلي :

```
class Employee   {
char name [35] ;
float age ;
public :
void getdata ( void ) ;
void putdata ( void ) ;   } ;
```

مثال :

برنامج يوضح كيفية تخزين مصفوفة عناصر داخل الصف لعاملين تعطى بياناتهم بالاسم والعمر :

```
#include<iostream>
class Employee  {
char name [ 30 ] ;
float age ;
public :
void getdata ( void ) ;
void  putdata ( void ) ;
} ;
void Employee :: getdata ( void )
{   cout << " enter name : " ;
cin >> name ;
cout << " enter age : " ;
cin >> age ;    }
void Employee :: putdata ( void )
```

```
{  cout << " name : " << name << "\n " ;
cout << " age : " << age << " \n "     ;         }
cout int size = 3 ;
main ()   {
Employee  manager [ size ] ;
for ( int i = 0 ; i < size ; i++ )
{  cout << "\n  details of manager " << i+1 << " \n " ;
manager[i].getdata() ;   }
cout << "\n" ;
for ( i=o ; i < size ; i++ )
{  cout << "\n manager " << "\n" ;
manager [i] . putdata() ; }

return  0 ;
}
```

المحاضرة التاسعة

الوراثة

Inheritance

الوراثة هي جزء جوهري من البرمجة الكيانية تسمح بإعادة الشيفرة البرمجية ، مثلا يمكننا استخدام الصف أو تحويله مرات عديدة . وهذا مايقلل الوقت والكلفة

تعريف الوراثة :

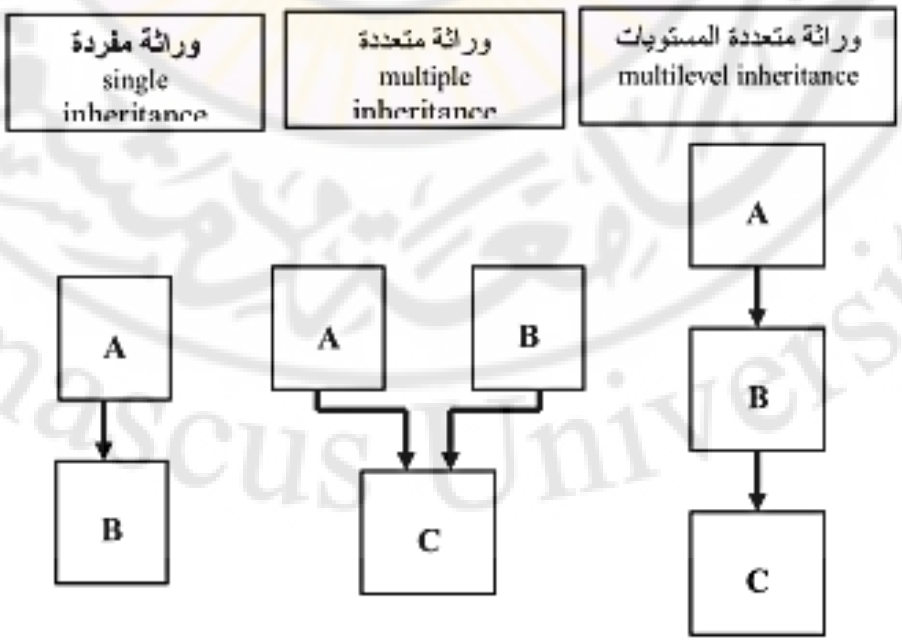
إعادة الاستخدام لاحدى الصفات البرمجية بدلا من إعادة خلق ذات الشيء كل مرة ، وذلك يوفر الوقت والجهد ويزيد الاعتمادية . C++ تدعم بشكل قوي فكرة إعادة الاستخدام ـ أي أنه عندما يتم كتابة الصف مرة ويختبر فمن الممكن أن يكيف بواسطة مبرمج آخر ، أي اشتقاق صف جديد من صف قديم ، الصف القديم يسمى الأساس Base Class أما الصف الجديد يسمى الصف المشتق Derived Class ويمكن للصف المشتق أن يرث من صف واحد أو أكثر .

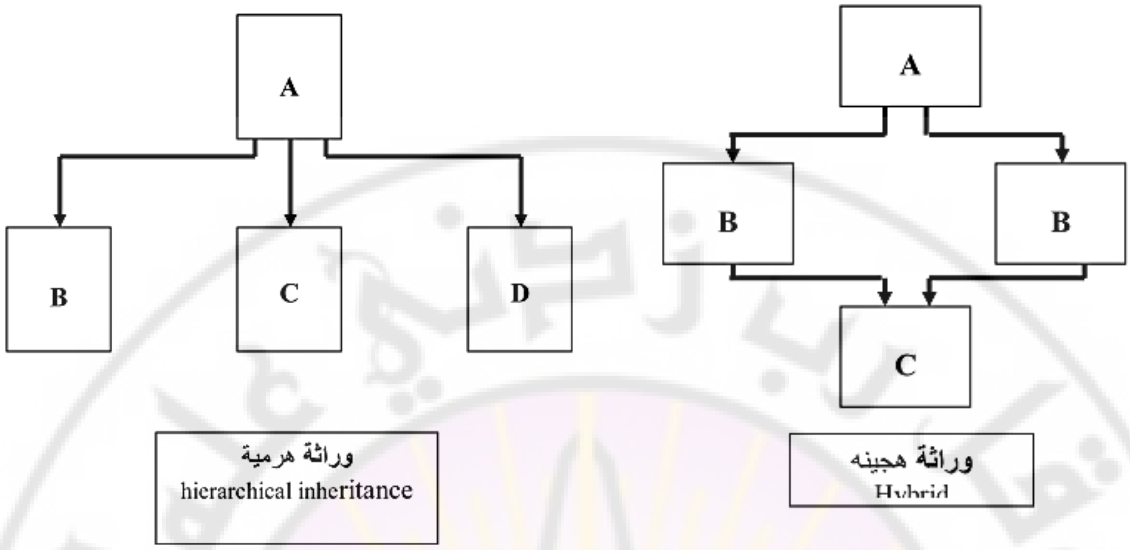
عندما يكون الصف المشتق له صف أساس واحد تسمى وراثة مفردة Single Inheritance

أما إذا كان للصف المشتق أكثر من صف أساس تسمى وراثة متعددة multiple Inheritance .

وعندما تروث صفات صف واحد إلى أكثر من صف مشتق تسمى وراثة هرمية .

أما آلية اشتقاق صف من صف آخر تدعى وراثة متعددة المستويات .





الصيغة القواعدية لاشتقاق صف :

عند الإعلان عن صف يمكن تحديد أي صف مشتق من صف أساس وذلك بكتابة نقطتين متعامدتين بعد اسم الصف المشتق ، ثم نوع الاشتقاق (عام أو خاص أو محمي) واسم الصف الذي يتم الاشتقاق منه (الصف الأساس) والذي يجب أن يكون معلن عنه سابقا .

وعندما يرث صف من صف آخر فإن أعضاء الصف الأساس تصبح أعضاء في الصف المشتق ، الصيغة العامة لوراثة الصف هي :

```
class derived_class_name: access base_class_name {
...
...
...
};
```

حالة الوصول لاعضاء الصنف الاساس داخل الصنف المشتق تحدد بمحددات الوصول. محددات وصول الصنف الاساس يجب ان تكون اما عامة، محمية، او خاصة (public, protected, or privat). اذا لم يكن محدد الوصول موجود عند الأعلان عن الصنف المشتق فان محدد الوصول سيكون خاص بالافتراض.

ان C++ تميز بين نوعين من الوراثة (العام والخاص) (public and private). بالافتراض فان الاصناف تشتق واحدة من الاخرى على انها خاصة، ولذلك يجب ان تخبر المترجم خارجيا بنوع الوراثة العامة اذا اردت ان تكون الوراثة عامة وليست خاصة.

فالصيغة العامة للوراثة : اسم الصف الأساس : اسم الصف الوارث

مثال عام :

ليكن لدينا الصف التالي :

الصف الأساسي	#include <iostream> #include <string> using namespace std; class A { protected: int x,y; public: void show() { cout<<x<<" "<<y<<endl; } };
الصف الموروث لكي تتم عملية الوراثة يجب ان تكون المتغيرات الخاصة في التابع الأساسي محمية	class B : public A { int y; B() { y=10 , x=12 ; } } ;
التابع الرئيسي	void main() { B b; b.show(); }

في الخرج سوف يظهر قيم الصف A لأن الصف الموروث يرث الصف الأساسي بالإضافة إلى القيم والتوابع المضافة داخله .

المتغيرات المحمية protected يمكن استخدامها في مكانين فقط وهما الصف الأساسي والصف الوارث .

<u>مثال توضيحي :</u>

```
#include <iostream>

#include <string>

using namespace std;

class A {

protected:
```

```
        int x,y;
public:
        A()  { x=0 , y=0 ; }
        A( int v1 , int v2 ) { x= v1 , y= v2 ; }
        void show() { cout<<x<<"   "<<y<<endl; }  };
class B : public A  {
         int y;
public :
         B() { y=10 ; }
         B(int c) { y=c , A() {x=2; }
         void s () { cout<<y <<x <<; }  } ;
void main()  {
       B b;
       b.(); }
```

البناء أولا يجب أن يكون تابع فارغ . الصف الأساسي الذي سوف يورث يجب أن يكون فيه دائما دالة بناء ..

نقوم عند الصف الذي سوف يرث بوضع اسم الصف الأساسي كما يلي : A() {x=2; } أي أنه يمكننا تغيير قيم المتغيرات عند هذه المرحلة لأن الخرج سوف يستخدم عناصر الصف الرئيسي وللوصول إليها نذهب إلى التابع الرئيسي main وهناك يمكننا وضع B b(5) وبذلك سوف يضع 5 للمتغير y و 2 للمتغير x.

الوراثة المتعددة Multiple Inheritance

في لغة ++C من المحتمل جدا أن يرث صف أعضاء من أكثر من صف واحد وهذا يتم بفضل الصفوف الأساسية المختلفة بواسطة فاصلة في إعلان الصفوف المشتقة .

مثال : بفرض لدينا صف للطباعة على الشاشة باسم CO ونريد ان نوجد صف باسم CR و CT وكان المطلوب لهذين الصفين وراثة أعضاء الصف الساسي بالإضافة إلى ماهو موجود في الصف CP يمكن كتابتها كما يلي :

```
Class CR : public CP , public CO ;
```

Class CT : public CP , public CO ;

هنا يمثل بالصف CP والذ يشتق من CT , CR

تمرين : إيجاد مساحة مثلث ومستطيل يرثان من صف آخر :

```
#include <iostream>
using namespace std;
class Polygon {
  protected:
   int width , height;
  public:
   void set_values (int a , int b)
     { width=a; height=b;}
  };
class Rectangle: public Polygon {
  public:
   int area ( )
     { return (width * height); }
  };
class Triangle: public Polygon {
  public:
   int area ( )
     { return (width * height / 2); }
  };
 int main () {
  Rectangle r;
```

```
  Triangle t;
  rect.set_values (4, 5);
  trgl.set_values (4, 5);
  cout << r.area( ) << endl;
  cout << t.area( ) << endl;
  return 0;
}
```

الخرج : 20 10

إن محدد الوصول (protected) مشابهة لمحدد الوصول (private) الفرق الوحيد في الوراثة حيث عندما يرث صف من صف آخر فإن أعضاء الصف المشتق بإمكانها ان تصل إلى الأعضاء المحمية الموروثة من الصف الأساس لكنها لايمكن أن تصل إلى أعضائه الخاصة .

مثال، اذا كان الصنف ابنة (daughter) مشتق من الصنف الام (mother) والتي تعرف كما يأتي:

```
class daughter: protected mother;
```

هذه سوف تحدد (protected) كأعلى مستوى وصول لاعضاء الصنف (daughter) والتي ورثت من الام (mother). وعليه، كل الاعضاء التي هي عامة في الصنف الام سوف تصبح محمية في الصنف الابنة. وبالطبع هذا لايقيد الصنف الابنة من الاعلان عن اعضاء خاصة بها من النوع العام، لذلك فان مستوى الوصول الاعظم يحدد فقط للاعضاء الموروثة من الام.

مثال : برنامج لإيجاد نتيجة رفع عدد معين إلى أس معين مع مراعاة الإعلان عن الصف المشتق على أنه عام :

```
#include <iostream>
class powe {
  float x;
public:
 void set (float s) { x = s; }
```

```
 void ptwo() { cout<<"\n"<<x<<" to power 2: "<<x*x; }
  };
class mpow : public powe {
  float y;
 public:
  mpow (float p) { y = p; }
  void pthree() { cout<<"\n"<<y<<" to power 3: "<<y*y*y; }
};
int main() {
  mpow ob(1.3);
  ob.set(10);
  ob.ptwo();
  ob.pthree();
return 0;
}
```

نفس البرنامج السابق مع الإعلان على ان الصف المشتق خاص :

```
#include <iostream>
class powe {
   float x;
public:
 void set (float s) { x = s; }
 void ptwo() { cout<<"\n"<<x<<" to power 2: "<<x*x; }
  };
class mpow : private powe {
```

```
  float y;
 public:
  mpow (float p) { y = p; }
  void pthree() { cout<<"\n"<<y<<" to power 3: "<<y*y*y; }
};
int main() {
  mpow ob(1.3);
  ob.set(10);
  ob.ptwo();
  ob.pthree();
return 0;
}
```

مثال للوراثة المتعددة :

```
class A {
protected:
        int a;
public:
        void p() { cout<< “a is “ <<a<< endl; }   };
class B {
protected:
        int b;  };
class C  {
protected:
        int c ; } ;
```

```
class D : public A , public B , public c {     };
void main ()
{ D dd;
        dd. (سوف تظهر محتويات جميع الصفوف ويمكن استخدامها)
```

<u>الوراثة المتعددة بشكل مستويات كما في المثال التالي :</u>

```
#include <iostream>
#include <string>
using namespace std;
class A {
protected:
        int a1;
public:
        void print () { cout<< "a1 is" <<a1<<endl; }  };
class B : public A  {
protected :
        int a2 ;
public :
void print () { cout<< "a2 is "<< a2<<endl; }  };
class C : public B {      // سوف يرث من  الصفين
        int a3; } ;   //يعتبر محمي لانه وارث من صف محمي
void main ()  {
        A aa;
        aa. print () ;
        C cc ;
```

```
cc. print () ; }
```

هنا سوف يطبع a_1 و a_2 كلاهما .....