المحاضرة السابعة

البرمجة 1

(*Programming Language 1*)

إعداد

الدكتور المهندس فراس الزين

الكلمات المفتاحية

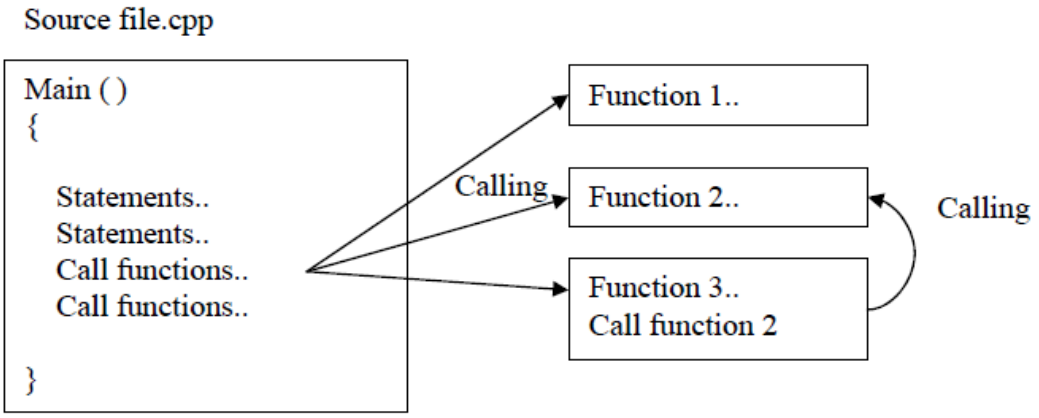
التوابع , استدعاء , العودة , المستخدم , الإجراء , بارامتر .

Function , calling , return , user , procedure , parameter .

الدوال Functions:

وهي عبارة عن برامج فرعية تشبه البرنامج الرئيسي main ، البرنامج main هو عبارة عن دالة تتميز بأن بيئة C++ تقوم بتشغيلها ، وتتولى دالة main تشغيل ما بداخلها واستدعاء الدوال الفرعية وتشغيلها.

تعريفها:
- مجموعة من التعليمات والأنشطة المكتوبة داخل برنامج مستقل (فرعي) يتم استدعائها داخل البرنامج الرئيسي.



ملاحظة:
استدعاء الدوال وتفعيلها يجب أن يكون من داخل الدالة الرئيسية main .

مميزات الدوال:

١- تقسيم البرامج الكبيرة إلى أجزاء صغيرة داخل البرنامج "Source file".
٢- اختصار الكثير من السطور في اسم الدالة (ذات الطابع التكراري).
٣- تنظيم البرنامج.
٤- قابلية استخدامها أكثر من مرة.
٥- سهولة التطوير والتعديل واكتشاف الأخطاء.

أنواع الدوال:

- دوال قياسية Standard function:
كل الدوال الموجودة في المكتبات مثل pow() و cout ولا يمكن تعديلها.

- دوال من تعريف المستخدم User define function:
الدوال التي يقوم المستخدم بتكوينها.

أشكال الدوال:

- **إجراء Procedures:**
وهي دالة تقوم بعمل معين وتنفذه أو تطبعه على الشاشة، و تسمى (إجراء) لأنها لا تعيد قيمة.

- **دالة Function:**
وهي دالة تقوم بعمل معين وتعيد قيمة ويمكنها أن تطبع شيء على الشاشة وتعيد قيمة في نفس الوقت، ويمكن إسناد هذه القيمة إلى متغير ثم طباعته على الشاشة.

الصيغة العامة للإجراء:

التصريح

```
void FunctionName(parameters)
{
```

البناء

```
    Statements...
{
```

الصيغة العامة للدالة:

- استخدام الكلمة المحجوزة return لإعادة قيمة:

```
DataType FunctionName(parameters)
{
    Statements...
    return value;
{
```

- استخدام نفس اسم الدالة لإعادة قيمة:

```
DataType FunctionName(parameters)
{
    Statements...
    FunctionName = value;
{
```

يمكن أن يحتوي الإجراء أو الدالة على باراميترات Parameters ويمكن ألا تحتوي عليها، فهذا يرجع للمبرمج.

مثال لإجراء:

```
1. void sum(int a, int b)
2. {
3.       cout << a + b;
4. {
```

استدعاء الإجراء داخل البرنامج:

```
sum( 7 , 8 );
```

الناتج: 15

ملاحظات:

- لا يمكن إسناد الإجراء لمتغير فهو لا يعيد أي قيمة.
- لا يمكن إدخال الإجراء ضمن عمليات حسابية.

مثال لدالة:

```
1. int sum(int a, int b)
2. {
3.       Return (a + b);
4. {
```

استدعاء وطباعة الدالة داخل البرنامج "سيتم طباعة ناتج الدالة":

```
cout << sum( 7 , 8 );
```

الناتج: 15

أو استدعاء وإسناد الدالة إلى متغير "سيتم إسناد ناتج الدالة لمتغير":

```
1. int s;
2. s = sum( 7 , 8 );
3. cout << s;
```

الناتج: 15

استدعاء وطباعة الدالة داخل البرنامج وإدخالها ضمن عملية حسابية "سيتم في العلمية الحسابية التعامل مع ناتج الدالة كقيمة":

```
1. cout << sum( 7 , 8 ) + 5;
```

الناتج: 20

استدعاء وإسناد الدالة إلى متغير وإدخالها ضمن عملية حسابية.. "سيتم إضافة ناتج الدالة إلى العملية الحسابية":

```
1. s = sum( 7 , 8 ) + 5;
2. cout << s;
```

الناتج: 20

تقديم نوع البيانات int قبل اسم الدالة (int function_name) يعني أن الدالة تعيد قيمة من النوع int بينما تقديم الكلمة void قبل اسم الدالة (void function_name) يعني أن الدالة لا تعيد أي قيمة من أي نوع.

تعريف الدوال :

- يمكن تعريف الدوال قبل الدالة الرئيسية main (التصريح والبناء معاً).

```
1.  int sum(int a, int b)
2.  {
3.      return a + b;
4.  {
5.
6.  void main()
7.  {
8.      cout << sum( 5 , 4 );
9.  }
```

- يمكن تعريف الدوال تحت الدالة الرئيسية main بشرط التصريح عنها قبل الدالة.

التصريح عن الدالة

طباعة الدالة "ناتجها"

بناء الدالة

```
1.  int get_sum( int , int );
2.
3.  void main()
4.  {
5.      cout << get_sum( 5 , 4 );
6.  }
7.
8.  int get_sum(int a, int b)
9.  {
10.     return a + b;
11. }
```

تطبيق المثال :

```
#include <iostream.h>

int get_sum( int , int );
////////////////////////////////
int main(){

    cout << get_sum()
};          int get_sum ( int a=0, int b=0)
////////////////////////////////
int get_sum( int a=0, int b=0){
    return a + b;
}
```

تمرين :

قم بعمل مثلث باسكال يقوم برسم عدة أشكال للمثلث باستخدام دالة واحدة تحوي دالة for واحدة، حيث يتم تغيير شكل المثلث بالتلاعب بقيمة متغيرات الدالة فقط.

القيم الابتدائية في الدوال:

تعتبر كنوع من تحديد القيمة الابتدائية لمتغيرات الدالة أو الإجراء، فيمكنك عدم إسناد أي قيمة لمتغيرات الدالة:

مثال :

```
1.  double _div( float a=0, float b=0){
2.  {
3.      if(b==0){
4.          return 0;
5.      }else{
6.          return (a / b);
7.      }
8.  }
9.
10. void main()
11. {
12.     cout << _div() << endl;
13.     cout << _div(5) << endl;
14.     cout << _div(5, 0) << endl;
15.     cout << _div(8, 5) << endl;
16. }
```

تحديد قيم افتراضية لمتغيرات الدالة "أصفار"

إرجاع 0 إذا كان المتغير الثاني صفراً

إرجاع ناتج قسمة العددين إذا كان المتغير الثاني لا يساوي الصفر.

الناتج: 0
الناتج: 0
الناتج: 0
الناتج: 1.6

ملاحظات:

- يمكن جعل إحدى متغيرات الدالة تحمل قيمة ابتدائية والأخرى لا تحمل قيمة ابتدائية فهذا يعني أنه يجب إسناد قيمة لهذا المتغير الآخر بشرط أن لا تجعل المتغيرات التي تحمل قيمة ابتدائية قبل المتغيرات التي لا تحمل قيم ابتدائية (لماذا؟ "ابحث عن هذا الموضوع").
- تم كتابة div_ تسبقها شرطة لأن هناك دالة من دوال مكتبة iostream.h تحمل الاسم div.

استخدام #Define:

يستخدم هذا الأمر ليخبر المترجم باستبدال سلسلة من الأحرف بالقيمة المجاورة للأمر define فهذا الأمر لا يفحص نوع القيمة فقد تكون قيمة أو معالجة لعملية حسابية أو غيرها كما في الدوال:

١) استخدام الأمر define لتعريف الثوابت:

الشكل العام Public formula:

```
#define Constant  value;
```

مثال :

```
1.  #define MAX  100;
2.  main ( )
3.  {
4.      cout << MAX;
5.  }
```

الناتج: 100

٢) استخدام الأمر define بدلا عن الدوال:

الشكل العام Public formula:

```
#define Function_name (parameters)  Statements...
```

مثال :

```
1.  #define SUM( x , y ) x + y;
2.  main ( )
3.  {
4.      Int z = SUM( 1 , 2 );
5.      cout << z << endl;
6.      cout << SUM(3.5, 7.5);
7.  }
```

الناتج: 3
الناتج: 11

مميزات define:

- لا يحتاج لتعريف نوع البيانات.
- لا يحتاج لتعريف نوع الدوال.
- يمكن إسناد قيمتها إلى متغير بشرط أن يكون المتغير من نفس نوع البيانات المعادة من الدالة.
- يمكن الاستغناء بها عن التحميل الزائد للدوال overload "سيتم دراسته في الفصل الثاني".

عيوب define:

- لا يمكن عمل مجموعة إجراءات "جمل" في سطور متعددة تحت الأمر define لأن المترجم سيتجاهل السطور اللاحقة ويعتبرها خطأً.

ملاحظات:

- نلاحظ أن هذه الطريقة لا تحتاج لتعريف نوع المتغيرات في الدالة SUM ، ويمكننا عند الاستدعاء أن نكتب أي قيم من أي نوع، لكن ما يحدد نوع المتغيرات المرسلة هو نوع العملية وهي "x+y" ففي هذه الحالة سيحدث خطأ عند إرسال قيم حرفية نظراً لأن جملة الدالة تحتوي على جمع ، حيث لا يمكن جمع قيم نصية.
- هذه الطريقة تشبه إلى حد كبير تعريف الثوابت const حيث لا يمكن تغيير القيمة بعد تعريفها.
- يجب أن يتم وضع define قبل الدالة الرئيسية main "في منطقة التصاريح العامة".