ARRAYS

&

FUNCTIONS

المصفوفات ARRAYS

Pointer to an array مؤشر الي مصفوفة

An array name is a constant pointer to the first element of the array. Therefore, in the declaration:

```
double balance[50];
```

balance is a pointer to &balance[0], which is the address of the first element of the array balance. Thus, the following program fragment assigns **p** the address of the first element of **balance**:

```
double *p;
double balance[10];

p = balance;
```

It is legal to use array names as constant pointers, and vice versa. Therefore, *(balance + 4) is a legitimate way of **accessing the data at balance[4]**.

Once you store the address of first element in p, you can access array elements using *p, *(p+1), *(p+2) and so on.

Below is the example to show all the concepts discussed above:

```
Void main ()
{
   double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
   double *p;
   p = balance;
   cout << "Array values using pointer " << endl;
   for ( int i = 0; i < 5; i++ )
   {
       cout << "*(p + " << i << ") : "; cout << *(p + i) << endl;
   }
   cout << "Array values using balance as address " << endl;
   for ( int i = 0; i < 5; i++ )
   {   cout << "*(balance + " << i << ") : ";
       cout << *(balance + i) << endl;   }
}
```

When the above code is compiled and executed, it produces following result:

```
Array values using pointer
*(p + 0) : 1000
*(p + 1) : 2
*(p + 2) : 3.4
*(p + 3) : 17
*(p + 4) : 50
Array values using balance as address
*(balance + 0) : 1000
*(balance + 1) : 2
*(balance + 2) : 3.4
*(balance + 3) : 17
*(balance + 4) : 50
```

Passing array to a function تمرير مصفوفة الي دالة

C++ does not allow to pass an entire array as an argument to a function. However, You **can pass a pointer** to an array by specifying the array's name without an index.

If you want to pass a single-dimension array as an argument in a function, you would have to declare function formal parameter in one of **following three ways** and all three declaration methods **produce similar results** because **each tells** the compiler that an integer pointer is going to be received **(تمرير مؤشر لعدد صحيح)**.

Way-1
Formal **parameters as a pointer** as follows:

```
void myFunction(int *param){...}
```

Way-2

Formal **parameters as a sized array** as follows:

```
void myFunction(int param[10]){...}
```

Way-3

Formal **parameters as an unsized array** as follows:

```
void myFunction(int param[]){...}
```

```
# include<iostream.h>
double getAverage(int arr[], int size)
{   int   i, sum = 0;
    double avg;
    for (i = 0; i < size; i++)  {     sum += arr[i];    }
    avg = double(sum) / size;
    return avg;
}
void main ()
{
int balance[5] = {1000, 2, 3, 17, 50};
// pass pointer to the array as an argument.
double avg = getAverage( balance, 5 ) ;
// output the returned value
cout << "Average value is: " << avg << endl;     }
```

Return array from function الرجوع بمصفوفة من الدالة

C++ does not allow to return an entire array as an argument to a function. However, **You can return a pointer to an array** by specifying the array's name without an index.

If you want to return a single-dimension array from a function, you would have to declare a function returning a pointer as in the following example:

```
int * myFunction()
{
.
.
.
}
```

Example:

Pass an array arr[] to the function to change arr[] items by adding 10 to it ,then return the modified array arr[].

```
int* getAverage( int arr[], int size)
{
  int *p1;
  for (int i = 0; i < size; i++)
  {    arr[i]=arr[i]+10;     }
  p1=arr;
  return p1;
}
void main()
{   int a[]={1,2,3,4,5,6};
int *d = getAverage(a,6);
cout<<*d<<"      "<<*(d+1)<<"      "<<*(d+2)<<"      "<<*(d+3)<<"
"<<*(d+4)<<"  "<<*(d+5);
}
```

FUNCTIONS الدوال

المكتبات في C++ هي تجمع من الدوال + جمل برمجية داخل ملف رأس header file

الدوال Functions

- الدالة هي مجموعة من الجمل البرمجية مجمعة معا لاداء مهمة معينة.
- برنامج c++ يحتوي علي دالة واحدة علي الاقل و هي الدالة main().

انواع الدوال

- Built – in

sqrt() , rand(),.....

- User - defined

مثال:

ايجاد الجزر التربيعي لعدد مع طباعة العدد و الجذر التربيعي له

```
# include<iostream.h>
# include<math.h>
Void main()
{
    int x;
    cin>>x;
    cout<<"\t"<<x<<"\t"<<sqrt(x)<<endl;
}
```

- الدالة sqrt تشبه الصندوق الاسود ندخل فيه العدد يخرج منه الجذر التربيعي له.

User – defined functions

- دالة لا يمرر اليها قيم ولا تعود بنتيجة
- دالة لا يمرر اليها قيم وتعود بنتيجة
- دالة يمرر اليها قيم وتعود بنتيجة
- دالة يمرر اليها قيم ولا تعود بنتيجة

اقسام الدالة

- الاعلان عن الدالة definition
- تعريف الدالة declaration
- استدعاء الدالة calling

الاعلان عن الدالة definition

- يسمى رأس الدالة وموقعه قبل الدالة الرئيسية أو داخلها وينتهى بـ ; و يقوم باخبار المترجم عن اسم الدالة و نوعها و المعاملات الممررة لها ان وجدت.

تعريف الدالة declaration

- يسمى جسم الدالة و موقعه قبل الدالة الرئيسية مع جملة الاعلان (مدمجا معها) أو بعد الدالة الرئيسية

استدعاء الدالة calling

- داخل الدالة الرئيسية أو داخل اى دالة اخرى حسب الحاجة

مثال:

- انشئ دالة cube تقوم بضرب ثلاثة اعداد بالأشكال الاربعة السابقة

دالة لا يمرر اليها قيم ولا تعود بنتيجة

```
# include<iostream.h>
Void cube ( );        // definition
Void main()
{
    cube( );          // call
    …..
    cube( );
}
Void cube ( )         // declaration
{
   int x1,x2,x3;            cin>>x1>>x2>>x3;
    cout<<x1*x2*x3<<endl;
}
```

دالة لا يمرر اليها قيم و تعود بنتيجة

```
# include<iostream.h>
 int cube ( )          // definition
{
     int x1,x2,x3,z;          cin>>x1>>x2>>x3;
     z=x1*x2*x3;              return z;
}
Void main()
{
     int r=cube( );          // call
     …..
     cout<<cube( );
}
```

دالة يمرر اليها قيم و تعود بنتيجة

```
# include<iostream.h>
Int cube ( int x , int y , int z)
{ return (x*y*z); }
Void main()
{
    int z,w,e,r
    z=cube(3,4,5);
    cin>>w>>e>>r;
    cout<<"\t"<<cube(w,e,r)<<endl;
}
```

دالة يمرر اليها قيم و لا تعود بنتيجة

```
# include<iostream.h>
Void cube ( int , int , int );
Void main()
{
    cube(2,6,8);
     int w,e,r;
     Cin>>w>>e>>r;
     Cube(w,e,r);
}
Void cube ( int a, int b, int c)
{
    cout<< a*b*c ;
}
```

الاستدعاء بالقيمة call by value

- يتم اخذ نسخة من قيمة المتغيرات الفعلية التي ستمرر للدالة في الاستدعاء فعند حدوث تغير في المعاملات داخل الدالة لا يؤثر علي قيمة المعاملات الفعلية.

الاستدعاء بالإشارة call by reference

- يتم اخذ نسخة من عنوان المتغيرات الفعلية التي ستمرر للدالة في الاستدعاء و وضعها في المعاملات الممررة للدالة فعند حدوث تغير في المعاملات داخل الدالة فانه يؤثر علي قيمة المعاملات الفعلية.

مثال:

```
#include <iostream>
void swap(int x, int y);      // function declaration
void  main ()
{
   int a = 100;       int b = 200;

   cout << "Before swap, value of a :" << a << endl;
   cout << "Before swap, value of b :" << b << endl;

   // calling a function to swap the values.
   swap(a, b);

   cout << "After swap, value of a :" << a << endl;
   cout << "After swap, value of b :" << b << endl;
}
```

```
// function definition to swap the values.
void swap(int x, int y)
{
   int temp;
   temp = x;       /* save the value of x */
   x = y;           /* put y into x */
   y = temp;        /* put x into y */
}
```

ناتج التنفيذ

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :100
After swap, value of b :200
```

مثال:

```
#include <iostream>
void swap(int *x, int *y);
Void main ()
{
   int a = 100;       int b = 200;

   cout << "Before swap, value of a :" << a << endl;
   cout << "Before swap, value of b :" << b << endl;

   swap(&a, &b);

   cout << "After swap, value of a :" << a << endl;
   cout << "After swap, value of b :" << b << endl;
}
```

```
// function definition to swap the values.
void swap(int *x, int *y)
{
   int temp;
   temp = *x;    /* save the value at address x */
   *x = *y;         /* put y into x */
   *y = temp;    /* put x into y */
}
```

ناتج التنفيذ

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :200
After swap, value of b :100
```

Passing array to a function تمرير مصفوفة للدالة

❖هناك ثلاث طرق لتمرير مصفوفة للدالة:

١- نمرر مؤشر للمصفوفة.

```
void myFunction(int *param)
{...}
```

٢- نمرر مصفوفة ذو حجم محدد.

```
void myFunction(int param[10])
{...}
```

٣- نمرر مصفوفة غير محددة الحجم.

```
void myFunction(int param[])
{...}
```

مثال:

صممي دالة getAverage يمرر اليها المصفوفة arr و حجم المصفوفة ، ثم احسبي مجموع عناصر المصفوفة و العودة بالمجموع.

```
# include<iostream.h>
int getAverage(int arr[], int size)
{
   int   i,sum=0;
   for (i = 0; i < size; i++)       sum += arr[i];
   return sum;
}
void main ()
{   int balance[5] = {10, 2, 3, 17, 50};
    int avg = getAverage( balance, 5 ) ;  // pass pointer to the array
    cout << avg << endl;
 }
```

الرجوع بمصفوفة من الدالة

عن طريق العودة بمؤشر للمصفوفة.

```
int * myFunction()  {  . . .  }
```

مثال:

صممي دالة getAverage يمرر اليها المصفوفة arr، حيث يتم اضافة القيمة 10 لجميع عناصر المصفوفة و العودة بالمصفوفة.

```
int* getAverage( int arr[3])
{   int *p1;
    for (int i = 0; i<3; i++)      arr[i]=arr[i]+10;
    p1=arr;      return p1;
}
void main(){
    int a[]={1,2,3};
     int *d = getAverage(a);
     cout<<*d<<"  "<<*(d+1)<<"  "<<*(d+2);   }
```

انواع دوال السطر

- ضمنية implicit (مكونه من سطر واحد و لا نذكر inline).
- صريحة explicit (مكونه من سطر او عدة اسطر مع ذكر inline)

مثال : (ضمنية)

```
Int max (int a, int b)
{ if(a>b) return a ;    else return b; }
```

مثال : (ضمنية)

```
Int max (int a, int b){
    return ((a>b) ? a : b) ; }
```

مثال : (صريحة)

```
inline float zakat(float m,int nesab , int month)
{
    float z;
    if(m>nesab && month>12)  z=2.5*m/100;
    else   z=0;
    return  z;
}
```

- Inline expansion مفيد للدوال ذات السطر لان حجم البرنامج لا يتغير باستبدال الكود مكان الاستدعاء على عكس الدوال متعددة الاسطر .

تمرير تركيبة للدالة

صممي تركيبة Books حيث تحتوي علي عنوان الكتاب ، رقم الكتاب، دالة printBook يمرر اليها التركيبة Books لطباعة بياناتها.

```
#include <iostream>
struct Books {
   char title[50];
   int id;
}Book1;
void printBook(struct Books b1)
{   cout<<b1.title<<"   "<<b1.id<<endl;   }
void main()
{
   strcpy(Book1.title,"Learn C++ Programming");
   Book1.id = 12;
   printBook(Book1);
}
```