بسم الله الرحمن الرحيم

ماهو مفهوم برمجة الكائنات (Object-Oriented Programming)

قبل أن أبتدأ في مفهوم برمجة الكائنات نحتاج أن نضع قاعدة مهمة لكي نستطيع أن نفهم ماهو مفهوم برمجة الكائنات(Object-Oriented Programming)

Every things is an Object كل شيء عبارة عن كائن

لقد وضعنا هذا الإفتراض لأن كل شي تراه بالعين المجردة والغير مجردة عبارة عن كائن

لماذا إفترضنا أن كل شي عبارة عن كائن ؟
إفترضنا هذا الإفتراض لأن كل كائن يتكون من:

1. خصائص (Properties, Attributes)
2. أفعال (Action, Methods, Behaviour)

والآن بما أننا إعتبرنا أن كل شيء عبارة عن كائن فذلك يدل على أن كل شي له خصائص و أفعال

خصائص الكائن (Properties, Attributes):

هي كل شي بالكائن ولا تفارقه أبدا (هي مواصفات الكائن) وأقرب مثال على ذلك {الإنسان} لنرى ماهي خصائص الإنسان:

خصائص الإنسان	
إسم الخاصية	قيمة الخاصية
الإسم	محمد
العمر	٢٥ سنة
الطول	١٦٠ سنتيمتر
الوزن	٧٠ كيلو

لقد ذكرنا بالسابق بعض الخصائص الخاصة بالإنسان (هل هذه الخصائص تفارق الإنسان ؟).
فهذا بدل على أن كل شي تضع عليه عينيك يعتبر كائن
وسوف أذكر تعريف توضيحي أخير

كل شي يأتي على صيغة (Name = Value) يعتبر خاصية من خصائص الكائن

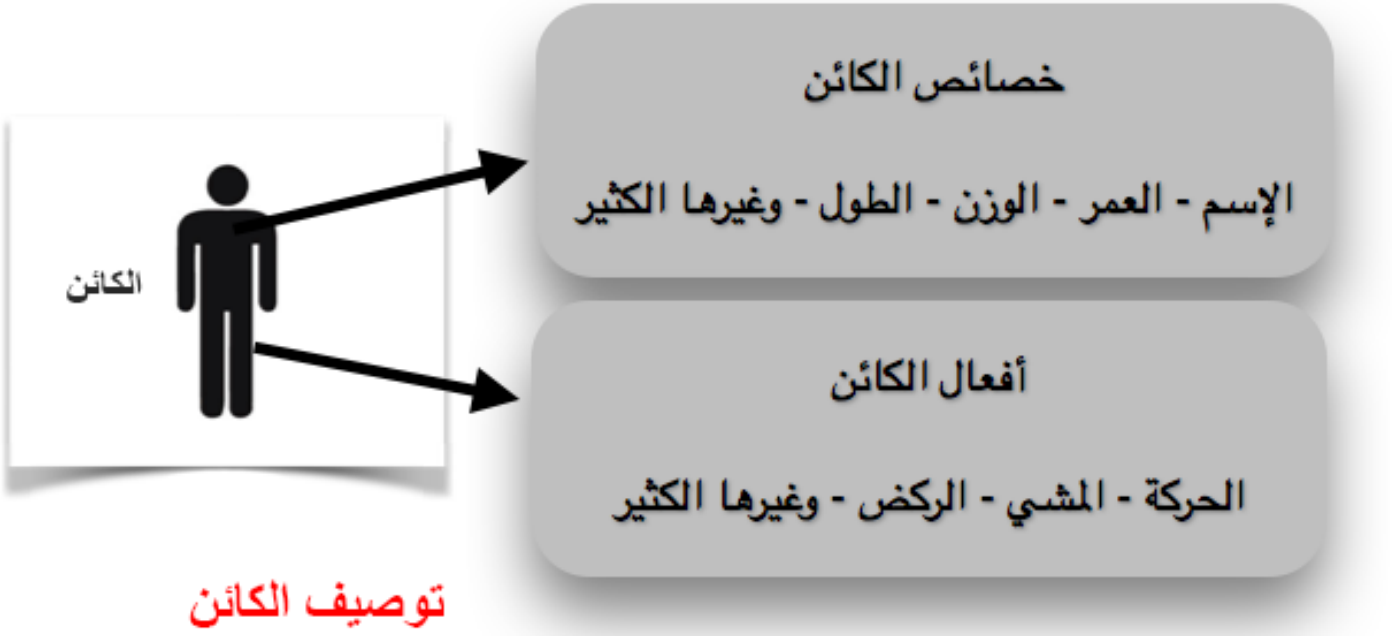
أفعال الكائن (Action, Methods, Behaviour):

هي كل ما يستطيع القيام به الكائن
بما إننا إفترضنا أن الإنسان كائن وله خصائص فلابد من أن له أفعال وأفعاله كالتالي:

- الحركة
- المشي
- الركض

ويوجد غيرها الكثير من الأفعال

في هذا الرسم نختصر كل الكلام المكتوب إعلاه



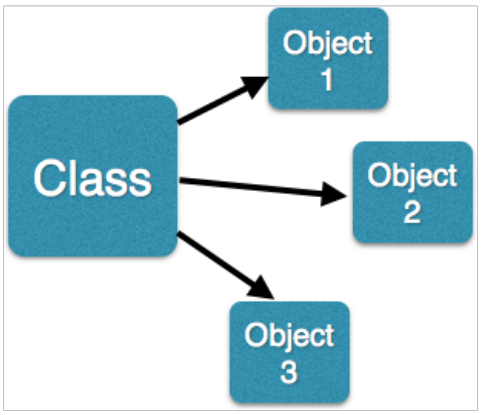
لو تلاحظ أننا إلى الآن لم نتحدث عن أي شي له علاقة بالبرمجة فسوف يكون سؤالك ماهي العلاقة بهذا الكلام في البرمجة

تمثيل مفهوم برمجة الكائنات في البرمجة:

لكي نقوم بربط مفهوم الكائنات بالبرمجة نحتاج أن نفهم ماهو الـ(Class) و (Object)

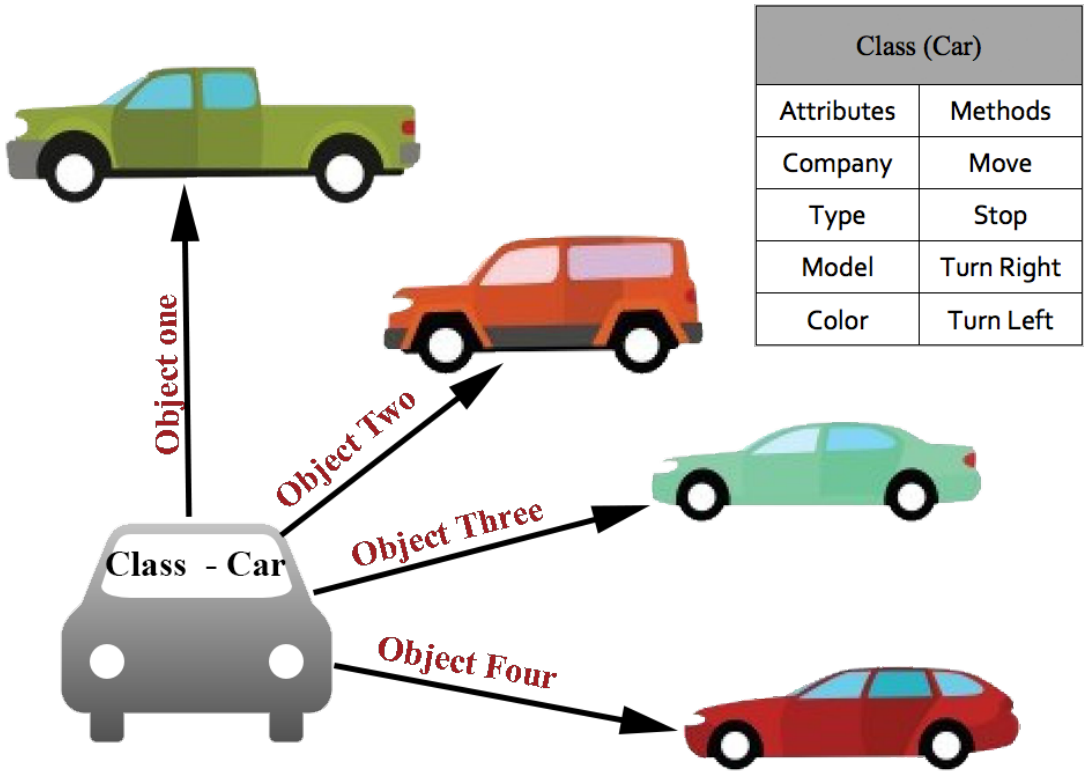
الـ(Class): هو عبارة عن كائن.
الـ(Object): هو عبارة عن كائن

كلها نفس المعنى ماهو الفرق بينهم !!!!!!



من الصورة أعلاه نفهم أن الـ(Object) عبارة عن نسخه من الـ(Class)

مثال توضيحي:



Class (Car)	
Attributes	Methods
Company	Move
Type	Stop
Model	Turn Right
Color	Turn Left

كيف تتعامل لغات البرمجة مع الـ(Classes) و الـ(Objects) ؟

لكي تتعامل لغات البرمجة مع الـ(Classes) و الـ(Objects) نحتاج لتمثيل الخصائص والأفعال كالتالي:

- الخصائص (Properties, Attributes) — المتغيرات (Variables)
- الأفعال (Action, Methods, Behaviour) — الدوال (Function)

والآن أصبح الـ(Class) و الـ(Object) عبارة عن مجموعة من المتغيرات والدوال

تنبيه: لا بد من إنشاء الـ(Class) لكي نتمكن من إنشاء الـ(Object)

مثال برمجي يطبق البرمجة بالكائنات:

```
1
2   #include <iostream>
3
4   Class Car {
5
6       String Company;
7       String  Type;
8       int Model;
9       String Color;
10
11
12
13      public void Stop()
14      {
15          .......;
16      }
17
18      public void Move()
19      {
20          ......;
21      }
22
23      public void Turn_Right()
24      {
25          ......;
26      }
27  };
28
29  int main()
30  {
31  Car Car1; //First Object
32
33  Car Car2;  // Second Object
34  }
```

Start Class

Variables

Function

Class

Objects

نستنتج من هذه الصورة أن الـ(Class) يتكون من متغيرات ودوال والـ(Object) عبارة عن نسخة من الـ(Class) هذا مثال على لغة C++

وفي الختام أتمنى أن أكون وفقت في توصيل مفهوم برمجة الكائنات للجميع

المتغيرات المرقمة والمصفوفات
Arrays and Matrices

مقدمة introduction

أن طرق التعامل مع أسماء المتغيرات والثوابت العددية والرمزية ، التي وردت في الفصول السابقة ، تعد صالحة للتعامل مع عدد محدود من هذه الثوابت والمتغيرات ، سواء في عمليات الإدخال والإخراج أو في العمليات الحسابية والمنطقية ، وعندما يصبح عدد المتغيرات كبيرا جدا ، تصبح تلك الطرق غير عملية ، فمثلا لو أردنا إدخال مائة قيمة للمتغيرات x2,x1- .... إلى x100 ، فكم الحيز المطلوب من البرنامج لعمليات الإدخال والإخراج والعمليات الحسابية والمنطقية لهذه المتغيرات ؟ هذا من جهة ، ومن جهة أخرى : فأننا نوفر مخزنا خاصا لكل متغير نتعامل معه ، أثناء تنفيذ البرنامج ، ولذلك لحفظ قيمته في مخون ، ومن ثم لاستعمال قيمته في عمليات أخرى تالية ، ومن ناحية ثالثة ، فان من الصعوبة بمكان ، بل من المستحيل استعمال اسم المتغير العددي أو الرمزي كمصفوفة ذات بعدين ، وثلاثة أبعاد ...الخ

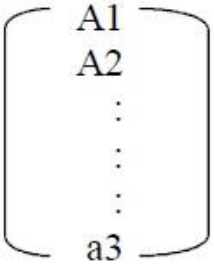
للأسباب الثلاثة الواردة أعلاه ، جاءت فكرة استعمال متغير جماعي يضم تحت اسمه عددا من العناصر يسمى بالمتغير الرقمي subscripted variable ، ويتم ترقميه بين قوسين مربعين [ ] يوضع بينهما قيمة العداد المرقم subscript ، وقد تسمية الدليل index أحيانا ، ويمكننا تشبيه المتغير المرقم بقسم الهاتف لمؤسسة ما ، فهو مقسم واحد ، تنظم تحته عدد من الأرقام الفرعية للموظفين وكل رقم من هذه الأرقام مستقل ومتميز عن الأرقام الفرعية الأخرى ، وله مخزن خاص في الذاكرة ، الآن انه كغيره من الأرقام الفرعية تابع للرقم العام لمقسم المؤسسة ، كما يمكن تشبيه المتغير المرقم بالجيش الذي يعامل كاسم متغير واحد ، لكن يضم عددا كبيرا من العناصر ، فمثلا العناصر التالية :(من اليمين إلى اليسار):

A[n] ...a[2], a[1], a[0]

تابع للمتغير الجماعي a[ ]
وكل عنصر من هذه العناصر له عنوان في الذاكر address ، فالعنوان الأول يكون للعنصر الأول والثاني للثاني والثالث للثالث ... وهكذا.
ويستعمل المتغير الجماعي [المرقم] أو المصفوفة ، في لغة c++ وغيرها ، حجز جماعي مسبق في الذاكرة لجميع عناصره ، فلو كان يتبعه خمسون عنصرا ، فانه يحجز له 50 مخزنا ، على الأقل في الذاكرة .

من الفوائد المهمة للمتغيرات المرقمة والمصفوفات : هو استعمالها في الترتيب التصاعدي والتنازلي للعناصر والقيم المختلفة ، وعمليات ترتيب الأسماء الأبجدي

النصوص الرمزية ، وفي عمليات ضرب المصفوفات ، وإيجاد معكوس المصفوفة وعملياتها الأخرى ، وفي التحليل العددي ... الخ.
المتغير المرقم (المصفوفة) ذو البعد الواحد one-dimensional Array
المتغير المرقم ذو البعد الواحد هو مصفوفة ذات بعد واحد أو متجه (vector) ويمثل في الجبر على النحو الأفقي [a1 a2 ...a3]
أو العمودي



ويأخذ المرقم المتغير في c++ الشكل العام التالي:

```
Type-specifier array-name[size];
```

ويبدأ العداد المرقم عادة من الصفر ، آي أن العنصر الأول من المصفوفة []a هو a[0] والثاني a[1] ... وهكذا فمثلا المصفوفة التالية:

Int a[20];

اسمها a ، وقد حجز لها 20 موقعا لعشرين عنصرا من النوع الصحيح .

والمصفوفة التالية:

Char name[15];

مصفوفة رمزية ، اسمها name يحجز لها خمسة عشر عنصرا من النوع الرمزي لها .

وهكذا...

عنوان عناصر المصفوفة في الذاكرة Addressing Array Elements in Memory

ذكرنا من قبل أن آي متغير أو عنصر من متغير ذاتي مرقم ، يحتل موقعا من الذاكرة يستعمل عادة مؤشرا لكل متغير أو عنصر ، ليكون دليلا على استعمال هذه المتغيرات والعناصر بسهولة ويسر ، والمثال التالي يوضح هذه العملية بالنسبة للمصفوفة ذات بعد واحد.

Int x[5];

يمكن تمثيل عناصر المصفوفة x المعلن عنها ، مع عناوينها بالشكل التوضيحي التالي (من اليسار إلى اليمين)

الأول	الثاني	الثالث	الرابع	الخامس
X[0]	X[1]	X[2]	X[3]	X[4]
100	101	102	103	104

امثلة حول استخدام المصفوفه ذات البعد الواحد ..

مثال1/ عملية إدخال ذاتي لقيم عناصر متغير مرقم مصفوفة ذي بعد واحد

```
#include < iostream.h>
main ()
{
int a[20];
int I;
for (I=0;I<20;++I)
 {
    a[I]=I+1;
 }
}
```

في هذه الحالة يتم إدخال عشرين عنصرا من عناصر المصفوفة a
A[0]=1 عندما يكون I=0
A[1]=2 عندما يكون I=1
...
...
...
a[19]=20 عندما يكون I=19

مثال2 / مصفوقتين ضرب مصفوقة في مصفوقة اخرى .

```
#include < iostream.h>
main ()
{
int x[5], y[5];
int I;
for (I=0;I<5;++I)
{
x[I]=I;
y[I]=I*I;
cout<<endl<<x[I]<<y[I];
}
}
```

وستكون قيم النتائج على النحو التالي:

0	0
1	1
2	4
3	9
4	16

إعطـاء قيمة أولية للمصفوفة ذات البعد الواحد Array Initialization

مثال على إدخال عدة عناصر من مصفوفة الدرجات grade[]

```
Int grade[5]={80,90,54,50,95}
```

ومثال على إدخال قيم عناصر المصفوفة الرمزية name[]

```
Char name[4]="nor"
```

لاحظ أن المتغير المرقم name[] مكون من أربعة عناصر بينما تم إعطاؤه ثلاثة عناصر فقط والسبب أن العنصر الرابع بالنسبة إلى المعطيات الرمزية يكون خاليا.

مثال3 / ادخال قيم الى المصفوفة مباشرتنا بدون استخدام دالة القراء.

```
#include < iostream.h>
main ()
{
int a[6]={40,60,50,70,80,90}
int I;
for(I=0;I<6;I++)
{
cout<<a[I]<<endl;
}
}
```

والناتج طبعا سيكون كالتالي :

40
60
50
70
80
90

مثال٤:
قم بكتابة برنامج يقوم بإيجاد مجموع ، ومعدل علامات الطالب في 5 مواد وهذه العلامات كالتالي:
87,67,81,90,55

```
#include < iostream.h>
main ()
{
int i;
int a[5]={87,67,81,90,55}
int s=0;
float avg;
for(i=0;i<5;i++)
{
s=s+a[i];
}
 avg=s/5;
cout<<avg<<endl;<<s<<endl;
}
```

والناتج سيكون كالتالي:

87
735

المعدل 87
والمجموع 735

مثال5/ ادخال بيانات المصفوفه عن طريقة داله الادخال.

```
#include < iostream.h>
main ()
{
int i;
int a[5];

for(i =0; i<=4;i++)
  {
    cout<<"enter Numbers Here:"<<endl;
     cin>>a[i];
  }

}
```

مثال6/ ادخال بيانات المصفوفه عن طريقة داله الادخال ثم طباعه الارقام المدخله.

```
 #include < iostream.h>
main ()
{
int i;
int a[5];

for(i =0; i<=4;i++)
  {
    cout<<"enter Numbers Here:"<<endl;
     cin>>a[i];
  }

for(i=0;i<=4;i++)
{
    cout<<a[i]<<endl;
}

}
```

مثال7/ ادخال بيانات المصفوفه عن طريقة داله الادخال وجمع القيم المدخله واخرج المعدل .

```
#include < iostream.h>
main ()
{
int i;
int a[5];
int s=0;
float avg=0.0;
for(i =0; i<=4;i++)
  {
    cout<<"enter Numbers Here:"<<endl;
     cin>>a[i];

  }

for(i=0;i<=4;i++)
{
  s=s+a[i];
}
   avg=s/5;
cout<<"Sum ="<<s;
cout<<"Avg = "<<avg;
}
```

مثال 8/ادخال سبع قيم اولية للمصفوفه ومعرفه عدد القيم التي اكبر من 50 واصغر من 50 ؟

```
#include < iostream.h>
main ()
{
int i,S,L;
int a[5]={87,67,41,90,55,53,38}
for(i =0; i<=6;i++)
  {
    if(a[i]>=50)
     {
          L=L+1;
     }
    else
     {
        S=S+1;
      }
  }
   cout<<" 50 اكبر من "<<L<<Endl;
   cout<<" 50 اصغر من "<<S<<Endl;

}
```

مثال 9/ ادخال عشرة قيم اولية للمصفوفه ومعرفه عدد القيم السالبه وعدد القيم الموجبة ؟

```
#include < iostream.h>
main ()
{
int i,N,P;
int a[5]={87,-167,141,90,55,53,38,-4,-2, 1}
for(i =0; i<=9;i++)
  {
    if(a[i]>=0)
     {
          P=P+1;
     }
    else
     {
        N=N+1;
      }
```

```
 }
   cout<<" الاعداد الموجبة "<<P<<Endl;
   cout<<" الاعداد السالبة "<<N<<Endl;

}
```

H/W:

1- ادخال عشرة قيم اولية للمصفوفه ومعرفه اكبر قيمه بينهما ؟
2- ادخال عشرة قيم اولية للمصفوفه ومعرفه اصغر قيمه بينهما ؟
3- ادخال عشرة قيم اولية للمصفوفه وتبدل قيمه العنوان الاول A[0] بالعنوان الاخير A[9] ثم طباعه المصفوفه بعد التدبيل ؟

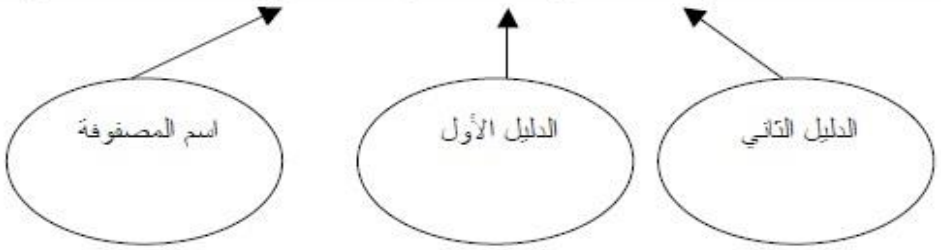
4- ادخال خمسه قيم اولية لمصفوفه a[5] ثم نقوم بازاحه قيم المصفوفه نحو الامام بحيث قيمه العنوان الاول تكون بدل قيمه العنوان الثاني ، والثاني بدل الثالث ، والثالث بدل الرابع وهكذا..... ، والاخير بدل الاول ؟

5- ادخال عشرة قيم اولية للمصفوفه وجمع قيم العناونين الزوجية للمصفوفه؟

Two-Dimensional Arrays المصفوفة ذات البعدين

تشبة المصفوفة ذات البعدين في طريقة تعاملها ، المصفوفة ذات البعد الواحد إلا أن لها عداديـن(index2) دليليـن أو مرقميـن إحداهمـا عـداد للصـفوف ، والأخـر عـداد للأعمدة ويأخذ الإعلان عن المصفوفة الشكل العام التالي:

```
Type-specifier arraey_name [index 1][index 2];
```



فمثلا المصفوفة :

```
Int x[2][3];
```

وهي مصفوفة صحيحة العناصر int أبعادها هي عدد الصفوف=2 ، وعدد الأعمدة =3

لاحظ أن عدد الصفوف يوضع بين قوسين وحده ، وكذلك عداد الأعمدة .

مثال / شاهد هذا المثال الذي يستخدم 5 صفوف و 3 اعمدة:

```
#include <iostream.h>
main ()
{
int m[5][3];
int I,j;
for(I=0;I<5;I++)
 {
for(j=0;j<3;j++)
  {
    cin>>m[I][j];
  }
 }
}
```

مثال / شاهد هذا المثال الذي يستخدم 5 صفوف و 3 اعمدة قراء القيم ثم طباعة القيم المدخلة:

```
#include <iostream.h>
main ()
{
int m[5][3];
int I,j;
for(I=0;I<5;I++)
 {
for(j=0;j<3;j++)
  {
    cin>>m[I][j];
  }
 }
for(I=0;I<5;I++)
 {
for(j=0;j<3;j++)
  {
    cout<<m[I][j];

  }
    cout<<endl;

 }

}
```

مثال / مصفوفة ذات بعدين 5*5 قراء جميع قيمها بالرقم 5

```
#include<iostream.h>
main()
{
int matrix [5] [5];
int m1,m2;
for (int m1=0 ; m1<5 ; m1++)
 {
      for (int m2=0 ; m2<5 ; m2++)
      {
       matrix [m1] [m2] = 5 ;
      }
 }
}
```

مثال / مصفوفة ذات بعدين 3*3 ادخال قيم اولية ثم طباعة القيم.

```
#include<iostream.h>
main()
{
int mat [3][3]= {{ 3,6,8 },{ 5,4,7 },{ 2,4,7 }};
int m1,m2;
for (int m1=0 ; m1<3 ; m1++)
 {
    for (int m2=0 ; m2<3 ; m2++)
    {
     cout << mat [m1] [m2];
    }
 }
}
```

مثال / مصفوفة ذات بعدين 5*5 ادخال قيم لها ثم البحث على الارقام الزوجية فقط ومعرفة عددها.

```
#include<iostream.h>
main()
{
int matrix [5] [5];
int m1,m2,even;
for (int m1=0 ; m1<5 ; m1++)
 {
      for (int m2=0 ; m2<5 ; m2++)
      {
        cin>> matrix [m1] [m2];
      }
 }

for (int m1=0 ; m1<5 ; m1++)
 {
      for (int m2=0 ; m2<5 ; m2++)
      {
         if( matrix [m1] [m2]%2 ==0)
           {
                even=even+1;
           }
      }
 }

}
```

مثال / مصفوفة ذات بعدين 5*5 ادخال قيم لها ثم جمع الاعداد الفردية فقط .

```
#include<iostream.h>
main()
{
int matrix [5] [5];
int m1,m2,sum;
for (int m1=0 ; m1<5 ; m1++)
 {
      for (int m2=0 ; m2<5 ; m2++)
      {
        cin>> matrix [m1] [m2];
      }
 }

for (int m1=0 ; m1<5 ; m1++)
 {
      for (int m2=0 ; m2<5 ; m2++)
      {
         if( matrix [m1] [m2]%2 ==1)
           {
                sum=sum+ matrix [m1] [m2];
           }
      }
 }
      cout << sum;
}
```

برمجة الكائنات

(Object-Oriented Programming)

الـــــدوال

الدوال المعرفة بواسطة المستخدم Programmer-defined Functions

الدوال تمكن المبرمج من تقسيم البرنامج إلى وحدات modules كل دالة في البرنامج تمثل وحدة قائمة بذاتها، ولذا نجد أن المتغيرات المعرفة في الدالة تكون متغيرات محلية (Local) ونعنى بذلك أن المتغيرات تكون معروفة فقط داخل الدالة.

أغلب الدوال تمتلك لائحة من الوسائط (Parameters) والتي هي أيضاً متغيرات محلية. هنالك عدة أسباب دعت إلى تقسيم البرنامج إلى دالات

١ /تساعد الدوال المخزنة في ذاكرة الحاسب على اختصار البرنامج إذ يكتفي باستدعائها باسمها فقط لتقوم بالعمل المطلوب.

٢ /تساعد البرامج المخزنة في ذاكرة الحاسب أو التي يكتبها المستخدم على تلافى عمليات التكرار في خطوات البرنامج التي تتطلب عملاً مشابها لعمل تلك الدوال.

٣ /تساعد الدوال الجاهزة في تسهيل عملية البرمجة.

٤ /يوفر استعمال الدوال من المساحات المستخدمة في الذاكرة.

كل البرامج التي رأيناها حتى الآن تحتوى على الدالة main وهي التي تنادى الدوال المكتبية لتنفيذ مهامها .سنرى الآن كيف يستطيع المبرمج بلغة ال سي ++ كتابة دوال خاصة به.

تعريف الدالة Function Definition

يأخذ تعريف الدوال الشكل العام التالي:

1- النوع الاول من كتابه الدوال .

وهذا النوع من الدوال لا يستقبل بيانات ولا يرجع اي قيمة .

```
function-name ( )
{
declarations and statements
}
```

حيث :

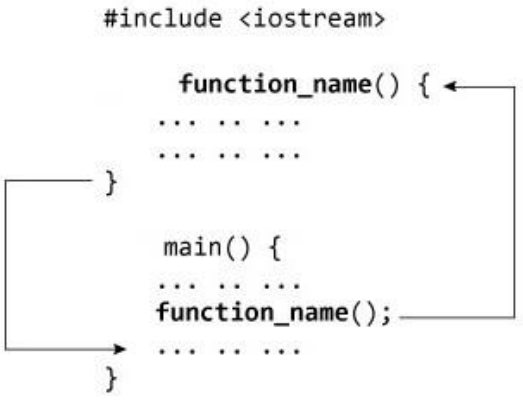
function-name:اسم الدالة والذي يتبع في تسميته قواعد تسمية المعرفات.

declarations and statements:

تمثل جسم الدالة والذي يطلق عليه في بعض الأحيان block يمكن أن يحتوى ال block على إعلانات المتغيرات ولكن تحت أي ظرف لا يمكن أن يتم تعريف دالة داخل جسم دالة أخرى .السطر الأول في تعريف الدالة يدعى المصرح declarator والذي يحدد اسم الدالة ونوع البيانات التي تعيدها الدالة وأسماء وأنواع وسيطاتها.

Ex.1

```
#include <iostream.h>
main ()
{
    print ();
}
Print ()
{
    cout << "Welcome to OOP ";
}
```



الشكل يوضح كيف يتم استدعاء الداله الثانوية داخل البرنامج الرئيسي

Ex.2

```
#include <iostream.h>
addition ()
{
  int a,b,r;
  a=10;
  b=20;
  r=a+b;
  cout<<r;
}
main ()
{
    addition ();
}
```

Ex.3

```
#include <iostream.h>
addition ()
{
  int a,b,r;
  cout <<"Enter a And b";
  cin >> a >> b;
  r=a+b;
  cout<<r;
}
```

```
main ()
{
    addition ();
}
```

2- النوع الثاني من كتابه الدوال .

وهذا النوع من الدوال يستقبل قيم ولا يرجع اي قيمة .

```
function-name (parameter list )
{
declarations and statements
                                                    }
```

حيث :

function-name:اسم الدالة والذي يتبع في تسميته قواعد تسمية المعرفات .

parameter list : هي لائحة الوسيطات الممرة إلى الدالة وهي يمكن أن تكون خالية (void) أو تحتوى على وسيطة واحدة أو عدة وسائط تفصل بينها فاصلة ويجب ذكر كل وسيطة على حدة.

declarations and statements: تمثل جسم الدالة والذي يطلق عليه في بعض الأحيان block يمكن أن يحتوى ال block على إعلانات المتغيرات ولكن تحت أي ظرف لا يمكن أن يتم تعريف دالة داخل جسم دالة أخرى . السطر الأول في تعريف الدالة يدعى المصرح declarator والذي يحدد اسم الدالة ونوع البيانات التي تعيدها الدالة وأسماء وأنواع وسيطاتها.

ex1.

```
#include <iostream>
addition (int a)
{
  int r;
  r=a*2;
cout << r;
```

```
}
 main ()
{
   addition (5);
}
```

ex2

```
#include <iostream>
addition (int a)
{
 int r;
 r=a+4;
 cout << r;
}
 main ()
{
 int x;
 cout << ""x";
 cin >> x;
 addition (x);
}
```

ex3

```
#include <iostream>
 main ()
{
 int x,y;
```

```
  cout << ""x,y";
  cin >> x >>y;
  addition (x,y);
  sub(x,y);
  Muilt(x,y);
  Div(x,y);

}
addition (int a,int b)
{
  int r;
  r=a+b;
cout <<"the sum = " << r;
}
sub (int a, int b)
{
  int r;
  r=a-b;
  cout <<"the sub = " << r;
}

Muilt (int a, int b)
{
  int r;
  r=a*b;
  cout <<"the Muilt = " << r;
```

}

```
Div (int a, int b)
{
  int r;
  r=a/b;
  cout <<"the Div = " << r;
}
```

```
# include <iostream>

    main() {
    ... .. ...
          add(num1, num2);  // Actual parameters: num1 and num2
    ... .. ...
}

    add(int a, int b) {     // Formal parameters: a and b
    ... .. ...
     add = a+b;
    ... .. ...
}
```

الشكل يوضح كيفية استدعاء القيم واستخدمها داخل الداله الثانوية

المتغيرات المرقمة والمصفوفات
Arrays and Matrices

مقدمة introduction

أن طرق التعامل مع أسماء المتغيرات والثوابت العددية والرمزية ، التي وردت في الفصول السابقة ، تعد صالحة للتعامل مع عدد محدود من هذه الثوابت والمتغيرات ، سواء في عمليات الإدخال والإخراج أو في العمليات الحسابية والمنطقية ، وعندما يصبح عدد المتغيرات كبيرا جدا ، تصبح تلك الطرق غير عملية ، فمثلا لو أردنا إدخال مائة قيمة للمتغيرات x2,x1- .... إلى x100 ، فكم الحيز المطلوب من البرنامج لعمليات الإدخال والإخراج والعمليات الحسابية والمنطقية لهذه المتغيرات ؟ هذا من جهة ، ومن جهة أخرى : فأننا نوفر مخزنا خاصا لكل متغير نتعامل معه ، أثناء تنفيذ البرنامج ، ولذلك لحفظ قيمته في مخون ، ومن ثم لاستعمال قيمته في عمليات أخرى تالية ، ومن ناحية ثالثة ، فان من الصعوبة بمكان ، بل من المستحيل استعمال اسم المتغير العددي أو الرمزي كمصفوفة ذات بعدين ، وثلاثة أبعاد ...الخ

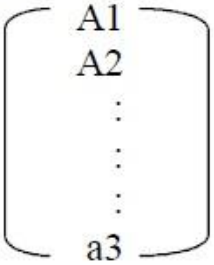
للأسباب الثلاثة الواردة أعلاه ، جاءت فكرة استعمال متغير جماعي يضم تحت اسمه عددا من العناصر يسمى بالمتغير الرقمي subscripted variable ، ويتم ترقميه بين قوسين مربعين [ ] يوضع بينهما قيمة العداد المرقم subscript ، وقد تسمية الدليل index أحيانا ، ويمكننا تشبيه المتغير المرقم بقسم الهاتف لمؤسسة ما ، فهو مقسم واحد ، تنظم تحته عدد من الأرقام الفرعية للموظفين وكل رقم من هذه الأرقام مستقل ومتميز عن الأرقام الفرعية الأخرى ، وله مخزن خاص في الذاكرة ، الآن انه كغيره من الأرقام الفرعية تابع للرقم العام لمقسم المؤسسة ، كما يمكن تشبيه المتغير المرقم بالجيش الذي يعامل كاسم متغير واحد ، لكن يضم عددا كبيرا من العناصر ، فمثلا العناصر التالية :(من اليمين إلى اليسار):

A[n] ...a[2], a[1], a[0]

تابع للمتغير الجماعي a[ ]
وكل عنصر من هذه العناصر له عنوان في الذاكر address ، فالعنوان الأول يكون للعنصر الأول والثاني للثاني والثالث للثالث ... وهكذا.
ويستعمل المتغير الجماعي [المرقم] أو المصفوفة ، في لغة c++ وغيرها ، حجز جماعي مسبق في الذاكرة لجميع عناصره ، فلو كان يتبعه خمسون عنصرا ، فانه يحجز له 50 مخزنا ، على الأقل في الذاكرة .

من الفوائد المهمة للمتغيرات المرقمة والمصفوفات : هو استعمالها في الترتيب التصاعدي والتنازلي للعناصر والقيم المختلفة ، وعمليات ترتيب الأسماء الأبجدي

النصوص الرمزية ، وفي عمليات ضرب المصفوفات ، وإيجاد معكوس المصفوفة وعملياتها الأخرى ، وفي التحليل العددي ... الخ.
المتغير المرقم (المصفوفة) ذو البعد الواحد one-dimensional Array
المتغير المرقم ذو البعد الواحد هو مصفوفة ذات بعد واحد أو متجه (vector) ويمثل في الجبر على النحو الأفقي [a1 a2 ...a3]
أو العمودي



ويأخذ المرقم المتغير في c++ الشكل العام التالي:

```
Type-specifier array-name[size];
```

ويبدأ العداد المرقم عادة من الصفر ، آي أن العنصر الأول من المصفوفة []a هو a[0] والثاني a[1] ... وهكذا فمثلا المصفوفة التالية:

Int a[20];

اسمها a ، وقد حجز لها 20 موقعا لعشرين عنصرا من النوع الصحيح .

والمصفوفة التالية:

Char name[15];

مصفوفة رمزية ، اسمها name يحجز لها خمسة عشر عنصرا من النوع الرمزي لها .

وهكذا...

عنوان عناصر المصفوفة في الذاكرة Addressing Array Elements in Memory

ذكرنا من قبل أن آي متغير أو عنصر من متغير ذاتي مرقم ، يحتل موقعا من الذاكرة يستعمل عادة مؤشرا لكل متغير أو عنصر ، ليكون دليلا على استعمال هذه المتغيرات والعناصر بسهولة ويسر ، والمثال التالي يوضح هذه العملية بالنسبة للمصفوفة ذات بعد واحد.

Int x[5];

يمكن تمثيل عناصر المصفوفة x المعلن عنها ، مع عناوينها بالشكل التوضيحي التالي (من اليسار إلى اليمين)

الأول	الثاني	الثالث	الرابع	الخامس
X[0]	X[1]	X[2]	X[3]	X[4]
100	101	102	103	104

امثلة حول استخدام المصفوفه ذات البعد الواحد ..

مثال1/ عملية إدخال ذاتي لقيم عناصر متغير مرقم مصفوفة ذي بعد واحد

```
#include < iostream.h>
main ()
{
int a[20];
int I;
for (I=0;I<20;++I)
 {
    a[I]=I+1;
 }
}
```

في هذه الحالة يتم إدخال عشرين عنصرا من عناصر المصفوفة a
A[0]=1 عندما يكون I=0
A[1]=2 عندما يكون I=1
...
...
...
a[19]=20 عندما يكون I=19

مثال2 / مصفوقتين ضرب مصفوقة في مصفوقة اخرى .

```
#include < iostream.h>
main ()
{
int x[5], y[5];
int I;
for (I=0;I<5;++I)
{
x[I]=I;
y[I]=I*I;
cout<<endl<<x[I]<<y[I];
}
}
```

وستكون قيم النتائج على النحو التالي:

0	0
1	1
2	4
3	9
4	16

إعطـاء قيمة أولية للمصفوفة ذات البعد الواحد Array Initialization

مثال على إدخال عدة عناصر من مصفوفة الدرجات grade[]

```
Int grade[5]={80,90,54,50,95}
```

ومثال على إدخال قيم عناصر المصفوفة الرمزية name[]

```
Char name[4]="nor"
```

لاحظ أن المتغير المرقم name[] مكون من أربعة عناصر بينما تم إعطاؤه ثلاثة عناصر فقط والسبب أن العنصر الرابع بالنسبة إلى المعطيات الرمزية يكون خاليا.

مثال3 / ادخال قيم الى المصفوفة مباشرتنا بدون استخدام دالة القراء.

```
#include < iostream.h>
main ()
{
int a[6]={40,60,50,70,80,90}
int I;
for(I=0;I<6;I++)
{
cout<<a[I]<<endl;
}
}
```

والناتج طبعا سيكون كالتالي :

40
60
50
70
80
90

مثال٤:
قم بكتابة برنامج يقوم بإيجاد مجموع ، ومعدل علامات الطالب في 5 مواد وهذه العلامات كالتالي:
87,67,81,90,55

```
#include < iostream.h>
main ()
{
int i;
int a[5]={87,67,81,90,55}
int s=0;
float avg;
for(i=0;i<5;i++)
{
s=s+a[i];
}
 avg=s/5;
cout<<avg<<endl;<<s<<endl;
}
```

والناتج سيكون كالتالي:

87
735

المعدل 87
والمجموع 735

مثال5/ ادخال بيانات المصفوفه عن طريقة داله الادخال.

```
#include < iostream.h>
main ()
{
int i;
int a[5];

for(i =0; i<=4;i++)
  {
    cout<<"enter Numbers Here:"<<endl;
     cin>>a[i];
  }

}
```

مثال6/ ادخال بيانات المصفوفه عن طريقة داله الادخال ثم طباعه الارقام المدخله.

```
 #include < iostream.h>
main ()
{
int i;
int a[5];

for(i =0; i<=4;i++)
  {
    cout<<"enter Numbers Here:"<<endl;
     cin>>a[i];
  }

for(i=0;i<=4;i++)
{
    cout<<a[i]<<endl;
}

}
```

مثال7/ ادخال بيانات المصفوفه عن طريقة داله الادخال وجمع القيم المدخله واخرج المعدل .

```
#include < iostream.h>
main ()
{
int i;
int a[5];
int s=0;
float avg=0.0;
for(i =0; i<=4;i++)
  {
    cout<<"enter Numbers Here:"<<endl;
     cin>>a[i];

  }

for(i=0;i<=4;i++)
{
  s=s+a[i];
}
   avg=s/5;
cout<<"Sum ="<<s;
cout<<"Avg = "<<avg;
}
```

مثال 8/ادخال سبع قيم اولية للمصفوفه ومعرفه عدد القيم التي اكبر من 50 واصغر من 50 ؟

```
#include < iostream.h>
main ()
{
int i,S,L;
int a[5]={87,67,41,90,55,53,38}
for(i =0; i<=6;i++)
  {
     if(a[i]>=50)
      {
            L=L+1;
      }
     else
```

```
    {
       S=S+1;
     }
  }
   cout<<" 50 اكبر من "<<L<<Endl;
   cout<<" 50 اصغر من "<<S<<Endl;

}
```

مثال 9/ ادخال عشرة قيم اولية للمصفوفه ومعرفه عدد القيم السالبه وعدد القيم الموجبة ؟

```
#include < iostream.h>
main ()
{
int i,N,P;
int a[5]={87,-167,141,90,55,53,38,-4,-2, 1}
for(i =0; i<=9;i++)
  {
    if(a[i]>=0)
     {
          P=P+1;
     }
    else
     {
        N=N+1;
      }
  }
   cout<<"  الاعداد الموجبة"<<P<<Endl;
   cout<<" الاعداد السالبة"<<N<<Endl;

}
```

H/W:

1- ادخال عشرة قيم اولية للمصفوفه ومعرفه اكبر قيمه بينهما ؟
2- ادخال عشرة قيم اولية للمصفوفه ومعرفه اصغر قيمه بينهما ؟
3- ادخال عشرة قيم اولية للمصفوفه وتبدل قيمه العنوان الاول A[0] بالعنوان الاخير A[9] ثم طباعه المصفوفه بعد التدبيل ؟

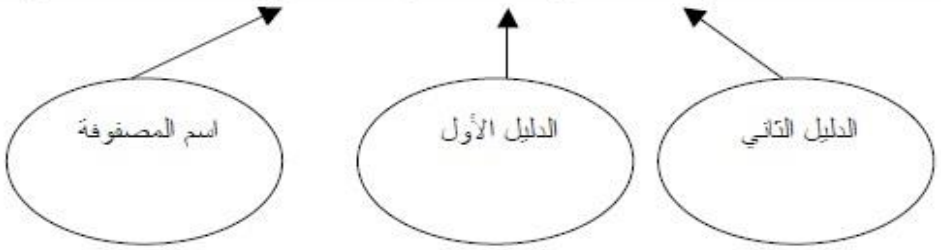
4- ادخال خمسه قيم اولية لمصفوفه a[5] ثم نقوم بازاحه قيم المصفوفه نحو الامام بحيث قيمه العنوان الاول تكون بدل قيمه العنوان الثاني ، والثاني بدل الثالث ، والثالث بدل الرابع وهكذا..... ، والاخير بدل الاول ؟

5- ادخال عشرة قيم اولية للمصفوفه وجمع قيم العناونين الزوجية للمصفوفه؟

Two-Dimensional Arrays المصفوفة ذات البعدين

تشبة المصفوفة ذات البعدين في طريقة تعاملها ، المصفوفة ذات البعد الواحد إلا أن لها عداديـن(index2) دليليـن أو مرقميـن إحداهمـا عـداد للصـفوف ، والأخـر عـداد للأعمدة ويأخذ الإعلان عن المصفوفة الشكل العام التالي:

```
Type-specifier arraey_name [index 1][index 2];
```



فمثلا المصفوفة :

```
Int x[2][3];
```

وهي مصفوفة صحيحة العناصر int أبعادها هي عدد الصفوف=2 ، وعدد الأعمدة =3

لاحظ أن عدد الصفوف يوضع بين قوسين وحده ، وكذلك عداد الأعمدة .

مثال / شاهد هذا المثال الذي يستخدم 5 صفوف و 3 اعمدة:

```
#include <iostream.h>
main ()
{
int m[5][3];
int I,j;
for(I=0;I<5;I++)
 {
for(j=0;j<3;j++)
  {
    cin>>m[I][j];
  }
 }
}
```

مثال / شاهد هذا المثال الذي يستخدم 5 صفوف و 3 اعمدة قراء القيم ثم طباعة القيم المدخلة:

```
#include <iostream.h>
main ()
{
int m[5][3];
int I,j;
for(I=0;I<5;I++)
 {
for(j=0;j<3;j++)
  {
    cin>>m[I][j];
  }
 }
for(I=0;I<5;I++)
 {
for(j=0;j<3;j++)
  {
    cout<<m[I][j];

  }
    cout<<endl;

 }

}
```

مثال / مصفوفة ذات بعدين 5*5 قراء جميع قيمها بالرقم 5

```
#include<iostream.h>
main()
{
int matrix [5] [5];
int m1,m2;
for (int m1=0 ; m1<5 ; m1++)
 {
      for (int m2=0 ; m2<5 ; m2++)
      {
       matrix [m1] [m2] = 5 ;
      }
 }
}
```

مثال / مصفوفة ذات بعدين 3*3 ادخال قيم اولية ثم طباعة القيم.

```
#include<iostream.h>
main()
{
int mat [3][3]= {{ 3,6,8 },{ 5,4,7 },{ 2,4,7 }};
int m1,m2;
for (int m1=0 ; m1<3 ; m1++)
 {
    for (int m2=0 ; m2<3 ; m2++)
    {
     cout << mat [m1] [m2];
    }
 }
}
```

مثال / مصفوفة ذات بعدين 5*5 ادخال قيم لها ثم البحث على الارقام الزوجية فقط ومعرفة عددها.

```
#include<iostream.h>
main()
{
int matrix [5] [5];
int m1,m2,even;
for (int m1=0 ; m1<5 ; m1++)
 {
      for (int m2=0 ; m2<5 ; m2++)
      {
        cin>> matrix [m1] [m2];
      }
 }

for (int m1=0 ; m1<5 ; m1++)
 {
      for (int m2=0 ; m2<5 ; m2++)
      {
         if( matrix [m1] [m2]%2 ==0)
           {
                even=even+1;
           }
      }
 }

}
```

مثال / مصفوفة ذات بعدين 5*5 ادخال قيم لها ثم جمع الاعداد الفردية فقط .

```
#include<iostream.h>
main()
{
int matrix [5] [5];
int m1,m2,sum;
for (int m1=0 ; m1<5 ; m1++)
 {
      for (int m2=0 ; m2<5 ; m2++)
      {
        cin>> matrix [m1] [m2];
      }
 }

for (int m1=0 ; m1<5 ; m1++)
 {
      for (int m2=0 ; m2<5 ; m2++)
      {
         if( matrix [m1] [m2]%2 ==1)
           {
                sum=sum+ matrix [m1] [m2];
           }
      }
 }
      cout << sum;
}
```

برمجة الكائنات

(Object-Oriented Programming)

الـــــدوال

الدوال المعرفة بواسطة المستخدم Programmer-defined Functions

الدوال تمكن المبرمج من تقسيم البرنامج إلى وحدات modules كل دالة في البرنامج تمثل وحدة قائمة بذاتها، ولذا نجد أن المتغيرات المعرفة في الدالة تكون متغيرات محلية (Local) ونعنى بذلك أن المتغيرات تكون معروفة فقط داخل الدالة.

أغلب الدوال تمتلك لائحة من الوسائط (Parameters) والتي هي أيضاً متغيرات محلية. هنالك عدة أسباب دعت إلى تقسيم البرنامج إلى دالات

١ /تساعد الدوال المخزنة في ذاكرة الحاسب على اختصار البرنامج إذ يكتفي باستدعائها باسمها فقط لتقوم بالعمل المطلوب.

٢ /تساعد البرامج المخزنة في ذاكرة الحاسب أو التي يكتبها المستخدم على تلافى عمليات التكرار في خطوات البرنامج التي تتطلب عملاً مشابها لعمل تلك الدوال.

٣ /تساعد الدوال الجاهزة في تسهيل عملية البرمجة.

٤ /يوفر استعمال الدوال من المساحات المستخدمة في الذاكرة.

كل البرامج التي رأيناها حتى الآن تحتوى على الدالة main وهي التي تنادى الدوال المكتبية لتنفيذ مهامها .سنرى الآن كيف يستطيع المبرمج بلغة ال سي ++ كتابة دوال خاصة به.

تعريف الدالة Function Definition

يأخذ تعريف الدوال الشكل العام التالي:

1- النوع الاول من كتابه الدوال .

وهذا النوع من الدوال لا يستقبل بيانات ولا يرجع اي قيمة .

```
function-name ( )
{
declarations and statements
}
```

حيث :

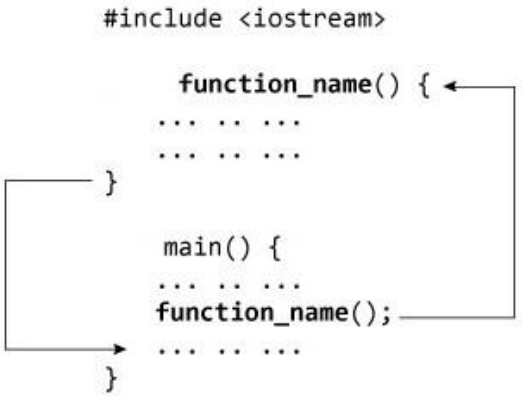
function-name:اسم الدالة والذي يتبع في تسميته قواعد تسمية المعرفات.

declarations and statements:

تمثل جسم الدالة والذي يطلق عليه في بعض الأحيان block يمكن أن يحتوى ال block على إعلانات المتغيرات ولكن تحت أي ظرف لا يمكن أن يتم تعريف دالة داخل جسم دالة أخرى .السطر الأول في تعريف الدالة يدعى المصرح declarator والذي يحدد اسم الدالة ونوع البيانات التي تعيدها الدالة وأسماء وأنواع وسيطاتها.

Ex.1

```
#include <iostream.h>
main ()
{
    print ();
}
Print ()
{
    cout << "Welcome to OOP ";
}
```



الشكل يوضح كيف يتم استدعاء الداله الثانوية داخل البرنامج الرئيسي

Ex.2

```
#include <iostream.h>
addition ()
{
  int a,b,r;
  a=10;
  b=20;
  r=a+b;
  cout<<r;
}
main ()
{
    addition ();
}
```

Ex.3

```
#include <iostream.h>
addition ()
{
  int a,b,r;
  cout <<"Enter a And b";
  cin >> a >> b;
  r=a+b;
  cout<<r;
}
```

```
main ()
{
    addition ();
}
```

2- **النوع الثاني من كتابه الدوال .**

وهذا النوع من الدوال يستقبل قيم ولا يرجع اي قيمة .

```
function-name (parameter list )
{
declarations and statements
}
```

حيث :

function-name:اسم الدالة والذي يتبع في تسميته قواعد تسمية المعرفات .

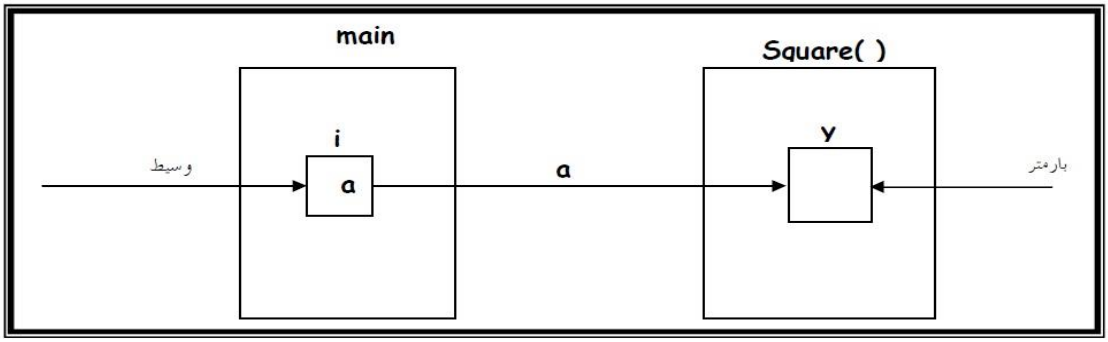
parameter list : هي لائحة الوسيطات الممرة إلى الدالة وهي يمكن أن تكون خالية (void) أو تحتوى على وسيطة واحدة أو عدة وسائط تفصل بينها فاصلة ويجب ذكر كل وسيطة على حدة.

declarations and statements: تمثل جسم الدالة والذي يطلق عليه في بعض الأحيان block يمكن أن يحتوى ال block على إعلانات المتغيرات ولكن تحت أي ظرف لا يمكن أن يتم تعريف دالة داخل جسم دالة أخرى . السطر الأول في تعريف الدالة يدعى المصرح declarator والذي يحدد اسم الدالة ونوع البيانات التي تعيدها الدالة وأسماء وأنواع وسيطاتها.

الوسيطات Parameters

الوسيطات هي الآلية المستخدمة لتمرير المعلومات من استدعاء الدالة إلي الدالة نفسها حيث يتم نسخ البيانات وتخزن القيم في متغيرات منفصلة في الدالة تتم تسمية هـــذه المتغيرات في تعريف الدالة.فمثلاً في المثال السابق تؤدى العبارة cout<< square(a); في ( )main إلى نسخ القيمة a إلى البارمتر y المعرف في تعريف الدالة.

المصطلح وسيطات Argument يعنى القيم المحددة في استدعاء الدالة بينما يعنى المصطلح بارمترات parameters المتغيرات في تعريف الدالة والتي تم نسخ تلـــك القيم إليها، ولكن غالباً ما يتم استعمال المصطلح وسيطات لقصد المعنيين.



البرنامج التالي يستخدم دالة تدعى maximum والتي نرجع العدد الأكبر بين ثلاثة أعداد صحيحة. يتم تمرير الأعداد كوسائط للدالة التي تحدد الأكبر بينها وترجعه للدالة main باستخدام العبارة return ويتم تعيين القيمة التي تمت إعادتهاا إلى المتغير largest الذي تتم طباعته.

```
#include <iostream.h>
int maximum (int, int, int);
main( )
{
int a, b, c;
cout << "Enter three integers: " ;
```

```
cin >> a >> b >> c ;
cout << " maximum is : " << maximum (a, b, c) << endl;
return 0;
}
int maximum (int x, int y, int z)
{
int max = x;
if (y > x)
max = y;
if (z > max)
max = z;
//Continued
return max;
}
```

ex1.

```
#include <iostream>
addition (int a)
{
  int r;
  r=a*2;
cout << r;
}
  main ()
{
    addition (5);
```

```
}
```

ex2

```
#include <iostream>
addition (int a)
{
  int r;
  r=a+4;
  cout << r;
}
  main ()
{
  int x;
  cout << ""x";
  cin >> x;
  addition (x);
}
```

ex3

```
#include <iostream>
  main ()
{
  int x,y;
  cout << ""x,y";
  cin >> x >>y;
  addition (x,y);
  sub(x,y);
```

```
  Muilt(x,y);
  Div(x,y);

}
addition (int a,int b)
{
  int r;
  r=a+b;
cout <<"the sum = " << r;
}
sub (int a, int b)
{
  int r;
  r=a-b;
  cout <<"the sub = " << r;
}

Muilt (int a, int b)
{
  int r;
  r=a*b;
  cout <<"the Muilt = " << r;
}

Div (int a, int b)
{
```

```
  int r;
  r=a/b;
  cout <<"the Div = " << r;
}
```

```
# include <iostream>

    main() {
    ... .. ...
          add(num1, num2);  // Actual parameters: num1 and num2
    ... .. ...
}

    add(int a, int b) {     // Formal parameters: a and b
    ... .. ...
     add = a+b;
    ... .. ...
}
```

الشكل يوضح كيفية استدعاء القيم واستخدمها داخل الداله الثانوية

3- النوع الثالث من كتابه الدوال .

وهذا النوع من الدوال يستقبل قيم و يرجع قيمة .

```
return-value-type  function-name  (parameter list )
{
declarations and statements
}
```

حيث:

parameter list : هي لائحة الوسيطات الممرة إلى الدالة وهي يمكن أن تكون خالية (void) أو تحتوى على وسيطة واحدة أو عدة وسائط تفصل بينها فاصلة ويجب ذكر كل وسيطة على حدة.

return-value-type: نوع القيمة المعادة بواسطة الدالة والذي يمكن أن يكون أي نوع من أنواع بيانات وإذا كانت الدالة لا ترجع أي قيمة يكون نوع اعادتها void

function-name:اسم الدالة والذي يتبع في تسميته قواعد تسمية المعرفات .

declarations and statements: تمثل جسم الدالة والذي يطلق عليه في بعض الأحيان block يمكن أن يحتوى ال block على إعلانات المتغيرات ولكن تحت أي ظرف لا يمكن أن يتم تعريف دالة داخل جسم دالة أخرى .

السطر الأول في تعريف الدالة يدعى المصرح declarator والذي يحدد اسم الدالة ونوع البيانات التي تعيدها الدالة وأسماء وأنواع وسيطاتها.

قيم الإعادة Returned Values

بإمكان الدالة أن تعيد قيم إلي العبارة التي استدعتها .ويجب أن يسبق اسم الدالة في معرفها وإذا كانت الدالة لا تعيد شيئاً يجب استعمال الكلمة الأساسية void كنوع إعادة لها للإشارة إلي ذلك. هنالك ثلاث طرق إرجاع التي يمكن بها التحكم إلى النقطة التي تم فيها استدعاء الدالة:

1. إذا كانت الدالة لا ترجع قيمة يرجع التحكم تلقائياً عند الوصول إلى نهاية الدالة.
2. باستخدام العبارة return;

إذا كانت الدالة ترجع قيمة فالعبارة return expression; تقوم بإرجاع قيمة التعبير expression إلى النقطة التي استدعتها

البرنامج التالي يستخدم دالة تدعى maximum والتي نرجع العدد الأكبر بين ثلاثة أعداد صحيحة. يتم تمرير الأعداد كوسائط للدالة التي تحدد الأكبر بينها وترجعه للدالة main باستخدام العبارة return ويتم تعيين القيمة التي تمت إعادتهاا إلى المتغير largest الذي تتم طباعته.

Ex.

ارسال ثلاث قيم الى داله واسترجاع القيمه الاكبر الى الداله الرئيسية ..

```
#include <iostream.h>
int maximum (int x, int y, int z)
{
int max = x;
if (y > max)
max = y;
if (z > max)
max = z;
return max;
}

main( )
{
int a, b, c;
cout << "Enter three integers: " ;
cin >> a >> b >> c ;
cout << " maximum is : " << maximum (a, b, c) << endl;
}
```

Ex.

داله تستقبل5 قيم لطالب ما ثم ترجع قيمه المعدل النهائي للطالب ..

```
#include <iostream.h>
int Avg (int a, int b, int c,int d, int e )
{
float av;
av=a+b+c+d+e/5;
return av ;
}

main( )
{
int a, b, c,d,e;
cout << "Enter 5 integers: " ;
cin >> a >> b >> c >>d>e;
cout << " Avg is : " << Avg (a, b, c,d,e) << endl;
}
```

Ex

اكتب برنامج بلغة C++ مضروب n حيث $n! = n(n-1)(n-2) \times \dots 1$

باستخدام الداله ..

```
#include <iostream.h>
int FactNumber(int n )
{
int i,fact;
     fact=1;

     for (i=1;i<=n;i++)
          fact=fact*i;
     cout<<"fact = "<<fact<<endl;
```

```
}
main()
{
     int n;
     cin>>n;
  cout<< "The Fact Number is "<<  FactNumber(n);

}
```

النتائج o

```
n= 5
fact = 120
Press any key to continue
```

Ex.

اكتب برنامج يحسب حاصل جمع وضرب عددين :

```
#include <iostream.h>
void set_data(int a,int b)
     {
          cout<<"the sum of number = "<<a+b<<endl;
          cout<<"the multi of number = "<<a*b<<endl;
     }
     main()
    {
     int n1,n2;
     cin>>n1>>n2;
     num1.set_data (n1,n2);
     }
```

البرمجة الإجرائية:

وهذه أول تقنية ظهرت والتي كان من المفترض لها أن تظهر ؛ تركز البرمجة الإجرائية على إجراء البرنامج في خطوات واضحة ومحددة لا تحيد عنها وهي عبارة عن كتلة واحدة ..وبالرغم من أنها قدمت للمبرمجين الكثير إلا أن الأكواد تصبح أكثر تعقيداً حينما يتعامل الشخص مع مشاريع كبيرة الحجم .أيضاً لم يكن بإمكان الأشخاص العمل كفريق عمل لأنه لا يوجد تقسيم واضح في الكود فالكود عبارة عن كتلة واحدة ..أيضاً عند القيام بصيانة البرنامج فإن الأمر يصبح أكثر تعقيداً وخاصة عند تتبع سير البرنامج لمعرفة أين يوجد الخطأ.

البرمجة الهيكلية:

أتت البرمجة الهيكلية لحل المشاكل التي تعاني منها البرمجة الإجرائية إلا أنها لم تقدم الكثير؛ ولا أعتقد أنها قفزة نوعية في مجال البرمجة ، فهي لا تقوم بأي شيء سوى بتقسيم الكود إلى عدة أكواد أو إجراءات بالمعنى الأصح ....أيضاً لا يمكنك في بعض الحالات أن تعيد إستخدام هذه الإجراءات في برامج أخرى وفي بعض اللغات التي لا تملك ميزة المرجعيات والمؤشرات لا يمكن للإجراء ألا يعيد سوى قيمة واحدة ..أيضاً لا يمكنك إستخدام المتغيرات العامة بكثرة فهي تعقد البرنامج أكثر وتجعل من عملية تتبع سير البرنامج عملية مستحيلة ..وبسبب أن بعض الإجراءات تعتمد على وجود هذه المتغيرات العامة في البرنامج فلن يمكنك إعادة إستخدام هذا الإجراء في برامج أخرى لأن هذا الإجراء ليس مستقلاً كما يخيل للبعض ...من أجل كل هذه العيوب والنواقصفي البرمجة الهيكلية والإجرائية ظهرت البرمجة الشيئية والتي لم ترتكز إلا على تغيير مفهوم البرمجة.

البرمجة الشيئية:

ترتكز البرمجة الشيئية في وجودها ليسعلى إجراءات وإنما على وجود الأصناف والكائنات ؛ فالأصناف هي الوحدة الأساسية لأي برنامج يكتب بالبرمجة الشيئية ؛ تتألف الأصناف من متغيرات ودوال .وبمعنى برمجي شيئي بحت دون التدخل في لغات البرمجة فإن الصنف يتكون من شيئين اثنين هما **Attributes** و **Behaviors**

تعريف الصنف :هو عبارة عن قالب يعرّف مجموعة من الخواص والسلوك كما هي موجودة في العالم الحقيقي.

مثال برنامج تسجيل الطلاب في الجامعة:

فمثلاً لو أردنا القيام بعمل برنامج لتسجيل الطلاب في الجامعة فمن الممكن أن نقسم البرنامج إلى عدة أصناف وهي صنف الطالب ؛ صنف الكلية أو القسم ؛ صنف مسجل الطلاب صنف عمادة القبول والتسجيل ؛ وسنأخذ مثال صنف الطالب أولاً ، يتألف صنف الطالب من متغيرات ودوال، من أمثلة المتغيرات لدى الطالب درجة الطالب ، تقدير الطالب، عمر الطالب ومن أمثلة الدوال لدى الطالب ، دالة إختيار الكلية المرغوب بها أما بالنسبة لصنف الكلية أو القسم فمن أهم المتغيرات لديه هي اسم الكلية واسم التخصص والدرجة التي يقبل على أساسها الطالب أما بالنسبة للدوال فمن أهمها دالة القبول المبدئي والتي تتأكد من توافق شروط القبول مع الطالب (ودالة القبول النهائي) وهي الدالة التي تفاضل بين الطلاب حسب معايير الكلية وبالتالي تقبل الطالب؛ وبإمكاننا هنا وضع متغير جديد ألا وهو مصفوفة الطلاب المقبولين قبولاً مبدئياً ومتغير آخر هو مصفوفة الطلاب المقبولين قبولاً نهائياً ولن نتعرضهنا على دوال أخرى مثل دالة الفصل النهائي لأننا نكتب هنا برنامج لتسجيل الطلاب بالنسبة للصنف الأخير وهو صنف مسجل الطلاب أو بالمعنى الأصح عمادة القبول والتسجيل ؛ صنف عمادة القبول والتسجيل يتألف من هذه المتغيرات :مصفوفة الطلاب الراغبين بدخول الجامعة وتحوي أيضاً قائمة بالتخصصات المرغوبة ؛ ومصفوفة أيضاً تحوي أسماء الكليات وأقسامها ومن الدوال دالة تقوم بتسجيل الطلاب في قوائم القبول المبدئيتتأكد من توافق الشروط العامة للجامعة مع الطلاب ودالة أخرى تقوم بإرسال اسم الطالب ودرجة الطالب إلى الكلية ودالة ثالثة تستقبل أسماء الطلاب المقبولين قبولاً نهائياً في الكليات وبالتالي فبإمكاننا تمثيل هذه الأصناف في الأشكال التالية:

كما تلاحظ فلقد أتممنا تصميم برنامج تسجيل الطلاب في دقائق قليلة ولم نحتاج فقط إلا للقليل من التركيز وقمنا بتمثيل واقعي للكائنات في البرنامج ؛ تخيل الآن مالذي سيحدث لو أننا قمنا بتصميم برنامج هيكلي تخيل مدى التعقيد الواقع في البرنامج وكيفية تتبع سير البرنامج وماذا علينا أن نضعه متغيرات عامة ومالذي لا نضعه متغيرات عامة وماهي المتغيرات التي نرسلها لكل دالة وماهي القيم المعادة .. إلخ هذه هي فكرة الكائنات وهي فكرة تبسط من المسائل المعقدة لجعلها تبدو بسيطة وتجعل صيانة البرنامج أكثر تقدماً وسرعة.

إنشاء كائن Creating a Class Instance

الآن سنأتي إلى بعض النقاط المهمة ..كما تلاحظ فلقد كتبنا الصنف الطالب لكن لم نحدد في هذا التصنيف ما هو اسم الطالب لنفرض أن عدد الطلاب الذين اتوا للتقديم هم خمسمائة طالب فمالذي علينا فعله كل الذي عليه أن نعرفه قبل أن نعمل أي شيء أن نفهم الفرق بين الصنف والكائن .. الصنف مثل المخطط بينما الكائن هو تطبيق هذا المخطط . أي أنك لو قمت بكتابة صنف الطالب في برنامج ما ؛ فلن يحجز له المترجم أي ذاكرة بالرغم من وجود المتغيرات لأنك في الأساس تخبر المترجم أن هناك صنف جديد فقط .لا تخبره بأن يحجز لك مكان في الذاكرة بالتالي فكل ما عليك فعله .أن تقوم بإنشاء كائن **Object** الآن انظر لهذا السطر الكودي الخارج عن الموضوع وحاول أن تفهم ما أحاول أن أقول:

```
int x=5;
```

كما ترى فأنت تحجز للمتغير **x** ذاكرة من نمط **int** العلاقة بين المتغير ونمط البيانات هي نفس العلاقة بين الكائن والصنف ؛

التغليف Encapsulation :

قبل أن نذكر فائدة التغليف فعلينا أن نحاول إيصال مفهومها إلى القارئ ؛تعرف ا التغليف على أنها إخفاء المعلومات عن المستخدم أقصد هنا (مستخدم الصنف) دعك عن لماذا الآن؟ من الممكن أن نشبه الصنف على أنه صندوق أسود هذا الصندوق له معلومات لاستخدامه فإذا أخذنا مثال الصراف الآلي فأنت تقوم بإدخال بطاقتك البنكية ورقمها السري لتجري بعض العمليات والتي لا يهمك أن تعرفها وتخرج لك ما تريد من الصراف ؛ بهذه الطريقة يمكن تشبيه التغليف ؛ لا يهمك أنت أن تعرف ماذا يحدث في الصراف وهذا أحد الأسباب وهناك سبب آخر وهو أن البنك لا يريدك أن تعبث بالصراف فإذا كان بإمكانك تغيير برنامج الصراف وبالتالي تغيير برنامج البنك على ما تشتهيه نفسك فقد تحصل كارثة اقتصادية في البلاد .. وهذا أيضاً على صعيد البرمجة الكائنية فمن جهة لا يهمك ما يحدث داخل الصنف ومن جهة أخرى فإنه لا ينبغي لك أن تعبث بالمحتويات الداخلية للصنف ... وهذه هي فائدة التغليف..وعلى الصعيد الكودي فهناك كلمتان public والتي تعني أن الأعضاء الذين تحتها هم أعضاء عامة بالإمكان التغيير فيهم والكلمة الأخرى هي private وتعني أن الأعضاء الذين تحتها هم أعضاء غير مرئيين خارج الطبقة أي أعضاء مغلفين.

الفئات Classes :

أساس البرامج المكتوبة باللغة C++ هو الكائنات التي يتم إنشاؤها بواسطة فئة تستعمل كقالب فعندما يكون هنالك الكثير من الكائنات المتطابقة في البرنامج لا يكون منطقياً وصف كل واحد منها على حدة ، من الأفضل تطوير مواصفات واحدة لكل من هذه الكائنات وبعد تحديد تلك المواصفات يمكن استخدامها لإنشاء قدر ما نحتاج إليه من الكائنات تسمى مواصفات إنشاء الكائنات هذه في *OOP* فئة **(Class)** تتميز الفئة في C++ .

/اسم الفئة والذي يعمل كنوع البيانات الذي ستمثله الفئة.

/مجموعة من الأعضاء البيانية في الفئة **(data members)** حيث يمكن أن تحتوى الفئة على صفر أو أكثر من أي نوع من أنواع البيانات في *C++*

/مجموعة من الأعضاء الدالية **(member functions)** معرفة داخل الفئة وهي تمثل مجموعة العمليات التي سيتم تنفيذها على كائنات الفئة.

/محددات وصول **(access specifiers)** وتكتب قبل الأعضاء البيانية والأعضاء الدالية لتحدد إمكانية الوصول إلى هذه الأجزاء من الأجزاء الأخرى في البرنامج.

تعريف الفئة The Class Definition

يتألف تعريف الفئة من الكلمة الأساسية **class** يليها اسم الفئة ثم جسم الفئة بين قوسين حاصرين { } ويجب أن ينهي تعريف الفئة فاصلة منقوطة أو عبارة إعلان عن كائنات تنتمي إلى الفئة فمثلاً:

```
class anyclass { /* class body*/ };
```

غالباً ما تكتب الفئة في C++ على النحو التالي في البرنامج .

```
class class_name{
private:
data members
public:
member functions
};
```

المثال التالي يوضح كيفية تعريف فئة

```
// This creates the class stack >
class stack {
private:
int stck[SIZE];
int tos;
public:
void init ( ); // Void تعني لا ترجع قيمة
void push(int i); //تعني لا ترجع قيمة
int pop ( );
};
```

مثال 1- كيفية كتابة برنامج باستخدام ال Class ؟

```
#include<iostream>
class programming
{
   private:
      int var;

   public:

      void input_value()
      {
cout << "In function input_value, Enter an integer\n";
cin >> var;
      }


      void output_value()
      {
         cout << "Variable entered is ";
         cout << var << "\n";
      }
};

main()
{
   programming object; // استدعاء ال class

   object.input_value();
   object.output_value();
}
```

مثال 2- كيفية كتابة برنامج باستخدام ال Class قراءة وطباعة بيانات ؟

```
#include <iostream.h>
class Persnal
{
private :
```

```
public :
void show_Data(string name,String Email,string
Number,int Age, int stage)
{
cout<<"the Name is  = "<<name<<endl;
cout<<"the Email is  = "<<Email<<endl;
cout<<"the Number is  = "<<Number<<endl;
cout<<"the Number Phone is  = "<<Age<<endl;
cout<<"the Age is  = "<<stage<<endl;
}
};

void main()
{
string name,number,email;
int age , stage;
cin>>name>>number>>email>>age>>stage;
Persna Pl;
Pl. show_Data(name,number,email,age,stage);
}
```

مثال 3- كيفية كتابة برنامج باستخدام ال Class عمل حاسبة ؟

```
#include <iostream.h>
class Cla
{
private :
int x ,y;

public :
void Set_Date(int a, int b)
{
 x= a;
```

```
 y= b;
}
int sum()
{
   return x+y;
}
int sub()
{
   return x-y;
}
int Div()
{
   return x/y;
}
int multi ()
{
   return x*y;
}

} ;

void main()
{
int x1 , x2;
cin>>x1 , x2 ;
Cla C;
C.SetDate(x1,x2);
C.Sum();
C.Sub();
C.Div();
C.multi();
}
```