CHAPTER

13

C Scope Rules

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable cannot be accessed. There are three places where variables can be declared in C programming language:

- Inside a function or a block which is called **local** variables,
- Outside of all functions which is called **global** variables.
- In the definition of function parameters which is called **formal** parameters.

Let us explain what are **local** and **global** variables and **formal** parameters.

Local Variables

Variables that are declared inside a function or block are called **local variables**. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using **local variables**. Here all the variables a, b and c are local to **main()** function.

```
#include <stdio.h>

int main ()
{
  /* local variable declaration */
  int a, b;
  int c;

  /* actual initialization */
  a = 10;
  b = 20;
  c = a + b;

  printf ("value of a = %d, b = %d and c = %d\n", a, b, c);

  return 0;
}
```

Global Variables

Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A **global variable** can be accessed by any function. That is, a **global variable** is available for use throughout your entire program after its declaration. Following is the example using global and local variables:

```
#include <stdio.h>

/* global variable declaration */
int g;

int main ()
{
  /* local variable declaration */
  int a, b;

  /* actual initialization */
  a = 10;
  b = 20;
  g = a + b;

  printf ("value of a = %d, b = %d and g = %d\n", a, b, g);

  return 0;
}
```

A program can have same name for local and global variables but value of local variable inside a function will take preference. Following is an example:

```
#include <stdio.h>

/* global variable declaration */
int g = 20;

int main ()
{
  /* local variable declaration */
  int g = 10;

  printf ("value of g = %d\n",  g);

  return 0;
}
```

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

```
value of g = 10
```

Formal Parameters

Function parameters, so called **formal parameters**, are treated as local variables within that function and they will take preference over the global variables. Following is an example:

```
#include <stdio.h>

/* global variable declaration */
int a = 20;

int main ()
{
  /* local variable declaration in main function */
  int a = 10;
  int b = 20;
  int c = 0;

  printf ("value of a in main() = %d\n",  a);
  c = sum( a, b);
  printf ("value of c in main() = %d\n",  c);

  return 0;
}

/* function to add two integers */
int sum(int a, int b)
{
    printf ("value of a in sum() = %d\n",  a);
    printf ("value of b in sum() = %d\n",  b);

    return a + b;
}
```

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

```
value of a in main() = 10

value of a in sum() = 10

value of b in sum() = 20

value of c in main() = 30
```

Initializing Local and Global Variables

When a local variable is defined, it is not initialized by the system, you must initialize it yourself. Global variables are initialized automatically by the system when you define them as follows:

Data Type	Initial Default Value
int	0

char	'\0'
float	0
double	0
pointer	NULL

It is a good programming practice to initialize variables properly otherwise, your program may produce unexpected results because uninitialized variables will take some garbage value already available at its memory location.