Programming fundamentals 2
Chapter 4:Function

Miss: Hanan Hardan

Function

- **Function** is a block of instructions that is executed when it is called from some other point of the program.

The main goal of Function is

- to write error-free code
- to **reuse** any possible code that has already been written and tested (this is called **reusability**)

Function

Two types of functions:

1. Functions that return a value
2. Functions that do not return a value

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

Function Definition

a) Definition of a Function that does not return a value

1- Function without arguments:

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

where, ( ) is the empty list.

Function Definition

2- Function with arguments:

void function-name(argument-list)
{
  declarations and statements
}

where, *argument-list* is a list that contains one or more *argument*s that are passed to the function.

The Call to a function

- The call to a function that does not return a value is given in the **CALL** statement which has the following syntax:

  function-name (**Actual parameters**)

Examples:

```
draw_circle ( );
sum(4,7);
sum(x,y);
```

The Call to a function

Interpretation of the function call

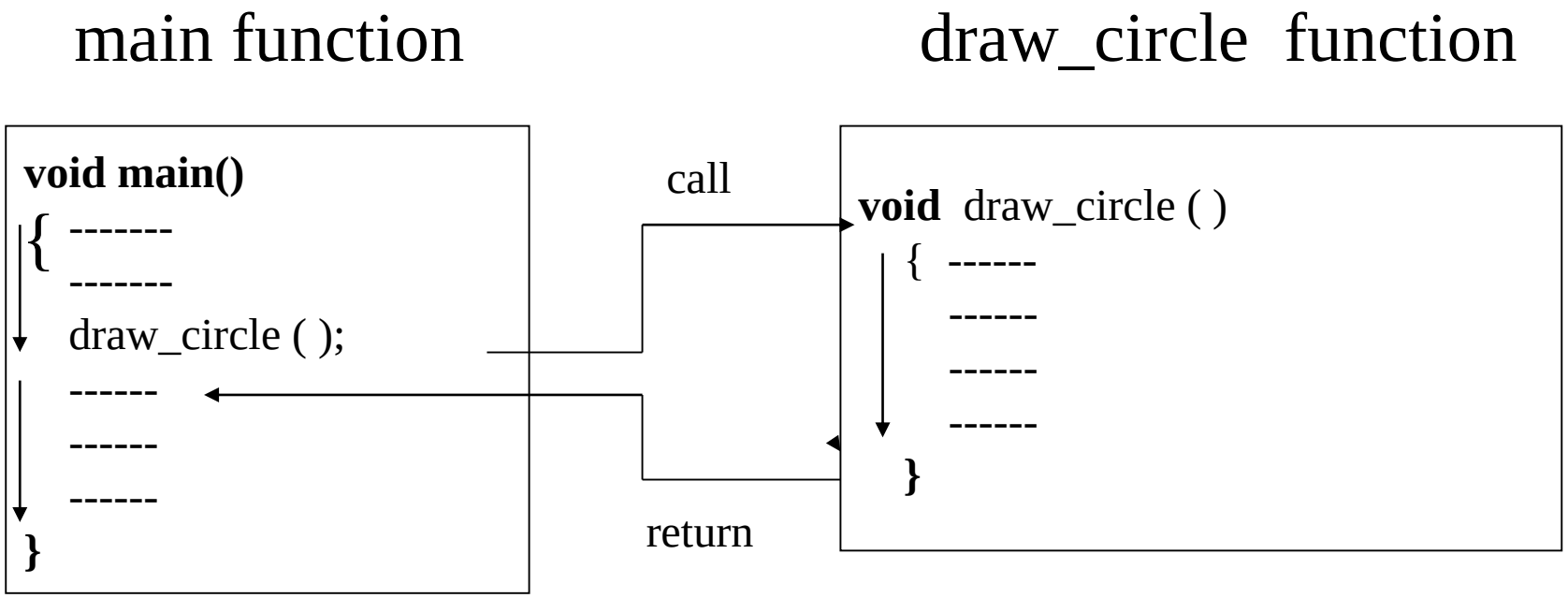
Suppose that main function *Figure* calls the function *draw_circle* using the call statement

draw_circle ( )

Then the flow of control between them will be as follows (see the next diagram):

- The flow of control in main function *Figure* stops on this statement to initiate the execution of the function draw_circle
- After this function has finished executing, the next statement in the calling main will be executed.

The Call to a Function



Example 1:

Syntax (in C++):

```
void print_box ()
    {
      cout << "* * * * * * *\n" ;
      cout << "*   " << 50 << " * \n" ;
      cout << "* * * * * * *\n" ;
          }

void main()
{
   print_box ();
   cout<<endl;
   print_box ();
}
```

Example 2:

Syntax (in C++):

```
void print_box (int   n)
        {
          cout << “* * * * * * *\n” ;
          cout << “*   “ << n << “ * \n” ;
          cout << “* * * * * * *\n” ;
         }
void main()
{
  int a;
  cout<<"\n Enter integer number";
  cin>>a;
  print_box (a);
  cout<<endl;
  print_box (5);
}
```

Function Definition

b) Definition of a function that returns a value:

1- Function without arguments:

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

- **Notes:**

- ***ftype***: is any data type that the result of a function can have.
- ( ) empty list.
- If the function has a type, then the *Statements* in the body of the function should have ***return*** statement, usually it is the last statement.

Function Definition

b) Definition of a sub-algorithm that returns a value:

2- Function without arguments:

ftype *function-name* (*argument-list*)

{

declarations and statements

}

- **Notes:**

- ***ftype***: is any data type that the result of a function can have.
- *argument-list :* includes one or more arguments.
- If the function has a type, then the *Statements* in the body of the function should have ***return*** statement, usually it is the last statement.

The Call to a function

- The call to a function that returns a value is given as follows:
  - The name of the function is given within the **output** statement
  - The name of the function is given within the **assignment** statement

Examples:

```
result = Sum (x, y );
cout<< Sum (x, y );
```

Example:

Syntax (in C++):

```
int sum ()
{
 int x=5, y=7 ;
return  x+y ;
}

void main()
{
   int a;
   cout<<“sum=“<<sum()<<endl;
   a=sum();
   cout<<“sum=“<<a<<endl;
}
```

Example:

Syntax (in C++):

```
int Rect_area (int L, int W)
{
  int a ;
   a = L * W ;
   return  a ;
}

void main()
{
 int a,b,z;
cout<<"\n Enter 2 integer number";
cin>>a>>b;
cout<<“area==“<<Rect_area(a,b)<<endl;
z=Rect_area(3,2);
cout<<“area=“<<z<<endl;
}
```

1) Write method that takes rectangle width and height as parameter (arguments) then returns the rectangle area

2) In main method use above method to print the area of a rectangle given width and height as input from user then print another area of a rectangle for width=2 and height =3

Example

```
#include <iostream>
using namespace std;
int Max (int Value1, int Value2) {
        if (Value1 > Value2)
                return Value1;
        else
                return Value2;
}

void main() {
        int a, b;
        cout<<"\nPlease Enter the values of a and b: ";
        cin>>a>>b;
        cout<<"\n the maximum one is: "<<Max(a,b)<<endl;
}
```

Function Definition

Calling the function in an expression like cout<<, condition, assignment statements

Example

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

int Max (int Value1, int Value2) {
        if (Value1 > Value2)
                return Value1;
        else
                return Value2;
}

void main() {
        int a, b;
        cout<<"\nPlease Enter the values of a and b: ";
        cin>>a>>b;
        cout<<"\n the maximum one is: "<<Max(a,b)<<endl;
}
```

1) Write method that takes two integers as parameter (arguments) then returns the maximum value

2) In main method use above method to print the maximum value of two integers given as input from user.

Example

1) Write method that takes two integers as parameter (arguments) then returns the summation value

2) In main method use above method to print the summation value of two integers given as input from user.

```
#include <iostream>
using namespace std;
int Sum (int A, int B) {
        return (A+B);
}

void main() {
        int N1, N2, S;
        cout<<"\n Please Enter N1 and N2: ";
        cin>>N1>>N2;
        S = Sum(N1,N2);
        cout<<"\nSum= "<<S<<endl;
}
```

Example

1) Write method that takes integer as parameter (argument) then returns the true if the value > 0 else return false.

2) In main method use above method to print "**the number is positive**" if the value of integer given as input from user is positive else print "**the number is negative**".

```
#include <iostream>
using namespace std;
bool Positive (int Num) {
        if (Num > 0)
                return true;
        else
                return false;
}
void main() {
        int Number;
        cout<<"\nEnter Number: ";
        cin>> Number;
        if (Positive(Number))
                cout<<"\n the number is positive";
        else
                cout<<"\n the number is negative";
        cout<<endl;
}
```

Example

1) Write method that takes circle radius as parameter (argument) then returns the circle area

2) In main method use above method to print the area of a circle given radius as input from user .

```
#include <iostream>
using namespace std;
float Area (int R) {
        return (3.14 * R * R );
}

void main() {
        int Radius;
        cout<<"Enter the Redius: ";
        cin>>Radius;
        cout<<"\nCircle Area is: "<<Area(Radius);
        cout<<endl;
}
```

Example

```
#include <iostream>
using namespace std;
long Power(int Base, int Exp) {
        int M=1;
        for(int i=1; i<=Exp; i++)
                M*=Base;
        return M;
}

void main() {
        int B, E;
        cout<<"\nEnter Base: ";
        cin>>B;
        cout<<"\nEnter Exponent: ";
        cin>>E;
        cout<<"\n Result= "<<Power(B,E);
        cout<<endl;
}
```

1) Write method that takes two integers as parameter (arguments) then returns the value of number1^number2

2) In main method use above method to print the value of number1^number2 For two integers given as input from user.

Example

1) Write method that takes integer as parameter (argument) then returns the factorial for that integer

2) In main method use above method to print the factorial of given integer input from user .

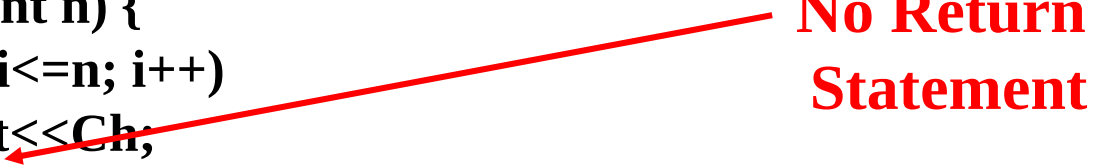
```
#include <iostream>
using namespace std;
long Fact (int Num) {
        int F = 1, i = Num;
        while (i>=1){
                F *= i;
                i--; }
        return F;
}

void main() {
        int Number;
        cout<<"Enter an integer number: ";
        cin>>Number;
        cout<<endl<<Number<<"!= "<<Fact(Number);
        cout<<endl;
}
```

Example

```
#include <iostream>
using namespace std;
void Print(char Ch, int n) {
        for (int i=1; i<=n; i++)
                cout<<Ch;
        cout<<endl;
}

void main() {
        char Sym;
        int Number;
        cout<<"\nEnter the Symbol: ";
        cin>>Sym;
        cout<<"\nHow many times: ";
        cin>>Number;
        Print(Sym,Number);
}
```



Example

```
#include <iostream>
using namespace std;
int Mul(int V1, int V2) {
        return V1 * V2; }

void Result() {
        cout<<"\n5 x 9 = "<<Mul(5,9);
        cout<<"\n4 x 7 = "<<Mul(4,7);
        cout<<"\n6 x 4 = "<<Mul(6,4)<<endl; }

void main() {
        Result() ;
}
```

The Function Prototype

- Like other identifiers in C++, function must be declared before it can be referenced.
- To declare a function, we can insert a ***function prototype*** before the ***main function***.

- The ***function prototype*** provides all information that the C++ compiler needs to know to translate calls to the function correctly.

- A function prototype tells the compiler the

  - ***data type*** of the function
  - the ***function name***
  - ***information*** about the arguments that the function expects.

- Examples:

```
void draw_circle ( );
int  m ( ) ;
void print_box (int) ;
int Rect_area (int , int);
```

Function Prototype

```
#include <iostream.h>
int Mul(int, int);
int Add(int, int);
void Show();

void main() {
        Show(); }

int Mul(int X, int Y) { return X*Y; }

int Add(int X, int Y) { return X+Y; }

void Show() {
        int A=10, B=20;
        cout<<Add(A,B)<<'\t'<<Mul(A,B)<<endl; }
```

Function Prototype
contains only data types
But may contain identifiers.

Scope of Variables

(1) Global variables:

- Those variables that are declared before the main function.
- These are visible from any point of the code, inside and outside any function.

(2) Local variables:

- These are declared inside a block or a function.
- The scope of local variables is limited to the same nesting level in which they are declared.

Visibility of Identifiers

- **Global Scope**

  Anything identified or declared outside of any function is visible to all functions in that file

- **Function level scope**

  Declaring variables inside a function can be used in the whole function

- **Block level scope**

  Variables or integers declared inside block are used inside block

Identifiers Important Points

- Do not create variables with same name inside blocks, inside functions or inside bigger blocks
- Try to use separate variable names to avoid confusion
- Reuse of variables is valid

Example of Local and Global Variables

```
// File: global.cpp
#include <iostream.h>
int   x = 7  ;                  // global variables
int fun1 (int );                // function prototype
void main ( )
{  int  z ;                     // local variable in main
    cout << "The global variable: "  << x ;
   z = fun1 ( 5 ) ;             // calling add function
  cout << " The result is " << z << endl ;
}

int fun1 ( int a )
{ int   r ;                     // local variable in fun1
   r =  a * a * a ;
   return  r ;  }
```

Global Variable

- Can be used anywhere in program
- Can cause logical problems if same variable name is used in local variable declarations

For good programming

- Try to minimize the use of global variables
- Try to use local variables as far as possible

Static variable

- Another class of local variable is the static type. It is specified by the keyword static in the variable declaration.
- The most striking difference from a non-static local variable is, a static variable is not destroyed on exit from the function

Example:

```
#include <iostream>
using namespace std;
int add()
{
   int  a=5;
    a++;
    return a;
}
void main()
{
    cout<<add();
    cout<<add();
    cout<<add();
}
```

666

```
#include <iostream>
using namespace std;
int add()
{
    int  static a=5;
    a++;
    return a;

}

void main()
{
    cout<<add();
    cout<<add();
    cout<<add();
}
```

678