Chapter-13

USER DEFINED FUNCTIONS

➢ Definition:
  • User-defined function is a function defined by the user to solve his/her problem.
  • Such a function can be called (or invoked) from anywhere and any number of times in the program.
  • The purpose of using a function is to make the program design process easy, understandable and thereby avoiding ambiguity.

➢ Advantages of user-defined functions:
  • If there is set of statements to be repeated several times in the program, these statements can be replaced as a function and called whenever and whenever required.
  • If a task in needed in more than one program or to a group of programmers, a function can be developed and made available to other programs or other programmers.
  • Programs can be modularized into functions. Each function can be developed independently, debugged and tested.
  • If a problem is divided into sub-problems, a team of programmers can be formed to solve each sub-problem, rather solving by a single person.
  • If functions are developed in a program, the number of line decreases.

➢ Function definition or structure of user-defined function:

- **Return type-specifiers**: is the data type of the value return by the function to another function when it is called. It can be char, int, float or void. Data type void is used when the function return no value to calling function.
- **Function-name**: is the name of the function. It is an identifier to identify the particular function in
a program.

- **Argument-list with declaration**: is the list of arguments or parameters or variables with their declaration. Each argument should be declared separately. Each declaration should be separated by comma. The list should be enclosed within parenthesis.

- The complete line is called the **function header**. Note that there is no semicolon at the end.

- **Local-variable declaration**: is the declaration of the variables that are used within the function. Since these variables are local to the function, these variables are called as local variables.

- **Executable statements**: are the statements that perform the necessary operations to solve the problem. If the function is returning a value, a return statement should be included. Otherwise return statement is not necessary.

- Local declaration and executable statements are together called as **body of the function**. The body of the function should be enclosed within the curled braces.

- **Sample Program: To find the greatest of three numbers.**

```cpp
#include<iostream.h>
#include<conio.h>
void main( )
{
    int biggest (int, int, int); //Function prototype
    int a, b, c, big;
    clrscr( );
    cout<<"Enter the three numbers";
    cin>>a>>b>>c;
    big = biggest(a, b, c); // Function call-statement
    cout<<"Biggest="<<big<<endl;
    getch( );
}
int biggest (int x, int y, int z)
{
    int big;
    big = x>y?(x>z?x:z):(y>z?y:z);
    return (big);
}
```

- **Calling a function:**

  - The definition of the program can come either before the main( ) function or after the main( ) function.
  - Calling function is a function that transfer control from it to another function by specifying the name of that function and passing arguments.
  - Called function is a function that receives the call and arguments from another calling function.
• Function call is the statement that is used to call or make another function execute.
• When a function call is made, the control jumps calling function to the called function.
• The general form of function call is:

<table>
<thead>
<tr>
<th>General Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable = function_name (argument list); OR variable = function_name( );</td>
<td>big = biggest(a, b, c);</td>
</tr>
</tbody>
</table>

➤ main ( ) function:
• The main ( ) function returns a value of type int to the operating system. If the program runs successfully, 0 is returned. Otherwise, a non-zero value is returned to the operating system, indicating that the program contains errors.
• If the main ( ) function is not returning a value, the data type void can be used as return type specifiers.
• The general form of main ( ) function is:

<table>
<thead>
<tr>
<th>int main ( )</th>
<th>void main ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ Executable-Statements; return 0; }</td>
<td>{ Executable-Statements; }</td>
</tr>
</tbody>
</table>

➤ Returning a value:
• When a function is called, the statements in the called function are executed.
• After executing the statements, the function returns value to the calling function. The return statement is used to return a value.
• A function can return only one value or none to the calling function, for every function call.
• The general form of return statements is:

| return (expression); OR return 0; |

➤ Function Prototype:
• A function prototype is a declaration of the function that tells the program about the type of the value return by the function and the number and type of arguments.
• The general form of function prototype is:

<table>
<thead>
<tr>
<th>General Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>return-typeSpecifier function-name ( type arg1, type arg2…); OR return-typeSpecifier function-name ( type, type …);</td>
<td>int biggest (int x, int y, int z); OR int biggest ( int, int , int);</td>
</tr>
</tbody>
</table>
Types of Arguments:
- A variable in a function header is called an argument. The arguments are used to pass information from the calling function to the called function.

Actual Arguments:
- The function call statement contains name of the function and the list of arguments to be passed. These arguments or parameters are called as actual arguments.
- The arguments can be constants, variables, or expressions.
- Example: big = biggest (a, b, c);

Formal Arguments:
- The function header contains return-type-specifier, function name and list of arguments with their declaration.
- These arguments are called as formal arguments or dummy arguments. Formal arguments get their values from the actual arguments.
- Example: int biggest (int x, int y, int z)

Local Variables:
- The variables declared inside function or block is said belong to that block. These variables are called as Local variables.
- Values of local variables are accessible only in that block.
- The function’s formal arguments are also considered as local variables.
- Example:
  ```c
  function-name (int a, int b)
  {
      int x, y;  //x and y are local variables;
      ...........
  }
  ```

Global Variables:
- The variables declared outside the function are called as global variables. These variables are referred by the same data type and same name throughout the program in both calling function and called function.
- Whenever if some variables are to treated as same value in both main( ) and in other functions, it is advisable to use global variables.
- Example:
```c
int a, b; // a and b are global variables
void main( )
{
    int p, q; // p and q are local variables
    ..........}
```

- **Program: To find the LCM of two numbers:**

```c
#include<iostream.h>
#include<conio.h>
int a,b; // a and b are global variables
int gcd(int x, int y)
{
    int r; // r is local variable
    while(b!=0)
    { 
        r = a%b;
        a=b;
        b=r;
    }
    return a;
}
void main( )
{
    int gcd(int x, int y) //Function Prototype
    int product, lcm, g;
    clrscr( );
    cout<<"enter two numbers:";
    cin>>a>>b;
    product= a *b;
    g = gcd(a, b);
    lcm = product/g;
    cout<<"GCD="<<g<<endl;
    cout<<"LCM="<<lcm<<endl;
    getch();
}
```

- **Scope of Variables:**
- Scope of variables refers to the part of the program where the value of the variable can be used.
- The scope of the variables begins from where the variable declared.
- If the declaration is inside a function, the scope extends to the end of the innermost block that contains the declaration.
- Example: To illustrate scope of variables
  ```c
  #include<iostream.h>
  ```
```c
#include<conio.h>
void f( );
void g( );
int x=10;
void main( )
{
    int x = 20;
    {
        int x = 30;
        cout<<"In block inside main(): x="<<x<<endl;
    }
    cout<<"In main(): x="<<x<<endl;
    cout<<"In main(): ::x="<<::x<<endl;
f( );
g( );
}
void f()
{
    cout<<"In f(): x="<<x<<endl;
}
void g()
{
    int x=40;
    cout<<"In g(): x="<<x<<endl;
}

Types of Function:
- There are different types of functions:
  a. Function with no arguments and no return values.
  b. Function with arguments and with no return values.
  c. Function with no arguments and with return values.
  d. Function with arguments and with return values.
  e. Recursive function.

✓ Function with no arguments and no return values:
- In this method, the simply performs independent task. The function does not receive or send any arguments.
```

<table>
<thead>
<tr>
<th>General Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>void function-name( )</td>
<td>void natural( )</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>Statements;</td>
<td>for(int i=1; i&lt;=10; i++)</td>
</tr>
<tr>
<td>}</td>
<td>cout&lt;&lt;&quot;\t&quot;&lt;&lt;i;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
Function with arguments and with no return values:

- In this method, the function receives some arguments and does not return any value.

<table>
<thead>
<tr>
<th>General Form</th>
<th>Example</th>
</tr>
</thead>
</table>
| void function-name(argument list)  
  {  
    Statements;  
  } | void sum(int x, int y, int z)  
  {  
    int sum;  
    sum = x + y + z;  
    cout << "Sum = " << sum;  
  } |

Function with no arguments and with return values:

- In this method, the function receives no arguments but returns a value.

<table>
<thead>
<tr>
<th>General Form</th>
<th>Example</th>
</tr>
</thead>
</table>
| return-type function-name()  
  {  
    Statements;  
    return(value)  
  } | int greatest()  
  {  
    if(a > b)  
      return(a);  
    else  
      return(b);  
  } |

Function with arguments and with return values:

- In this method, the function receives some arguments and returns a value.

<table>
<thead>
<tr>
<th>General Form</th>
<th>Example</th>
</tr>
</thead>
</table>
| return-type function-name(argument-list)  
  {  
    Statements;  
    return(value)  
  } | float interest(float p, float t, float r)  
  {  
    si = (p * t * r)/100;  
    return(si);  
  } |

Recursive function:

- Recursive function is a function that calls itself. The process of calling function by itself is called as recursion.

<table>
<thead>
<tr>
<th>General Form</th>
<th>Example</th>
</tr>
</thead>
</table>
| return-type function-name(argument-list)  
  {  
    Statements;  
    return(function-name(argument-list))  
  } |

Passing default arguments to functions:

- In C++, to call a function we need not pass all the values for the arguments to a function from the calling function.
Chapter 13- User Defined Functions

- It allows us to assign default values to the formal arguments.
- Default values can be assigned to the argument which does not have a matching argument int the function call.
- Default values are assigned in the function prototype.
- The default values are assigned to the arguments as we initialize variables.
- Example:
  
  ```
  float interest(float amt, int time, float rate = 0.15) 
  si = interest(5000, 5)  //third argument is missing 
  si = interest(5000, 5, 0.2)  //No missing arguments 
  ```

➤ Uses of default values:

- Default arguments are useful in situations where some arguments always have the same value.
- Default arguments provide flexibility to the programmer.
- Default arguments are used to add new arguments to the existing functions.
- Default arguments are used to combine similar functions into one.

➤ Passing Constant arguments:

- In C++, we can declare some arguments as constants. The compiler cannot modify the arguments marked as constants.
- Example:
  ```
  int strlen(const char *p);
  int total(const int x, const int y);
  ```

➤ Pass by value or Call by Value:

- A function can be called by passing arguments from the calling function into the called function.
- Thus data is transferred through argument list.
- The calling function sends the data to the called function through actual parameters.
- The called function receives the data into its corresponding formal parameters.
- In pass by value, a copy of data sent by calling function stored in temporary locations,
- The called function uses these values as the initial values of the formal parameters.
- The formal parameters are processed or changed to generate the required result.
- However these changes are affected to the original values stored in the actual parameters.
- Example:
  ```
  void main() 
  { 
  ```
void swap(int, int);  //Function prototype
{
    int a, b;
    clrscr();
    cout<<"Enter two numbers";
    cin>>a>>b;
    cout<<"Before calling the function: a=\"<<a<<"\" and b=\"<<b<<endl;
    swap(a, b);  //Actual Arguments
    cout<<"After calling the function: a=\"<<a<<"\" and b=\"<<b<<endl;
    getch();
}

void swap(int x, int y)  //Formal Arguments
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}

➢ Pass by reference or call by reference:

- Sometimes we need to change the values of the actual arguments in the calling function. This is not possible in the passing by value method.
- C++ allows us to pass arguments to the function by using reference variables.
- When we pass arguments by reference, the formal arguments int the called function become the aliases to the actual arguments of the calling function i.e. the called function is actually uses the original data with a different name.
- Example:

```c++
void main( )
{
    void swap(int &, int &);  //Function prototype
    
    int a, b;
    clrscr();
    cout<<"Enter two numbers";
    cin>>a>>b;

    Output:
    Enter two numbers 10 20
    Before calling function: a=10 and b=20
    After calling function: a=10 and b=20
```
cout<<"Before calling the function: a="<<a<<""and b="<<b<<endl;
swap(a, b);        //Actual Arguments
cout<<"After calling the function: a="<<a<<""and b="<<b<<endl;
getch();
}

void swap(int &x, int &y)    //Formal Arguments
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}

---

**Output:**
Enter two numbers 10 20
Before calling function: a=10 and b=20
After calling function: a=20 and b=10

---

---

**CHAPTER 13 – USER DEFINED FUNCTIONS BLUE PRINT**

<table>
<thead>
<tr>
<th></th>
<th>VSA (1 marks)</th>
<th>SA (2 marks)</th>
<th>LA (3 Marks)</th>
<th>Essay (5 Marks)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 Question</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>01 Question</td>
<td>06 Marks</td>
</tr>
</tbody>
</table>

***************