المصفوفات:

هي عبارة عن مجموعة ذات حجم ثابت من العناصر التي لها نفس نوع البيانات ولكنها تختلف في القيم المُسندة إليها.

يحتاج المبرمج في كثير من الأحيان إلى تخزين البيانات بعد قراءتها بهدف الرجوع إليها في إجراء بعض العمليات، وعندها يحتاج إلى التصريح وبشكل منفصل عن عدة متحولات لها نفس نوع البيانات مثلاً num1, num2, ...num50 ، وهنا تبرز أهمية المصفوفات في تخزين قيم عدة متحولات من نفس النوع ضمن تسلسل معين مما يسهّل الوصول إليها والتعامل معها لاحقاً، حيث يتم تعريف متحول واحد من نوع مصفوفة وليكن على سبيل المثال numbers ، ثم يتم التعامل مع عناصر هذه المصفوفة بشكل مستقل عن طريق الـ Index (الدليل)الخاص بهذا العنصر

numbers[0], numbers[1], numbers[2],.....

أنواع المصفوفات:

- المصفوفات أحادية البعد (vector).
- المصفوفات ثنائية البعد.

التصريح عن مصفوفة أحادية البعد: يتم التصريح عنها بالشكل التالي:

```
type  array-name[n];
```

حيث:

Type : نوع عناصر المصفوفة (int , float , char, ...).

arrayName : اسم المصفوفة.

n : عدد عناصر المصفوفة

مثال:

```
int arr [ 5 ] ;
```

هنا تم التصريح مصفوفة اسمها arr مؤلفة من 5 أعداد صحيحة.

للوصول إلى أي عنصر من عنصر المصفوفة لا بد من استخدام دليل يدل على هذا العنصر، علماً ان ترتيب عناصر المصفوفة يبدأ من الصفر، وبالتالي فإن:

ترتيب آخر عنصر من عناصر المصفوفة = عدد العناصر - ١

مثال : اذا كانت عناصر المصفوفة X بالشكل التالي [0 , 7, 4, 3, 6, 2] فإن :

x[0]=2

x[1]=6

x[2]=3

x[3]=4

x[4]=0

إدخال قيم عناصر المصفوفة:

يتم التعامل مع عناصر المصفوفة كأي مجموعة من المتغيرات المشتركة بالنوع، لكن كونها تقع ضمن مصفوفة فإن هذا الأمر يقدم مجموعة من الميزات كإمكانية العودة إليها باستمرار كما ذكرنا سابقا، وكذلك إمكانية إدخال أو قراءة قيمها بشكل أبسط وأسرع من إدخال قيمة كل متغير على حدا، الأمر الذي يصبح معقدا جدا عند التعامل مع أعداد ضخمة من المتغيرات، يمكننا استخدام الحلقات للتعامل مع عناصر المصفوفة مثل حلقة For،

مثال : اكتب برنامج لإدخال عناصر مصفوفة مؤلفة من 10 عناصر:

```
#include <iostream.h>
main()
{
int i ;
```

```
int  myarr [10] ;
for ( i = 0 ; i < 10 ; i++)
    {
      Cout << " myarr[ "<< i << " ]= " ;
       Cin >> myarr[ i ];
      }
}
```

مثال : اكتب برنامج لإدخال عناصر مصفوفة مؤلفة من 10 عناصر، ثم طباعة العنصر الأكبر من عناصر المصفوفة.

```
#include <iostream.h>
main()
{
int i ;
int max;
int  arr [10] ;
for ( i = 0 ; i < 10 ; i++)
    {
       cin >> arr[ i ];
      }
max = arr[0];
for (  i = 0 ; i < 10 ; i++)
    {
      if (arr[ i ] > max)
        max = arr[ i ];
```

```
    }
cout <<"Maximum number of array is " << max;
}
```

المصفوفات ثنائية البعد: تتكون من أكثر من سطر وأكثر من عمود ، ويتم التصريح عنها بالشكل التالي:

```
type  array_name[n][m];
```

حيث:

type : نوع عناصر المصفوفة (int , float ,.....).

array-name : اسم المصفوفة.

n : عدد أسطر المصفوفة.

m : عدد أعمدة المصفوفة.

مثال: `int  arr [ 5 ][ 3 ] ;`

هنا عرفنا مصفوفة اسمها arr مؤلفة من 5 أسطر و ٣ أعمدة ، عناصر هذه المصفوفة أعداد صحيحة.

للوصول إلى أي عنصر من عنصر المصفوفة لا بد من استخدام دليل يدل على هذا العنصر، في المصفوفات الثنائية نحتاج إلى دليل للأسطر ودليل للأعمدة.

مثال : اكتب برنامج لإدخال عناصر مصفوفة مؤلفة من 3 أسطر و٤ أعمدة:

```
#include <iostream.h>
main()
{
int i , j ;
int  x [3][4] ;
for ( i = 0 ; i < 3 ; i++)
for (  j = 0 ; j < 4  ; j++)
   {
          Cin >> x[ i ][ j ] ;
    }
 }
```

مثال : اكتب برنامج لجمع عناصر مصفوفتين مربعتين تتألف كل منهما من ٣ أسطر و٣ أعمدة، ثم طباعة عناصر المصفوفة الناتجة بالشكل الرياضي المتعارف عليه في المصفوفات.

ملاحظة: المصفوفة المربعة هي المصفوفة التي عدد الأسطر فيها يساوي عدد الأعمدة.

الحل: لجمع مصفوفتين نجمع العنصر الاول من المصفوفة الاولى مع العنصر الاول المقابل من المصفوفة الثانية ثم العنصر الثاني من المصفوفة الاولى مع العنصر الثاني المقابل من المصفوفة الثانية.

```
#include <iostream.h>
main()
{
int  x [ 3 ][ 3 ] ;
int  y[ 3 ][ 3 ] ;
```

```
int  z [ 3 ][ 3 ] ;
cout<<" Enter elements of first Array:";
for (   i = 0 ; i < 3 ; i++)
for (   j = 0 ; j < 3  ; j++)
   {
          Cin >> x[ i ][ j ] ;
     }
cout<<" Enter elemnts of  Second Array: ";
for ( i = 0 ; i < 3 ; i++)
for ( j = 0 ; j < 3  ; j++)
   {
          Cin >> y[ i ][ j ] ;
     }
for ( i = 0 ; i < 3 ; i++)
for ( j = 0 ; j < 3  ; j++)
   {
          z[ i ][ j ] =  x[ i ][ j ] + y[ i ][ j ] ;
     }
for ( i = 0 ; i < 3 ; i++)
  {
     for ( j = 0 ; j < 3  ; j++)
         Cout <<   z[ i ][ j ] << " \t ";
     Cout << " \n ";
   }  }
```

مثال : اكتب برنامج يطلب من المستخدم تحديد أبعاد مصفوفة ثنائية البعد ، ثم إدخال عناصر هذه المصفوفة، ثم حساب وطباعة مجموع عناصر القطر الرئيسي في المصفوفة،

```
#include <iostream.h>
main()
{
int i , j ;
int m , n ;
int sum = 0 ;
int  myarr [ m ][ n ] ;
cout << "Enter number of rows in the Array";
cin >> m ;
cout << "Enter number of columns in the Array";
cin >> n ;

for ( i = 0 ; i < m ; i++)
for (  j = 0 ; j < n  ; j++)
   {
          Cout << " myarr [ " << i << " ][" << j << " ] =";
           Cin >> myarr[ i ][ j ] ;
           Cout << " \n ";
   }
for ( i = 0 ; i < m ; i++)
for (  j = 0 ; j < n  ; j++)
   {
```

```
        if ( i == j )
            Sum = sum + myarr [ i ][ j ] ;
    }
Cout << " sum = " << sum ;
}
```

الدوال (التوابع) Functions:

الدالة هي جزء من البرنامج تقوم بأداء مهمة معينة وتعيد نتيجة ما، ويمكن أن تُمرر لها بارمترات من البرنامج المستدعي.

الهدف الأساسي من استخدام الدوال هو منع تكرار كتابة التعليمات البرمجية لأكثر من مرة، مما يؤدي إلى تقليل أسطر البرنامج، وبالتالي سهولة تتبع الأخطاء في حال حدوثها في البرنامج، هذا بالإضافة إلى أن الدوال تجعل البرنامج منظم بشكل أفضل.

يمكن استخدام الدالة بعد التصريح عنها، عن طريق استدعاء هذه الدالة في البرنامج أو في دالة أخرى، وذلك بذكر اسم الدالة وتمرير قيم مناسبة للبارمترات الخاصة بهذه الدالة.
عند استدعاء دالة ما، ينتقل سير البرنامج إلى تلك الدالة ويتم تنفيذ التعليمات الموجودة ضمنها، ثم تتم العودة إلى البرنامج الذي قام بالاستدعاء.

التصريح عن الدالة (التابع): يتم التصريح عن التابع أو الدالة بالشكل التالي:

```
Type function_name(type parameter1, type parameter2 ,…..)
{
    التصريح عن المتغيرات المحلية الخاصة بالدالة

  ……..
    تعليمات الدالة

  ……..
  return …….
}
```

حيث:
Type : نوع القيمة التي ستعيدها الدالة (int , float ,....)، اذا كانت الدالة تعيد قيم
Function_name : اسم الدالة.
Type parameter1 : نوع واسم الوسيط الاول من وسطاء الدالة.

Type parameter2: نوع واسم الوسيط الثاني من وسطاء الدالة.

.....

ملاحظات:

١. يجب أن يتطابق نوع القيمة التي ستتم إعادتها في تعليمة return مع النوع المحدد في القيمة المعادة type.

٢. في حال كانت الدالة لا تعيد قيمة عندها تكون القيمة المعادة type هي **void**، وعندها لا نستخدم تعليمة return .

٣. يجب أن يتطابق عدد ونوع البارمترات المستخدمة عند التصريح عن الدالة مع عدد ونوع البارمترات التي سيتم تمريرها إلى هذه الدالة عند استدعائها.

٤. يتم التصريح عن الدوال عادة قبل الدالة الرئيسية Main.

المتغيرات المحلية (local) والمتغيرات العامة (global):

المتغيرات المحلية: هي المتغيرات التي يتم الإعلان عنها داخل الدالة، وتسمى بهذا الاسم لأنها تكون غير معروفة خارج الدالة المعرفة ضمنها، ومن الجدير بالذكر أن المتغير المحلي تنشأ له قيمة وكيان عند ابتداء تنفيذ الدالة التي ينتمي إليها فقط، وتختفي عند الانتهاء من تنفيذ تلك الدالة.

المتغيرات العامة: هي التي تكون قيمتها معروفة من أول البرنامج إلى آخره، ويمكن استعماله في أي جزء منه، وتحتفظ بقيمتها أثناء تنفيذ البرنامج، يتم التصريح عن هذه المتغيرات خارج حدود الدوال، ومن المعتاد عملياً التصريح عن هذه المتغيرات في بداية البرنامج.

تمرين ١: اكتب تابع يقوم بطباعة العبارة " Hello , in C++ " على الشاشة :

```
#include <iostream.h>
void  func1 ( )
{
   Cout<< Hello , in C++ " ;
```

```
}

main()
{
  func1( ) ;

}
```

نلاحظ من المثال السابق أن الأقواس ضرورية عند التصريح عن الدالة وكذلك عند استدعائها حتى وان لم تكن هذه الدالة تستخدم أية بارمترات.

تمرين ٢: اكتب دالة تقوم بجمع عددين وإعادة النتيجة .

```
#include <iostream.h>
int   add  ( int  x , int  y )
{
   int   z ;

   z = x +  y ;

return  z ;

}

main ( )
{
 int  num1 , num2  , result ;

cout << "Enter first number: " ;

cin >> num1 ;

cout << "Enter second number: " ;
```

```
cin >> num2 ;

result = add (num1 , num2);

cout<< " Result is : " << result ;

}
```

تمرين : اكتب برنامج للتعامل مع المصفوفات أحادية البعد يحوي الدوال التالية:

١. دالة لإدخال عناصر المصفوفة.
٢. دالة لطباعة عناصر المصفوفة.
٣. دالة لجمع عناصر المصفوفة.
٤. دالة لإيجاد اكبر عنصر في المصفوفة.
٥. دالة لترتيب عناصر المصفوفة تصاعدياً.

```
#include<iostream.h>
void read ( int x [ ] , int y )
{
   for( int i=0 ; i< y ; i++)
    {
        cout<< "x[" << i << "]=" ;
        cin>> x[ i ] ;
    }
}
void print( int x [ ] , int y )
{
   cout <<" [ ";
   for( int i=0 ; i< y ; i++)
```

```
        {
          cout<< x[ i ]<< " \t ";
        }
   cout  << " ] " ;
}
int add ( int x [ ] , int  y)
{
   int sum=0 ;
   for( int i=0 ; i< y ; i++)
                sum= sum +x [ i ] ;
    return sum ;
}
int findmax( int  x[ ] , int  y)
{
        int g = x[ 0 ] ;
        for( int  i = 0 ; i<y ; i++)
         {
                If ( x[ i ] > g )
                     g = x [ I ] ;
         }
        return g ;
}
void sort ( int  x [ ] , int  y )
{
```

```
   int  p ;
   for ( int i=0 ; i < y ; i++)
   for( int j=i ; j < y ; j++)
        {
          if ( x[ j ] < x[ i ] )
               {
                  p = x[ j ];
                  x[ j ] = x[ i ] ;
                   x[ i ] = p ;
               }
}
cout<< " After Sort "<< " \n " << " x = [ " ;
for( i = 0 ; i < y ; i++ )
cout << x[ i ]<<" \t " ;
cout<< " ] " ;
}
main()
{
 const  int  n = 5 ;
 int x[ n ] , s[ n ] ;
 cout <<" Please, Enter Array elements";
 read( x, n) ;
 cout << " \n " <<  " Array Elements are: "
 print( x , n ) ;
```

```
cout<< " \n ";
cout<< " Sum of Array Elements = " << add(x , n) << " \n " ;
cout<< " Max Number in Array = " << max( x , n ) << " \n " ;
sort( x , n ) ;
}
```

التراكيب (السجلات) Structures:

هي مجموعة متحولات **من أنواع مختلفة**،

التصريح عن التركيب (السجل): يتم التصريح عن التركيب أو السجل بالشكل التالي:

```
struct  structure_name
  {
    Type  var1 ;
    Type  var2 ;
    ........
    ..........
    .......
  }   object_name1 , object_name2;
```

حيث:

structure_name : اسم التركيب أو السجل.

Type var1 : نوع واسم المتحول الاول من متحولات السجل.

Type var2 : نوع واسم المتحول الثاني من متحولات السجل

.....

Object_name : هو اسم الغرض المشتق من السجل.

مع ملاحظة أنه يمكن وضع أنواع مختلفة من المتحولات ضمن السجل.

يتم التعريف عن السجل قبل الدالة ( ) main.

للوصول إلى أي عنصر (حقل) من عناصر السجل، نستخدم اسم الغرض المشتق من السجل متبوعاً بنقطة يليه اسم هذا العنصر.

مثال:

```
#include<iostream.h>
#include<string.h>
Struct student
{ char name[10];
  int id;
  float average;
 } std1;
main()
{
Cout<< " Enter student name: " ;
Cin >>std1.name;
Cout<< " Enter student  id: " ;
Cin >>std1.id;
Cout<< " Enter student average: " ;
Cin >>std1.avarage ;
student  std2;
std2.name = "Ahmad";
std2.id = 123;
std2.average = 99.5;
cout << " Student  Name is " << std2.name << "\n" ;
cout << " Student  id is " << std2.name << "\n" ;
cout << " Student average is " << std2.average  ;
}
```

الفئات (الأصناف) Classes:

يعتبر مفهوم الفئات واحد من أفضل ميزات لغة C++،

الفئة أو الكلاس هي مجموعة من البيانات والدوال التي تعمل على هذه البيانات.

التصريح عن الكلاس (الفئة): يتم التصريح عن الكلاس بالشكل التالي:

```
class  class_name
    {
    private:
      Type  var1 ;
      Type  var2 ;
      ........
    Public:
      Type  var3 ;
      Type  var4 ;
      .........
      ........
    };
```

حيث:

class_name : اسم الكلاس .

القسم private : في هذا القسم يتم تعريف المتحولات الخاصة بالكلاس.

Type var1 : نوع واسم المتحول الاول من متحولات الكلاس.

Type var2 : نوع واسم المتحول الثاني من متحولات الكلاس.

.....

القسم public : في هذا القسم يتم تعريف المتحولات العامة .

ملاحظات:

- يتم تعريف الكلاس قبل الدالة ( ) main .
- يبدأ تعريف الكلاس بالكلمة المحجوزة class يليها اسم اختياري للصف (يخضع لقاعدة تعريف الأسماء). يتم تحديد جسم الكلاس بواسطة قوسين { }، وينتهي تعريف الكلاس دوماً بالفاصلة المنقوطة (;).
- ينقسم جسم الكلاس إلى قسمين أساسيين، يبدأ كل قسم بكلمات مفتاحيه تسمى المحددات يحدد كل منها مستوى الوصول لهذا القسم :

  المحدد: public (ويعني عام)

  المحدد: private (يعني خاص أو داخلي)
- أعضاء الكلاس قد تكون متحولات أو دوال (توابع).
- الأعضاء التي تُعرّف في قسم Private لا يمكن التعامل معها إلا ضمن الكلاس نفسه،
- الأعضاء التي تُعرّف في قسم Public هي فقط التي يمكن التعامل معها ضمن الدالة الرئيسية للبرنامج ( ) main.
- في تعريف الكلاس لا يمكن إسناد قيم لمتحولات الكلاس عند التصريح عنها ..من هنا تبرز أهمية التابع الباني الذي نقوم فيه بتهيئة المتحولات بقيم ابتدائية.
- عند تعريف التوابع الأعضاء بعد تعريف الكلاس تستخدم العملية الثنائية (::).
- بعد أن يتم تعريف الكلاس يتم استخدام اسمه لإنشاء نسخ من هذا الصنف تسمى الأغراض objects (اسم الكلاس أصبح نمط بيانات جديد)،
- ان التصريح عن الكلاس لا يؤدي إلى حجز أي مساحة فعلية في الذاكرة، يتم حجز مكان في الذاكرة عند تعريف غرض (object) من هذا الكلاس.
- بعد التصريح عن الغرض تُستدعى الأعضاء المتعلقة بالغرض في البرنامج بكتابة اسم الغرض ثم (.) ثم اسم العضو.

التابع الباني Constructor:

- يستخدم لتهيئة المتحولات الأعضاء في الكلاس بقيم ابتدائية.
- يُعرّف ضمن قسم public .
- له نفس اسم الكلاس.
- لا يعيد أي قيم.
- يتم استدعاء هذا التابع تلقائياً بمجرد التصريح عن غرض من هذا الكلاس.
- يمكن تعريف أكثر من تابع باني ضمن الكلاس مع تغيير البارمترات المررة لهذا التابع.

مثال:

اكتب كلاس يعبر عن دائرة، وعرّف فيه تابع لحساب محيط الدائرة وتابع آخر لحساب المساحة، علماً أن نصف قطر الدائرة سيتم إدخاله من قبل المستخدم .

```
#include<iostream.h>
float  const   pi = 3.14 ;

class circle
{
   private:
   public:
     float radius ;          // نصف قطر الدائرة
    circle ()                // التابع الباني بدون بارمترات
         {
          radius = 10 ;
         }

    circle ( float  rr )   // التابع الباني مع بارمترات
         {
          radius = rr ;
         }
```

```
    float  calc_area()    // دالة حساب المساحة
    {
     float   area = pi * radius * radius ;
     return  area;
     }

   float calc_circ()     // دالة حساب المحيط
   {
     float  circ = 2 * pi * radius ;
     return  circ ;
    }
};

int main (int  argc  , char *argv[ ] )
{
  float  cr_r  ;
  circle  c1;       //تعريف غرض من الكلاس
   cout << " Radius of circle is: " << c1.radius << " \n " ;
   cout << "Area of circle is: " << c1.calc_area( )<< " \n " ;
   cout << "Circumference  of circle is: " << c1.calc_circ( )<< " \n " ;

   circle  c2 ( 3 ) ;      // تعريف غرض آخر من الكلاس مع تمرير نصف القطر ليتم تنفيذ التابع الباني الثاني
   cout << " Radius of circle is: " << c2.radius << " \n " ;
   cout << "Area of circle is: " << c2.calc_area( )<< " \n " ;
   cout << "Circumference  of circle is: " << c2.calc_circ( )<< " \n " ;
```

```
    cout << "Enter Radius of circle: " ;
    cin >> cr_r ;
    circle  c3 ( cr_r ) ;      // تعريف غرض آخر وهنا تم تمرير قيمة مدخلة من قبل المستخدم
    cout << " Radius of circle is: " << c3.radius << " \n " ;
    cout << "Area of circle is: " << c3.calc_area( )<< " \n " ;
    cout << "Circumference  of circle is: " << c3.calc_circ( )<< " \n " ;

    return 0;
}
```