Functions & Pointers

Diving inside a Program

Function

- A function is a self-contained block of statements that perform a logical task of some kind.
- Every C program can be thought of as a collection of these functions.
- Using a function is something like hiring a person to do a specific job for you.
- Sometimes the interaction with this person is very simple; sometimes it’s complex.

Operation of C Function

We will be looking at two things—a function that calls or activates the function and the function itself.

```
main( )
{
    message( ) ;
    printf ( "\nCry, and you stop
    the monotony!" );
}
message( )
{
    printf ( "\nSmile, and the
    world smiles with you..." );
}
```

And here’s the output...

```
Smile, and the world smiles
with you...
Cry, and you stop the
monotony!
```

Operation of C Function

```
main( )
{
   printf ("\nI am in main");
   italy( );
   brazil( );
   argentina( );
}
italy( )
{
   printf ("\nI am in italy");
}
brazil( )
{
   printf ("\nI am in brazil");
```

Properties of Functions:

- Any C program contains at least one function.
- If a program contains only one function, it must be main( ).
- If a C program contains more than one function, then one (and only one) of these functions must be main( ), because program execution always begins with main( ).

Properties of Functions:

- There is no limit on the number of functions that might be present in a C program.
- Each function in a program is called in the sequence specified by the function calls in main( ).
- After each function has done its thing, control returns to main( ). When main( ) runs out of function calls, the program ends.

Function Calls

```
main( )
{
    printf ( "\nI am in main" ) ;
    italy( ) ;
    printf ( "\nI am finally back in main" ) ;
}
italy( )
{
    printf ( "\nI am in italy" ) ;
    brazil( ) ;
    printf ( "\nI am back in italy" ) ;
}
brazil( )
{
    printf ( "\nI am in brazil" ) ;
    argentina( ) ;
}
```

Summarizing Some Facts

- C program is a collection of one or more functions.
- A function gets called when the function name is followed by a semicolon. For example,

```
main( )
{
    argentina( ) ;
}
```

Summarizing Some Facts

- A function is defined when the function name is followed by a pair of braces in which one or more statements may be present. For example,

```
argentina( )
{
   statement 1 ;
   statement 2 ;
   statement 3 ;
}
```

Summarizing Some Facts

- Any function can be called from any other function. Even main( ) can be called from other functions. For example,

```
main( )
{
   message( ) ;
}
message( )
{
   printf ( "\nCan't imagine life without C" ) ;
   main( ) ;
}
```

Summarizing Some Facts

- A function can be called any number of times. For example,

```
main( )
{
   message( ) ;
   message( ) ;
}
message( )
{
   printf ( "\nJewel Thief!!" ) ;
}
```

Summarizing Some Facts

- The order in which the functions are defined in a program and the order in which they get called need not necessarily be the same. For example,

```
main( )
{
    message1( ) ;
    message2( ) ;
}
message2( )
{
    printf ( "\nBut the butter was bitter" ) ;
}
message1( )
{
    printf ( "\nMary bought some butter" ) ;
}
```

Summarizing Some Facts

- A function can call itself. Such a process is called ‘recursion’.
- A function can be called from another function, but a function cannot be defined in another function. Thus, the following program code would be wrong, since argentina( ) is being defined inside another function, main( ).

```
main( )
{
    printf ( "\nI am in main" ) ;
    argentina( )
    {
        printf ( "\nI am in argentina" ) ;
    }
}
```

Summarizing Some Facts

- There are basically two types of functions:
    - Library functions Ex. printf( ), scanf( ) etc.
    - User-defined functions Ex. argentina( ), brazil( ) etc.
- As the name suggests, library functions are nothing but commonly required functions grouped together and stored in what is called a Library.
- This library of functions is present on the disk and is written for us by people who write compilers for us.
- Almost always a compiler comes with a library of standard functions.
- The procedure of calling both types of functions is exactly same.

Why to Use Functions?

Why write separate functions at all? Why not squeeze the entire logic into one function, main( )? Two reasons:

- Writing functions avoid rewriting the same code over and over.
- Using functions it becomes easier to write programs and keep track of what they are doing.
  - If the operation of a program can be divided into separate activities, and each activity is placed in a different function, then each could be written and checked more or less independently.
  - Separating the code into modular functions also makes the program easier to design and understand.

Definition of Functions

- A function definition, also known as function implementation shall include the following elements:
  1. Function name;
  2. Function type;
  3. List of parameters;
  4. Local variables declaration;
  5. Function statements;
  6. A return statement

Definition of Functions

- All the six elements are grouped into two parts, namely,
  - Function header (first three elements);
  - Function body (second three elements).
- General Format:

```
function_type function_name(parameter list)
{
    local variable declaration;
    executable statements…
    return statement;
}
```

Definition of Functions

- Function Type
  - The function type specifies the type of value (like float or double) that the function is expected to return to the program calling the function.
  - If the return type is not explicitly specified, C will assume it is an integer type.
  - If the function is not returning anything, then we need to specify the return type as **void**.
  - **void** is one of the fundamental datatypes in C.
  - It is a good programming practice to code explicitly the return type, even when it is an integer.
  - Value returned is the output produced by the function.

Formal Parameter List

- The parameter list declares the variables that will receive the data sent by the calling program.
- They serve as input data to the function to carry out the specific task, since they represent actual input values, they are often referred to as formal parameter values.
    - float quadratic(int a, int b, int c)

```
{
    …
}
```

Return Value & Their Types

- A function may or may not send back any value to the calling function.
- If it does, it is done through the return statement.
- While it is possible to pass to the called function, any number of values, the called function can only return one value per call, at the most.

Function Declaration

- Like variables, all functions in a C program must be declared, before they are invoked. A function declaration (also known as function prototype consists of four parts:
  - Function type(return type)
  - Function name
  - Parameter list
  - Terminating semicolon.

They are coded in the following format:

Function type function name(Parameter list);

Points to Note

1. The parameter list must be separated by commas.
2. The parameter names do not need to be the same in the prototype declaration and the function definition.
3. The types must match the type of parameters in the function definition, in number and order.
4. Use of parameter names in the declaration is optional.
5. If the function has no formal parameters, the list is written as (void).
6. The return type is optional when the function returns int type data.
7. The return type must be void if no value is returned.
8. When the declared types do not match the types in the function definition, the compiler will produce an error.

Prototype Declaration

- A prototype declaration may be placed in two places in the program.
  - Above all the functions (including **main**): When we place the declaration above all the functions, the prototype is referred to as a global prototype.
  - Inside a function definition: When we place it in a function definition, the prototype is called a local prototype.

Prototype

- Prototype declarations are not essential.
- If a function has not been declared before it is used, C will assume that its details are available at the time of linking.
- Since the prototype is not available, C will assume that the return type is an integer and that the types of parameters match the formal definitions.
- If these assumptions are wrong, the linker will fail and we will have to change the program.
- The moral is that we must always include a prototype declaration in the global declaration section.

Command Line Arguments

THANK YOU