Functions

Unit 7

INTRODUCTION

In the previous unit 6 we have learned important topics such as different types of arrays and strings and their declarations, initializations, structures and applications. We think that, we have able to usages of these topics when we will work on our programming fields. In this unit we will describe about another important topic in C language like functions. Actually we have used functions earlier in every program that we have discussed so far. We have already seen some functions like *printf(), scanf(), gets(), puts(), main(), sqrt(), strcat(), strcmp(), strlen()* etc. In this unit, we will describe in detail how a function is designed, mentioned, how two or more functions are situate together and how functions are communicated with one another.

Timeframe How long ?	We expect that this unit will take maximum 15 hours to complete.

Unit Structure

Lesson-1 Introduction to C functions

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Understand basic idea about C functions and its types.
- Learn necessity of functions in program.

✓	**Keywords**	**Function, Necessity of Function, Types**

FUNCTION IN C

The C programming language is related to most contemporary programming languages in that it agrees to the use of functions, self-controlled modules of code that receive inputs, perform computation, and generate outputs. A function is defined as ***"A function is a block of code that performs a particular task".*** Every C program has at least one function, which is ***main()***, and all the most trivial programs can define supplementary functions. We can break up or divide up our code into separate functions. Actually, how we divide up our code among dissimilar functions is up to our concept, but logically the partition is such that each function performs a definite task. C functions are fundamental building blocks in a program. Generally C programs are written using functions to improve re-usability, understandability and to keep track on them. Elaborately we also say that "A function is a section of code that takes information does some calculation, and returns a new piece of information based on the parameter information. C language provides an approach in which you need to declare and define a group of statements once and that can be called and used whenever required.

TYPES OF FUNCTION

In C, functions can be classified into two types such as:

- Library functions
- User defined functions

Library functions

Library functions are those functions which are defined by C library. For example *printf(), scanf(), strcat(), strlen(), gets(), puts(), sqrt()* etc. If we want to use library function we just need to include appropriate header files to use these functions. These are already declared and defined in C libraries.

User defined functions

C allows programmer or user to define their individual function according to their prerequisite. These types of functions are known as user-defined functions. User defined functions are those functions which are defined by or developed by the user at the time of writing program. The most important difference between library functions and user defined functions is that library functions are not required to be written by programmer or user, on the other hand a user defined function has to be defined or developed by the programmer or user at the time of writing a program. Actually we will describe about the user defined functions in this unit.

Note it!

A function is a block of code that performs a particular task. Every C program has at least one function, which is main(), and all the most trivial programs can define supplementary functions.

NEED FOR USER DEFINED FUNCTION

We mentioned earlier, a function is a block of code that performs a particular task. We can either use the built-in library functions or we can create our own defined functions. In every C program must have a main function to designate where the program has to start its execution. While it is probable to code any program utilizing only main function, it leads to a number of problems.

In practical, if the program may become too large and difficult then the task of debugging, testing and maintaining becomes difficult. In this case, if a program is divided into more than one functional part, then each part may be separately coded and later combined into a single unit. For removing these complexity of a program we need user define function. Actually user defined functions are extremely necessary for complex programs

Necessity of user defined function as follows:

1. User defined function provides modularity to the program.
2. The piece of a source program can be reduced by using user defined functions at proper places.
3. It is easy to locate and segregate a defective function for further exploration.
4. User defined functions helps to decompose the large program into small segments which makes programmer easy to understanding, maintaining, debugging and testing.
5. It is easy to code reusability. We just have to call the function by its name to use it from anywhere of a program.
6. A user defined function may be used by many other programs.

Activity

1. Mention the significance of using function in C language by your own idea.
   …..………………………………………………………………………….……
   ……………………………………………………………………………………

Summary Summary	
In this lesson we have ▪ Learned about function definition and declaration. ▪ Learned about the user defined and libraray functions. ▪ Also understood how a function is called in a program .	

Assessment

Assessment

Write 'T' for true and 'F' for false for the followings sentences

1. printf(), scanf() are the library function.

2. A function is a block of code that performs a specific task.
3. User defined function provides modularity to the program.
4. A user defined function may be used by only specific programs.

Multiple Choice Questions (MCQ)

1. Every C program has at least one function, which is—
   a) include
   b) define
   c) main()
   d) return
2. A function is--
   a) a block of code that performs a particular task
   b) set of statements
   c) a block of code that performs a verities task
   d) set of modules
3. sqrt() function is a
   a) user defined function
   b) library function
   c) main function
4. User defined functions helps to decompose the large program-
   a) Into small segments
   b) Into large segments
   c) Into variable segments
   d) None of these

Exercises

1. What is function? Explain user define function definition and declaration procedure.
2. Explain the necessity of user defined function in a program.

Lesson-2 Function Prototype and Definition

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Understand function prototype.
- Explain function definition or declaration.
- Understand how a function is declared in program.
- Explain function calling procedure.

Keywords **Function, Declaration, Prototype, Calling function**

FUNCTION PROTOTYPE OR DECLARATION

User defined functions that a programmer writes will generally require a prototype. In programming, a function prototype is a declaration of a function that specifies the function's name and type signature or arguments and return type, but omits the function body. The function prototype gives basic structural information such as (i) It tells the compiler what the function will return, (ii) What the function will be called, and (iii) what arguments the function can be passed. Function prototype is declared before it is defined in a program. Generally it is typed before main function of a program.

The general format for a function prototype or declaration is:

return-type function_name (data_type arg_1,data_type arg_2 ..., data_type arg_N);

Here, *data_type* means the data type of arguments or parameters such as *int, float, double, char* etc**.** and *arg_1, argu_2… arg_N* means that how many arguments or parameters are defined in the parameter list.
There can be more than one argument passed to a function or none at all, and also a function does not have to return a value. Functions that do not return values have a return type of void. Let's look at a function prototype or declaration

int addition(int num1, int num2);

In this declaration or prototype example specifies that the function ***addition*** will accept two arguments, both are integers, and function will return an integer type value. Here, do not forget the trailing semi-colon (;). Without semicolon the compiler will possibly think that you are trying to write the actual definition of the function.

Note it!

Function prototype is declared before it is defined in a program. Generally it is typed before main function of a program.

FUNCTION DEFINITION SYNTAX

The general form of function definition syntax in C programming language is as follows:

return-type function_name(parameter list)
{
function-body
}

In the function definitions, the first line ***return-type function-name(parameter list)*** is known as **function header** and the statement within curly braces { } is called **function body.** Function body is combination of more than one statement.
We will describe all the parts of a function as follows:

1. **return-type**: A function may return a value. The **return-type** is the data type (int, float, char, double) of the value the function returns. If a function performs the desired operations without returning a value, then the return_type is the keyword **void**.
2. **function-name**: function name specifies the actual name of a function. The function name is any valid C identifier and therefore must follow the same rule of formation as other variables in C.
3. **parameter list:** The parameter list declares the variables that will receive the data sent by calling program. A parameter is like a placeholder. When a function is invoked, we pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters. These parameters are also used to send values to calling program.
4. **function-body**: The function body contains a collection of statements and declaration of variables that define what the function does. The body is enclosed within curly braces { } and consists of three parts.

- ❖ Local variable declaration.
- ❖ Function statement that performs the tasks of the function.
- ❖ A return statement that return the value evaluated by the function.

Example:

Given below is the source code for a function called **summation()**. This function takes two parameters value1 and value2 and returns the summation of two values.

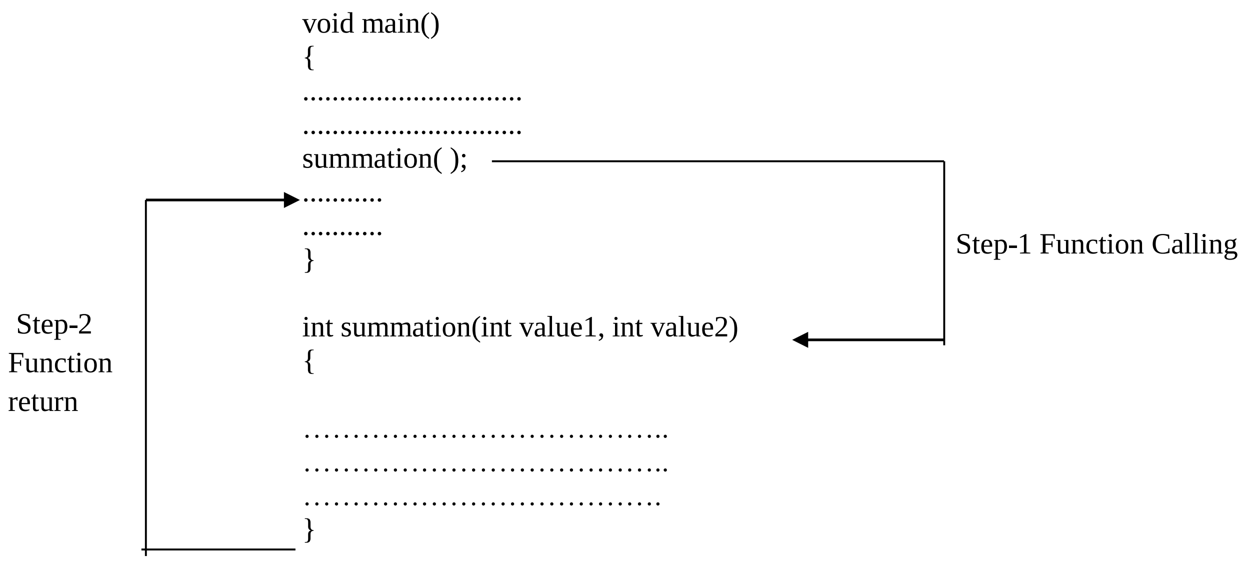
```
/* function returning the summation of two values */
            int summation(int value1, int value2)
            {
              int result;            /* local variable declaration */
             result= value1+value2
             return result;
            }
```

HOW USER-DEFINED FUNCTION WORKS IN C PROGRAMMING?

We mentioned earlier, that every C program starts from *main()* function and program starts executing the codes inside main() function. When the control of program reaches to function name inside *main()* function the control of program jumps to function definition part and executes the codes inside it. When all the codes inside that user-defined function are executed, control of the program jumps to the statement just after function name from where it is called.

Considering the following example:

```
#include <stdio.h>
int summation(int value1, int value2);     <———— Function prototype
```



```
void main()
{
.............................
.............................
summation( );
...........
...........
}

int summation(int value1, int value2)
{

………………………………..
………………………………..
……………………………….
}
```

In this above example, when function *summation()* is called from *main()* function the control jumps to function definition part and executes all statements of *summation()* function and after executing all statements it returns to statement just after function name *summation()* from where it is called.

FUNCTION CALLING

In C program, whereas creating a function, we give function definition of what the function has to do. To use a function, we will have to call that function to perform the defined task. A function can be called by using the function name in a statement of program.

Example:
Given below is example of a function calling process:

```
void main()
{
     int mulvalue;
    mulvalue = multiplication(20,5);
    printf("%d", mulvalue);
}
```

When a program calls a function, the program control is transferred to the called function. In the above example, when the compiler encounters a function (*multiplication(20,5))* call, the control is transferred to the function ***multiplication(x,y)*** which is already defined in program. This function is then executed and returns a value when a return statement is encountered and then the return value is stored or assigned to ***mulvalue*** variable.

Note it!

When a program calls a function, the program control is transferred to the called function.

Program 7.2.1 Write a program to calculate the value $z = x^y$ using function

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void powerval( );
void main( )
{
    powerval( );
    getch( );
}
void powerval( )
{
      long int x,y,z;
      printf("Enter the value of base  (x):");
      scanf("%ld",&x);
      printf("\nEnter the value of power (y):");
      scanf("%ld",&y);
     z= pow(x,y);
     printf(" \nThe power value is: %ld ",z);
}
………………………………………………………………………………………
```

Output

```
Enter the value of base  (x): 5
Enter the value of power (y):2
The power value is: 25
```

Program 7.2.2 Write a program to calculate the summation of 5 integer numbers using function

```
#include<stdio.h>
#include<conio.h>
void summation( );
void main( )
{
       summation( );
       getch( );
}
void summation( )
{
       int i,sum=0;
      int summ[10];
      printf("Enter 5 numbers:\n ");
      for(i=0;i<5;i++)
       scanf("%d",&summ[i]);
      for(i=0;i<5;i++)
       {
             sum=sum+summ[i];
       }
         printf(" \nThe Summation of entered values is:= %d ",sum);
}
……………………………………………………………………………………….
```

Output

```
Enter 5 numbers:
10
30
12
11
45
The Summation of entered values is:= 108
```

Activity

1. Mention the significance of using function in C language by your own idea.
………………..………………………………………………………………….…….
…………………………………………….……………………………………….
2. Create a function named as "average" to find out average value of 10 numbers with function definition.
………………..……………………….…………………………………..
……………………………………….…………………………………….
3. Write a program to determine the greatest common divisor(GCD) and least common multiple (LCM) of two integer number using function
……………………………………………….……………………………….
……………………………………………….……………………………….

Study skills

1. Suppose we want to determine summation in a range of 10 to 100 numbers using function. Which is the correct declaration?

   A. float number[10]; B. int summation[10,100]; C. double summation(1,100);
   D. int summation (10,100);

Solution:

According to function declaration "D" is the correct function declaration.

2. Which is the correct answer?

   a. Mul(x,y)=p; b. Mul[10,2]=x; c. X= mul[x,y]; d. X=mul(x,y);

Solution:

According to function declaration "D" is the correct function declaration.

3. Which are the incorrect function declarations?

   A. Mul(x,y)=p; B. Int record(2,x); C. Float calculation(p,Q); D. P= mul(x,y);

Solution:

According to function definition and declaration "A", "B", and "C" are the incorrect function declaration.

Summary Summary	
In this lesson we have ▪ Learned about functions prototypes and definition. ▪ Also understood how a function is called in a program. ▪ Understand that how user defined functions are used in program.	

ASSIGNMENT

Assignment

1. Write a program to determine the greatest common divisor (GCD) and least common multiple (LCM) of two integer number using function.
   ……………………………………………………………………………………..
   ……………………….……..……………………………………………………….
2. Write a program using function, that reads *n* numbers from the user and determine the maximum and minimum number from *n* numbers
   ……………………….……..……………………………………………………..
   ……………………….……..……………………………………………………..

Assessment

Assessment

Multiple Choice Questions (MCQ)

1. Function prototype is
   a) declared before it is defined in a program
   b) declared after it is defined in a program
   c) declared any place in a program
   d) declared after main() function
2. A function body consists of
   a) Four parts
   b) Two parts
   c) Three parts
   d) None of these

Exercises

1. What is function? Explain user define function definition and declaration procedure?
2. Write a program to calculate factorial value of a number using function.
3. Write a program using function, that reads *n* numbers from the user and determine the maximum and minimum number from *n* numbers.

Lesson-3 Categories of User Defined Functions

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Classify different types of functions.
- Understanding difference between calling and called functions.

	Keywords	**Categories, Calling function, Called function**

CATEGORY OF FUNCTIONS

We have already known that, when we write large difficult programs, it becomes difficult to maintain track of the source code of a program. The work of functions is to divide the large program to many separate modules based on their functionality. So a function depending on whether arguments are present or not and whether a value is returned or not, may belong to one of the following categories:

1. Function with no arguments and no return value
2. Function with no arguments and return value
3. Function with arguments and no return value
4. Functions with arguments and return value

In the next lessons we will describe these categories with example.

Summary

Summary

In this lesson we have

- Learned about categories of functions and its definitions

Assessment

Assessment

1. Write down the function categories
   …………………………………………………………………………………
   …………………………………………………………………………………

Exercises

1. Explain function categories with their definitions?
2. Explain the necessity of user defined function in a program.

Lesson-4 Function With No Arguments and No Return Value

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Explain function with no arguments and no return value technique.

	Keywords	**Function, Arguments, Return, Value**

FUNCTION WITH NO ARGUMENTS AND NO RETURN VALUE

When a function has no arguments, it does not receive data from the calling function as well as when it does not return value, the calling function does not receive data from the called function. If a function does not return value, we may be used keyword ***void*** as return type. The general syntax of this type of function is as follows:

```
void function_name( )
{
……………………………
Statements or function body
…………………………..
}
```

EXAMPLE

Program 7.4.1 Write a program to check whether a number entered by user is prime or not using function with no arguments and no return value.

```
#include <stdio.h>
#include<conio.h>
void prime_number( );
void main( )
{
   clrscr( );
    prime_number( );      /*No argument is passed to prime_number ()*/
    getch();
}
void prime_number ( )
{
    int number,i,flag=0;
    printf("Enter positive number to check:\n");
    scanf("%d",& number);
    for(i=2;i<= number /2;++i)
      {
          if(number % i==0)
              {
                flag=1;
              }
```

```
     }
    if (flag==1)
       printf("%d is not Prime Number", number);
   else
      printf("%d is Prime Number", number);
  }
…………………………………………………………………………………………
```

Output:

```
Enter positive number to check:
5
5 is Prime Number
```

Note it!

If a function does not return value, we may be used keyword void as return type.

Activity

1. Write down the benefits of function with no arguments and no return value by your own idea.
   ……..……….……………………………………………..……………………………
   …………...……………………………………………..……………………………
   …………..……………………………………………..……………………………
2. Create a function named as “power” to determine power of x^y using function with arguments and no return value concept.
   …………..……………………………..…..……………………………………..
   …..……...……………………………...……………………………………….

Study skills

1. Determine the output of the following program:

```
void main()
{
 product( );
 }
void product()
{
     int x = 5,p;
     int y = 20;
     p = x*y;
     printf(“Output is:=%d”,p);
}
```

Summary Summary	
In this lesson we have ▪ Learned about categories of functions with no arguments and no return value concept and its declaration and definition. ▪ Also understood how a function is called in a program.	

ASSIGNMENT

Assignment

1. Write a program to evaluate f(x)= $x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+ - - - - - - -$ using function with no arguments and no return value concept.
   ……….…...………………………………………………………………………..
   ..…………..…………………………………………………………………..

Assessment

Assessment

Multiple Choice Questions (MCQ)

1. When a function has no arguments it does not-
   a) receive data from the calling function
   b) receive data from the called function
   c) return data to calling function
   d) none of these
2. If a function does not return value, we may be-
   a) Used keyword ***float*** as return type
   b) Used keyword *void* as return type
   c) Used keyword ***int*** as return type
   d) None of these

Exercises

1. Explain function categories with their definitions?
2. Write a program using function with no arguments and no return value procedure, that calculates the value of Y in following series:
   $Y=1^2+3^2+5^2+$……………………………………..$+N^2$
3. Write a program using function with no arguments and no return value procedure, that calculates the value of Y in following series:
   $Y=1^2+3^4+5^6+$……………………………………..$+N^n$

Lesson-5 Function With No Arguments and Return Value

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Explain the function with no arguments and return value concept.

	Keywords	Function, No arguments, Return Value

FUNCTION WITH NO ARGUMENTS AND RETURN VALUE

When a function has no arguments but it may return a value so, we may be used return type. The general syntax of this type of function is as follows:

```
return_type function_name( )
  {
        ……………………………
        Statements or function body
        …………………………..
        …………………………..
        return variable-name/value
  }
```

Here, ***return_type*** is data type name which is returned by function. In this type of function after complete all statements of function; the keyword ***return*** with value or variable name may be used which is shown in syntax.

EXAMPLE

Program 7.5.1 Write a program to calculate average of two numbers using function with no arguments and return value.

```
#include<stdio.h>
#include<conio.h>
float average( );
void main( )
{
  float result;
  result=average( );
  printf("Average value of two numbers is: %f",result);
  getch();
}
float average( )
{
  int num1,num2;
  float aveg,value;
  printf("Enter the first Number: \n");
  scanf("%d",&num1);
  printf("Enter the Second Number: \n");
  scanf("%d",&num2);
  value=num1+num2;
  aveg=value/2;
  return aveg;
}
………………………………………………………………………………………..
```

Output:

```
Enter the first Number:
10
Enter the Second Number:
15
Average value of two numbers is: 12.500000
```

Activity

1. Write down the benefits of function with No arguments and return value by your own idea.
   ….. ……………………………………………………………………….………
   ….……………………………………………………………………………
2. Create a function named as "power" to determine power of x^{n+1} using function with no arguments and return value concept.
   ….…………………………………………………………………………..
   ………..………………………………………………………………………..
3. Write a program to calculate the value of following formula $A=2\pi r^3$ using function with no arguments and return value concept.
   ………………..……………………………………………………………….
   ……….………..………………………………………………………………

Study skills

1. Determine the output of the following program:

```
void main( ){
        int X;
        X =calculate( ); printf("%d",X);
 }
float calculate( ){

         float a,b;   a= 2; b=5; return (pow(a,b));

     }
```

2. Identify the errors of the following program segments:

```
       void main(){
                  float p, q;
                  q= Calculate( );  printf("%d," q);
              }
        int Calculate( ){
                 float x=15,y=2;  float t=0; t=x/y; return t;
   }
```

Summary Summary	
In this lesson we have	

- Learned about categories of functions with no arguments and return value concept.
- Learned how a function is worked in a program.
- Understood how a function is called in a program and returned value to calling function .

ASSIGNMENT

Assignment

1. Write a program to evaluate Y =1 $+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\cdots+\frac{1}{n!}$ using function with no arguments and return value concept.
   ……..……………………..…………………………………………………..
   ……………………………………………………………………………..
2. Write a program to calculate interest of employee loan using function with no arguments and return value
   …..……….………………………………………………………………
   …..…………………………………………………………………………

Assessment

Assessment

Multiple Choice Questions (MCQ)

1. A function has no arguments but it may-
   a) Return a value
   b) No return value
   c) Call a function
   d) None of these
2. Values of actual arguments are assigned to the-
   a) actual parameters on a *one to one* manner
   b) formal arguments on a *one to many* manner
   c) formal arguments on a *one to one* manner
   d) None of these

Exercises

1. Explain function with no arguments and return value procedure with an example.
2. Write a program to calculate factorial value of a number using function with no arguments and return value procedure.
3. Write a program using function with no arguments and return value procedure that will generate and print first ***n*** Fibonacci numbers.

Lesson-6 Function With Arguments And No Return Value

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Explain the function with arguments and no return value concept.

	Keywords	**Categories, With Arguments, No Return, Value**

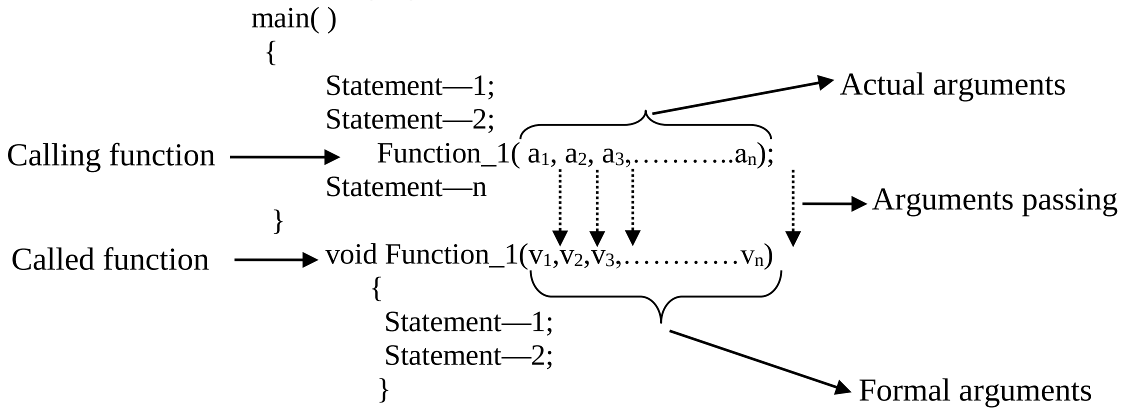
FUNCTION WITH ARGUMENTS AND NO RETURN VALUE

In this type of function category a function takes arguments from calling function but it does not return value. In this case, we could make the calling function to read data from the user or terminal and pass it on to the called function.

The general syntax of this type of function is as follows:

```
void function_name(arguments/parameters)
{
…………………………………
Statements / function body
………………………………..
……………………………….
}
```

Here, *arguments* also known as *parameters* means different data type or same data type value or variable list. There are two types of arguments such as ***(i) actual arguments and (ii) formal arguments.*** The arguments of calling function are called actual arguments and the arguments of called function are called formal arguments. The *actual* and *formal* parameters or arguments should match in number, type, and order. The values of actual arguments are assigned to the formal arguments on a *one to one* manner, starting with the first argument. Illustration of arguments passing between the function calling and the called function is shown in following figure 7.6.1:



```
main( )
 {
      Statement—1;
      Statement—2;
          Function_1( a1, a2, a3,………..an);
      Statement—n
 }
void Function_1(v1,v2,v3,…………vn)
    {
     Statement—1;
     Statement—2;
    }
```

Figure 7.6.1: Arguments passing between calling and called function

Note it!

The arguments of calling function are called actual arguments and the arguments of called function are called formal arguments

The data type should be matched between actual arguments and formal arguments. In case the actual arguments are more than formal arguments, the extra actual arguments are discarded. On the other hand, if the actual arguments are less than the formal arguments, the unmatched formal arguments are initialized to some garbage values. Any mismatch in data type may also result in passing of garbage values. Formal arguments must be valid variable names and the actual arguments may be variable names, expressions, or constants. The variables used in actual arguments must be assigned values before the function call is made. The following example shows the arguments passing procedure between calling and called function.

```
main( )
{
   ………………..
    …………………
    Value_check(10.5,20);   ←———— Calling function
    …………………….
     …………………….
}
void Value_check(float num1, int num2)   ←———— Called function
  {
      …………………..
      …………………..
       ………………….
    }
```

In this above example, when ***Value_check( )*** function calls with float type value 10.5 and integer type value 20, then these values are assigned to num1 and num2 variable of called function respectively.

Note it!

Formal arguments must be valid variable names and the actual arguments may be variable names, expressions, or constants

EXAMPLE

Program 7.6.1 Write a program to calculate greatest common divisor (GCD) of two numbers using function with arguments and no return value.

```
#include <stdio.h>
#include<conio.h>
void gcd(long int n1,long int n2);
void main()
{
   clrscr();
   long int num1, num2,temp;
   printf("Enter two integers: ");
   scanf("%ld %ld", &num1, &num2);

   if(num1>num2)
    {
     temp=num1;
```

```
      num1=num2;
      num2=temp;
     }
   gcd(num1,num2);
   getch();
}
void gcd(long int n1,long int n2)
{
      long int number1,number2;
      number1=n1;
      number2=n2;

      while (n1 != 0)
        {
           long int r = n2 % n1;
           n2 = n1;
           n1 = r;
        }
   printf("GCD of %ld  and %ld is: %ld\n",number1,number2,n2);

}
……………………………………………………………………………………
```

Output:

```
Enter two integers: 420  96
GCD of 420 and 96 is: 12
```

Activity

1. Write down the benefits of function with arguments and no return value by your own idea.
…………………………………………………………………………………….……
..………………………………………………………………………………………

Study skills

1. Identify the errors of the following program segments:

i)
```
main(){
     int p, q;
     Calculate(p,q); printf("%d," p);
}
void Calculate( x, y){
    int t=0;  t=x/y; return t;
   }
```

ii)
```
void main(){
     int x,y; float z; Z=x%y;
     Z=valuetest(z,x,y)
     printf(" output is :%d",z); getch( );
}
valuetest(float p, q, int t){
        Float v;
        v=(pow(t,q)/p);
        return v;
}
```

2. What will be the output of the program 1(ii) after error correction?

Summary Summary	
In this lesson we have ▪ Learned about categories of function with arguments and no return value concept. ▪ Learned how various types of functions are worked in a program.	

ASSIGNMENT

Assignment

1. Create a function named as "power" to determine power of x^y using function with arguments and no return value concept.
   ……………………………………………………………………………………..
   …..…………………………………………………………………………………
2. Write a program to determine the volume of sphere using function with arguments and no return value concept. The formulae of volume of sphere is as follows: $\frac{4}{3}\pi r^3$, Where r is the radius of sphere.
   ………….………………………………………………………………………....

Assessment

Assessment

Multiple Choice Questions (MCQ)

1. Values of actual arguments are assigned to the-
   a) actual parameters on a *one to one* manner
   b) formal arguments on a *one to many* manner
   c) formal arguments on a *one to one* manner
   d) None of these
2. If the actual arguments are more than formal arguments, then-
   a) the extra formal arguments are discarded
   b) the extra actual arguments are changed
   c) the extra formal arguments are discarded
   d) the extra actual arguments are discarded
3. Formal arguments must be valid variable names and –
   a) the actual arguments may be variable names, expressions, or constants
   b) the actual arguments only variable names
   c) the actual arguments may be constants
   d) none of these
4. The arguments of calling function are called actual arguments and the arguments of called function are called-
   a) Formal arguments
   b) Informal arguments
   c) Actual arguments
   d) None of these

Exercises

1. Explain the function with arguments and no return value with an example.
2. Write a program using function with arguments and no return value procedure, that calculates the value of Y in following series:
   $Y=1^2*3^2*5^2+$……………………………………..$*N^2$
3. Write a program using function with arguments and no return value procedure, that calculates the value of Y in following series: Y=n(n-1)(n-2)(n-3)…………………….1.

Lesson-7 Functions With Arguments And Return Value

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Explain the function with arguments and return value concept.

	Keywords	**Categories, Calling function, Called function**

Function With arguments and Return Value

In this type of function category a function takes arguments from calling function and it returns a value to calling function. So in this case, a variable must be used to receive value which is returned form called function. In this category, we could make the calling function to read data from the user or terminal and pass it on to the called function.
The general syntax of this type of function is as follows:

```
return_type function_name(arguments/parameters)
{
…………………………………
Statements / function body
………………………………..
……………………………….
return  value/ variable name;
}
```

Here, *arguments* also known as *parameters* means different data type or same data type value or variable list. There are also two types of arguments such as ***(i) actual arguments and (ii) formal arguments*** which are describe in previous section. The data type should be matched between actual arguments and formal arguments. In case the actual arguments are more than formal arguments, the extra actual arguments are discarded. On the other hand, if the actual arguments are less than the formal arguments, the unmatched formal arguments are initialized to some garbage values. Any mismatch in data type may also result in passing of garbage values. Formal arguments must be valid variable names and the actual arguments may be variable names, expressions, or constants. The variables used in actual arguments must be assigned values before the function call is made.

```
main( )
{
   ………………..
    …………………
    result=Value_check(10.50,20.50);      ←——————  Calling function
    …………………….
     …………………….
}
float Value_check(float num1, float num2)      ←——————  Called function
  {
      value=num1+num2;
      avg=value/2;
      return avg;
}
```

In this above example, when ***Value_check( )*** function calls with float type values 10.50 and 20.50, then these values are assigned to num1 and num2 variables of called function respectively and then calculates summation and average of two numbers and then return the average value (*avg*) to calling function using keyword ***return.*** So the ***result*** variable of calling function will receive or assign average value.

EXAMPLE

Program 7.7.1 Write a program to calculate least common multiple (LCM) of two numbers using function with arguments and return value.(Formulae: LCM=(number1*number2)/GCD)

```
#include <stdio.h>
#include<conio.h>
int lcm(long int n1,long int n2);
void main()
{
    clrscr();
    long int num1, num2,temp,result;
    printf("Enter two integers: ");
    scanf("%ld %ld", &num1, &num2);
    if(num1>num2)
     {
       temp=num1;
       num1=num2;
       num2=temp;
     }
     result=lcm(num1,num2);
    printf("LCM of %ld and %ld is: %ld ",num1,num2,result);
    getch();
}
int lcm(long int n1,long int n2)
{
   long int i,lcm, number1,number2;
   int gcd;
   number1=n1;
   number2=n2;
 while (n1 != 0)
     {
             long int r = n2 % n1;
             n2 = n1;
             n1 = r;
      }
   gcd=n2;
   lcm=(number1*number2)/gcd;
   return lcm;
}
…………………………………………………………………………………..
```

Output:

```
Enter two integers: 420  96
LCM of 420 and 96 is: 3360
```

Activity

1. Write down the benefits of function with arguments and return value by your own idea.
   ……..…………………………..…………………………………………….……
   ..………………………………..………………………………………………
   …..……………………………...……………………………………………
2. Write down the benefits of function with arguments and return value by your own idea.
   ……..…………………………..…………………………………………….……
   ..………………………………..………………………………………………
   …..……………………………...……………………………………………

Study skills

1. Determine the output of the following program:

```
void main()
{
   int x=15,p;
   int y=20;
   p=product(x,y);
   printf("%d",p);
}
int product(int a, int b)
{
    return (a*b);
}
```

2. Identify the errors of the following program segments:

i.
```
main()
 {
      int p, q;
      Calculate(p,q)
      printf("%d," p);
 }
 float Calculate( x, y)
   {
        int t=0;
        t=x/y;
        return t;
     }
```

ii.
```
void main()
 {
        int x,y;
        float z;
       Z=x%y;
       Z=valuetest(z,x,y)
       printf(" output is :%d",z);
       getch( );
 }
 valuetest(float p, q, int t)
 {
       Float v;
       v=(pow(t,q)/p);
      return v;
 }
```

3. What will be the output of the program 2(ii) after error correction?

Summary Summary	

In this lesson we have

- Learned about categoriey of function with arguments and return value concept.
- Learned how various types of functions are worked in a program.
- Also understood how a function is called in a program and returned value to calling function.

ASSIGNMENT

Assignment

1. Write a program to find out greatest number from *n* numbers using function with arguments and return value technique.
   ……………………………………………..……………………………
   …………………………………………….…………………………….
2. Write a program to print prime number list from 1 to 100 using function with arguments with return value concept
   ……………………………………………………………………………
   ……………………………………………………………………………

Assessment

Assessment

Write 'T' for true and 'F' for false for the following sentences

1. The data type should be matched between actual arguments and formal arguments.
2. In function if the actual arguments are less than the formal arguments, the unmatched formal arguments are initialized to some garbage values.
3. The variables used in actual arguments must be assigned values after the function call is made.

Multiple Choice Questions (MCQ)

1. In function any mismatch in data type may also result in passing of –
   a) garbage values
   b) integer values
   c) float types values
   d) global values
2. The data type should be matched between actual arguments and
   a) Global arguments
   b) Local arguments
   c) Formal arguments
   d) Actual arguments

Exercises

1. Explain the function with arguments and return value concept using an example.
2. Write a program to determine the volume of sphere using function with arguments and return value concept. The formulae of volume of sphere is as follows:

$\frac{4}{3}\pi r^3$,Where r is the radius of sphere.

Lesson-8 Recursive Function

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Define recursive function.
- Understand how recursive function executes in program.
- Learn the benefits of recursive functions.

	Keywords	**Recursive function, Advantages, Disadvantages**

RECURSION

In programming language, recursion is a technique or process to calling a function repeatedly. In general, recursion is nothing more than a function that calls itself. When a function calls itself, this type of function is called *recursive function.* The majority computer programming languages support recursion by allowing a function to call itself within the program text. In programming language, while using recursion, programmers need to be careful to define an exit condition from the function; otherwise it will go into an infinite loop. Therefore, every recursive function must be provided with a way to end the recursion. Recursive functions are very helpful to explain and solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, summation of n numbers etc. Consider the following example to understand the recursion technique, where the program to calculates sum of first n numbers using recursion.

```
#include <stdio.h>
int summation(int n);
void main( )
{
   int number,add;
   printf("Enter a positive integer:\n");
   scanf("%d",&number);
   add= summation (number);
   printf("summation of  numbers: %d",add);
}
int summation (int n)
{
   if(n==0)
      return n;
   else
     return n+ summation(n-1);    /*self call  to function summation() */
}
………………………………………………………………………………..
```

Output

```
Enter a positive integer:
6
summation of  numbers:  21
```

In, this example, summation( ) function is invoked from the same function. If *n* is not equal to 0 then, the function calls itself passing argument 1 less than the previous argument it was called with. Suppose, user enter the value of *n* is 6 initially. Then, during next function calls, 5 is passed to function and the value of argument decreases by 1 in each recursive call. When, *n* becomes equal to 0, the value of *n* is returned which is the summation of numbers from 6 to 1. So the result will be 21.

For better visualization of recursion in this example is shown in bellow step by step wise:

```
summation(6)
=6+ summation (5)
=6+5+ summation (4)
=6+5+4+ summation (3)
=6+5+4+3+ summation (2)
=6+5+4+3+2+ summation (1)
=6+5+4+3+2+1+ summation (0)
=6+5+4+3+2+1+0
=6+5+4+3+2+1
=6+5+4+3+3
=6+5+4+6
=6+5+10
=6+15
=21
```

EXAMPLE

Program 7.8.1: Write a program to generate the Fibonacci Series for a given number using a recursive function.

```
#include <stdio.h>
#include<conio.h>
int fibonacci(int i)
 {
  if(i = = 0)
  {
    return 0;
   }
  if(i = = 1)
  {
    return 1;
   }
  return fibonacci(i-1) + fibonacci(i-2);
}
void  main()
{
  clrscr();
  int i,n;
  printf("Enter the Fibonacci Range: \n");
  scanf("%d",&n);
  printf(" Fibonacci series is : \n");
  for (i = 0; i < 10; i++)
  {
    printf("%d\t", fibonacci(i));
  }
  getch();
}
```

```
……………………………………………………………………………….
Output
Enter the Fibonacci Range:
10
Fibonacci series is :
0    1    1    2    3    5    8    13    21    34
```

Program 7.8.2: Write a program to calculate factorial value of a given number using a recursive function.

```
#include<stdio.h>
#include<conio.h>
int factorial(int n);
void main()
{
  clrscr();
  int num,f;
  printf("\nEnter a number: ");
  scanf("%d",&num);
  f= factorial (num);
  printf("\n Factorial value of %d is: %d",num,f);
  getch();
}
int factorial (int n){
   if(n==1)
       return 1;
   else
       return(n* factorial (n-1));
 }



……………………………………………………………………………….
Output
Enter a number: 6
Factorial value of 6  is: 720
```

Note it!

When a function calls itself, this type of function is called recursive function

ADVANTAGES AND DISADVANTAGES OF RECURSIVE FUNCTIONS

Advantages of recursive functions are as follows:

1. Recursion is well-designed and requires few variables which make program clean.
2. It is used to avoidance of unnecessary calling of functions.
3. It is used to as a substitute for iteration where the iterative solution is very complex. For instance to reduce the code size for Tower of Honai application, a recursive function is best suited.
4. Recursion tremendously useful when applying the same solution.
5. Recursion can be used to replace complex nesting code by dividing the problem into the same problem of its sub-type.
6. It is very flexible in data structure such as stacks, queues, linked list and quick sort.

7. The length of the program can be reduced by using recursion technique.

Disadvantages of recursive functions are as follows:

1. Recursion technique is hard to think the logic of a recursive function.
2. It is also difficult to debug the code containing recursion.
3. It needs extra storage space**.**
4. This type of function is not efficient in execution speed and time

Activity

1. What is the benefit of recursive function in program?
   ………………………………………………………………………………….……
   .………………………………………………………………………………………

Summary Summary	
In this lesson we have ▪ Learned about recursion and recursive functions and its definitions. ▪ Also understood how a recursive function is called in a program. ▪ Shown different types of program using recurisive functions.	

ASSIGNMENT

Assignment

1. Write a program to evaluate f(x)= $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ------$ using recursive function.
   ……………………………………………………………………………………..
   .……………………………………………………………………………………
2. Write a program using recursive function to display the following pattern:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
```

Assessment

Assessment

Write 'T' for true and 'F' for false for the following sentences

1. Functions should be arranged in the order in which they are called.
2. C functions can return only one value.
3. A function can call itself.
4. A function in C should have at least one argument.
5. Every function should have a return statement.
6. While using recursion, programmers need to be careful to define an exit condition.
7. Recursion technique uses stack data structure.

Multiple Choice Questions (MCQ)

1. If a function does not return value, we may be-
   a) Used keyword *float* as return type
   b) Used keyword *void* as return type
   c) Used keyword *int* as return type
   d) None of these
2. When a function calls itself, this type of function is called
   a) Recursive function.
   b) Nested function
   c) User defined function
   d) None of these
3. Recursive function is flexible in
   a) Array
   b) Linked list
   c) Data structure
   d) Variable declaration
4. Every recursive function must be provided with-
   a) A way to start the recursion
   b) A way to end the recursion.
   c) A way of length of the program
   d) None of these

Exercises

1. What is recursion and recursive function? Mention the significance of recursion in program.
2. Mention the advantages and disadvantages of recursive function.
3. Write a program to calculate factorial value of a number using recursion process.
4. Write a program using recursive function that will generate and print first ***n*** Fibonacci numbers.

Lesson-9 Uses of Local and Global Variables in Function

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Define local and global variable.
- Understand the scope and lifetime of variables in functions.

	Keywords	**Local and Global Variable, Scope, Lifetime of Variables**

LOCAL AND GLOBAL VARIABLES

We have already known that C allows us to create functions of some sort and also known that functions are used to break up large programs to overcome the program complexity. In programming language, the variables can be categorized depending on the place of their declaration, as internal (i.e., local) or external (i.e., global). A ***local variable*** is a variable that is declared inside a particular function. Local variables are also referred to as automatic variables or internal variables. Local variables are declared inside a function and they are created when the function is called and automatically destroyed when the function is exited. Hence, they are called automatic.
Local variable example is shown in below:

```
main( )
{
  int number;   ←——— Local variable in main( ) function
  …………
 Myfunction( );
}
void Myfunction( )
{
   float area;  }
   int value;   }←——— Local variables in Myfunction( ) function
…………….
}
```

We can also use the keyword ***auto*** to declare automatic/local variables explicitly as follows:

```
void Myfunction( )
{
  auto int value;
  …………….
}
```

In this case, one significant characteristic of automatic variables is that their value cannot be changed accidentally by what happens in other functions in the program. For this reason, we can declare and use the same variable name in different functions in same program without causing any puzzlement to the compiler.
The following illustration shows that how automatic or local variables work in a program:

Program 7.9.1: Write a C program to demonstrate local variable.

```
#include<stdio.h>
#include<conio.h>

void valuetest1( );
void valuetest2( );
void main( )
{
  clrscr();
  int num=2000;
  valuetest2( );
  printf(" %d\n",num);
  getch();
}
void valuetest1( )
{
  int num=20;
  printf(" %d\n",num);
}
void valuetest2( )
{
  int num=200;
  valuetest1( );
  printf(" %d\n",num);
}
………………………………………………………………………………..
```

Output

```
20
200
2000
```

In this above program has two functions valuetest1( ) and valuetest2( ). This program, ***num*** is local variable and it is declared at the beginning of each function. It is (i.e., ***num*** variable) initialized to 20,200, and 2000 in valuetest1( ), valuetest2( ) and main function respectively. When executed this program, main calls valuetest2( ) which also calls valuetest1( ). When main is active, then num=2000; but when valuetest2( ) is called, the main's function **num** is temporarily put on the shelf and the new local variable **num** =200 becomes active. Similarly when valuetest1( ) is called, both the previous values of **num** are put on the shelf and the latest value of **num** =20 becomes active. As soon as valuetest1( )(num=20) is finished , valuetest2( )(num=200) takes over again then, main(m=2000) takes over. For these reasons, the value assigned to **num** is one function does not affect its value in the other functions and the local value of **num** is destroyed when it leaves a function. The clear outputs are shown in illustration.

Note it!

A local variable is a variable that is declared inside a particular function. Local variables are also referred to as automatic variables or internal variables.

On the other hand, a ***global variable*** is a variable that is declared outside **all** functions. Global variables are also referred to as ***external*** variables. It is good practice to declare global variable before main function. Global variables are both alive and active throughout the entire program. For this reason, global variables can be accessed by any function in the program. Once a variable has been declared as global, any function can use it and change its value and then successive functions can reference only that new value.

For instance, the external or global declaration of **integer number** and **float area** might appear as:

```
……………
int number;  }
float area;  }← Global variable declaration
main()
  {
   …………..
   …………..
  }
valuetest1( )
  {
    ……….
    ………
  }
valuetest2( )
  {
   ……….
   ………
  }
```

Here, the variables number and area are available for use in all the three functions such as main, valuetest1, valuetest2. If a local variable and a global variable have the same name, in this case, the local variable will have precedence over global variable in the function where it is declared. Consider the following illustration:

```
……………
 int value;
main( )
{
   value=10;
   ………….
   …………
}
void valuetest( )
 {
   int value=20;
   ………….
   ………….
   value=value+2;
 }
```

Here, we have seen that, global and local variable name (value) is similar. When the valuetest( ) references the variable ***value***, it will be referencing only its local variable, not global variable one. The value of ***value*** variable in main function will not be affected.

The following illustration shows that how global variables work in a program:

Program 7.9.2: Write a C program to demonstrate global variable.

```
#include <stdio.h>
#include<conio.h>
int add_numbers( void );
int  num1, num2, num3;
int sum_numbers( void )
{
            auto int result;
            result = num1+ num2+ num3;
            return result;
}
void main()
{
    clrscr();
    auto int result;
    num1 = 10;
    num2 = 20;
    num3 = 30;
    result = sum_numbers();
    printf("The sum of %d+%d+%d is:= %d\n",num1, num2,num3,result);
    getch();
}
……………………………………………………………………………..
```

Output

```
The sum of 10+20+30 is:= 60
```

Note it!

A global variable is a variable that is declared outside* all *functions. Global variables are also referred to as external variables.

SCOPE AND LIFE TIME OF VARIABLES IN FUNCTION

We have already studied that how global and local variables are worked in function. In this section we will study about scope and life time of variables in function. Scope **is t**he area where a variable can be accessed is known as scope of variable on the other hand life time is the time period for which a variable exists in the memory is known as lifetime of variable.

Actually, the scope and life time of a variable depends on the location where a variable is declared. Therefore, according to their declaration, variables are classified into the following three (03) categories:

1. Block variables
2. Local or internal variables
3. Global or external variables

Block variables

When variables are declared within a pair of braces { } this procedure is called block variables. A block { } may be a self-governing with any control structure but, not with a f. Consider the following statements of block variable declaration procedure:

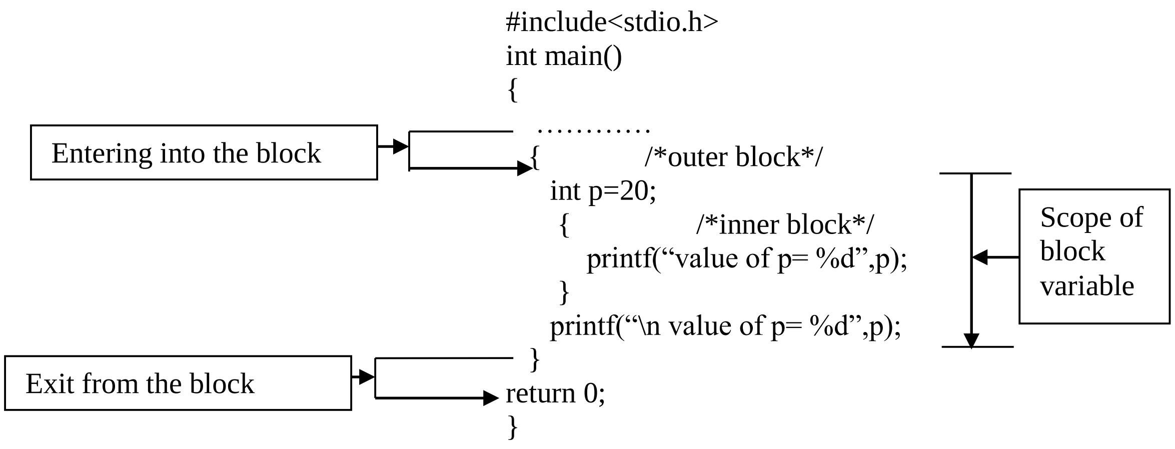
```
…………………….
…………………….
{
   /* independent block */
}

if(x<y)
{
    /* block with control structure */
}
```

Scope of block variables

It is notify that, block variables can be accessed within the block in which they are declared and can also be accessed into the inner block which is within the current block but, can’t be accessed outside the block. Following illustration is shown scope of block variables:

Program 7.9.3: Write a C program to demonstrate scope procedure of block variable



```
#include<stdio.h>
int main()
{
  ...........
  {           /*outer block*/
    int p=20;
     {            /*inner block*/
       printf(“value of p= %d”,p);
     }
    printf(“\n value of p= %d”,p);
  }
return 0;
}
```

…………………………………………………….

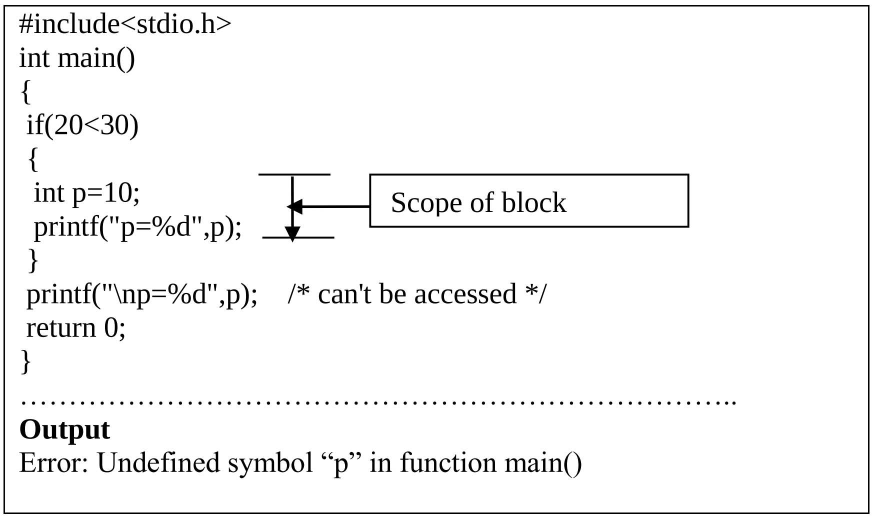
Output

```
value of p=20
value of p=20
```

Here ***p*** variable is declared in the outer block, disappears only when control come out the outer block. Hence, it is available for both inner and outer blocks.

Life time of block variables

In this declaration block variables come into view as the control enters into the block and disappears as the control go out of the block. Hence these variables can’t be accessed outside the block.



```
#include<stdio.h>
int main()
{
 if(20<30)
 {
  int p=10;
  printf("p=%d",p);
 }
 printf("\np=%d",p);    /* can't be accessed */
 return 0;
}
```

…………………………………………………………….

Output

```
Error: Undefined symbol “p” in function main()
```

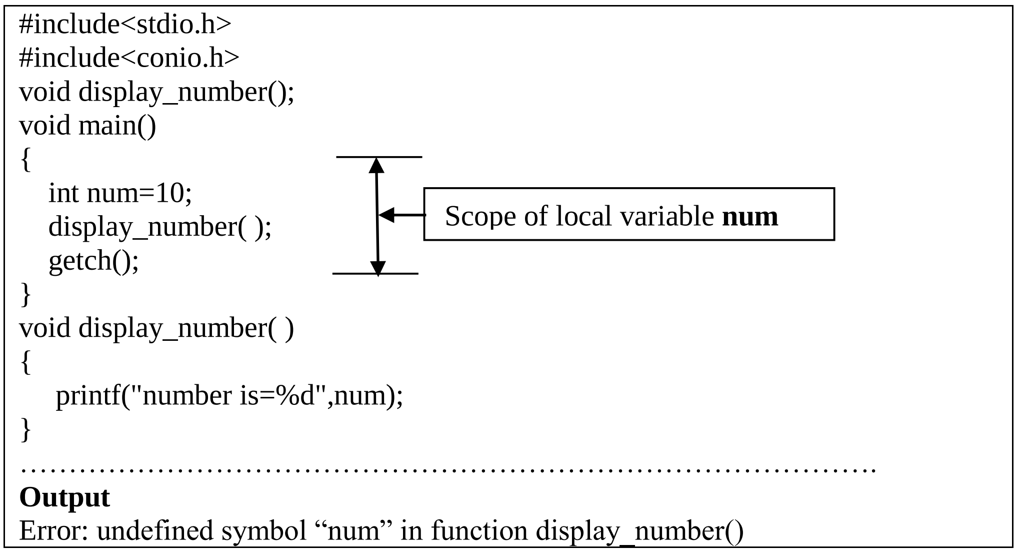
Figure 7.10.1: Block variable life time example

Here, “if statement” has compound statement with a block. There is a conditional expression so the variable p appears as control enters into the block and disappears on exit from the block. Hence variable is not available outside the block.

Scope of local variable

We have already known that a ***local variable*** is a variable that is declared inside a particular function. Local variable can be used only in the function in which it is declared. The declaration and access area is called scope of local variable. The visibility of local variables is limited to the home function in which they are declared, local variables belongs to one function can't be accessed from another function. Consider the following program example:

Program 7.9.4: Write a C program to demonstrate scope procedure of local variable



```
#include<stdio.h>
#include<conio.h>
void display_number();
void main()
{
   int num=10;
   display_number( );
   getch();
}
void display_number( )
{
    printf("number is=%d",num);
}
……………………………………………………………………………….
```

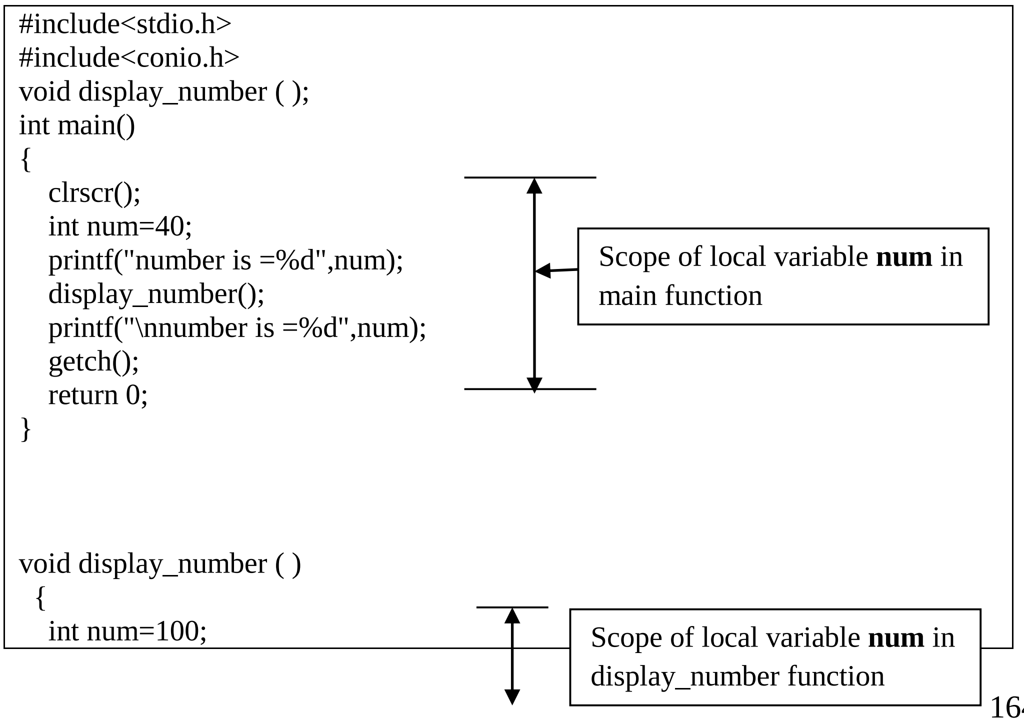
Output

```
Error: undefined symbol "num" in function display_number()
```

Figure 7.10.2: Local variable scope example-1

Here, ***num*** variable is the local variable to main( ) function, it can't be accessed from display_number( ) function, because the scope of ***num*** variable is limited to the main( ) function only. For this reason an error will occur.

Again, consider the following program example:


Scope of local variable **num** in main function
Scope of local variable **num** in display_number function

```
#include<stdio.h>
#include<conio.h>
void display_number ( );
int main()
{
   clrscr();
   int num=40;
   printf("number is =%d",num);
   display_number();
   printf("\nnumber is =%d",num);
   getch();
   return 0;
}



void display_number ( )
  {
   int num=100;
  printf("\nnumber is =%d",num);
 }
……………………………………………………………………………………..
Output
nnumber is =40
nnumber is =100
nnumber is =40
```

Figure 5.10.3: Local variable scope example-2

Here variable ***num*** is declared for two times that is in main( ) function and again in display_number( ) function. They are different memory locations; visibility of which is limited to their home functions. So we have conclude that, variable belongs to one function can't be accessed from another function.

Note it!

The declaration and access area is called scope of local variable.

Life time of local variables

We know that the time period for which a variable exists in the memory is known as lifetime of variable. Lifetime of local variables starts when control enters the function in which it is declared and it is destroyed when control exists from the function. Local variables of a function do not remain their values from one execution to another of a function because they lose their life at the end of every execution. We have described this procedure in the previous section. Consider the following example as life time of local variables.

Program 7.9.5: Write a C program to demonstrate life time procedure of local variable

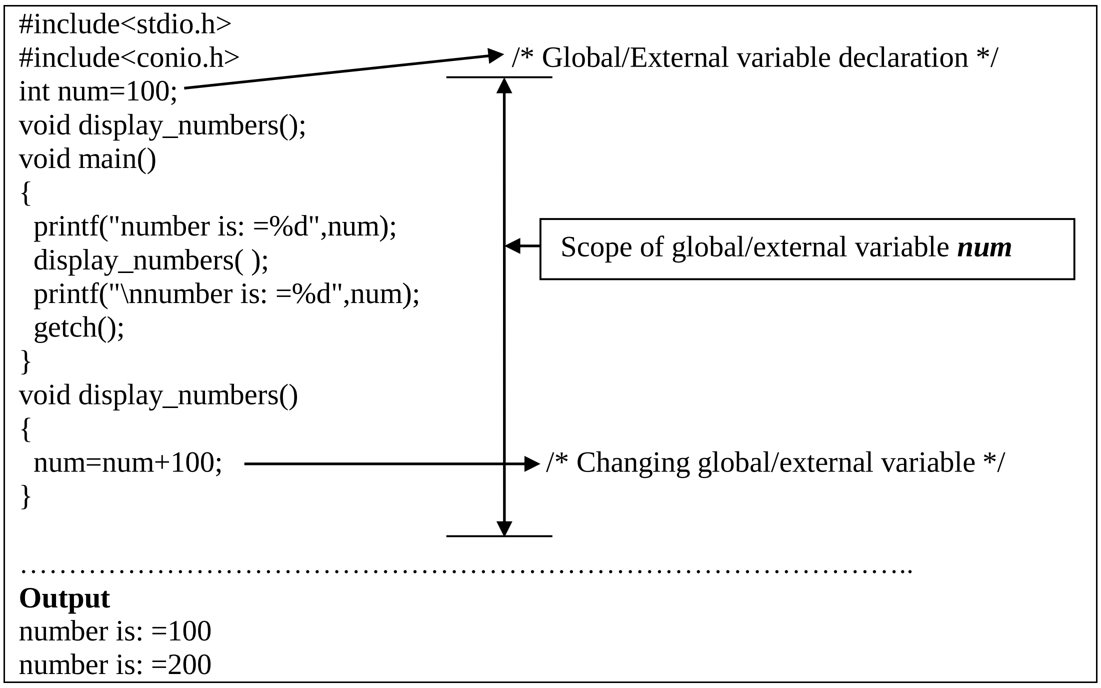
```
#include<stdio.h>
void display_numbers();
int main()
{
   display_numbers();
   display_numbers();
   display_numbers();
 return 0;
}
void display_numbers()
{
 int num=20;
 nump=num+20;
 printf("\n number is: %d",num);
}
……………………………………………………………………………………..
Output
number is: 40
number is: 40
number is: 40
```

In this example, ***num*** is a local variable to the function display_numbers( ), it doesn't keep its values for the next execution because it loses its life at the end of every execution. Hence, for every execution ***num*** is allocated and assigned with 20. So, the result would be same for every function call.

Scope of global variable

We know that a ***global variable*** is a variable that is declared outside **all** functions. Global variables are also referred to as ***external*** variables. The global variable can be used by all functions in the program. These types of variables are globally accessed from any part of the program like function. These are declared before main function. The global variable declaration area is called scope of global variable. The main scope concept of global variables is that, these variables are visible only to the functions, which are down to their definition; these can't be accessed from the functions above to their definition. Therefore, global variables are declared before main function. Consider the following program example:

Program 7.9.6: Write a C program to demonstrate scope procedure of global variable


/* Global/External variable declaration */
Scope of global/external variable ***num***
/* Changing global/external variable */

```
#include<stdio.h>
#include<conio.h>
int num=100;
void display_numbers();
void main()
{
  printf("number is: =%d",num);
  display_numbers( );
  printf("\nnumber is: =%d",num);
  getch();
}
void display_numbers()
{
  num=num+100;
}
```

…………………………………………………………………………………..

Output

```
number is: =100
number is: =200
```

Figure 7.10.4: Global variable scope example-1

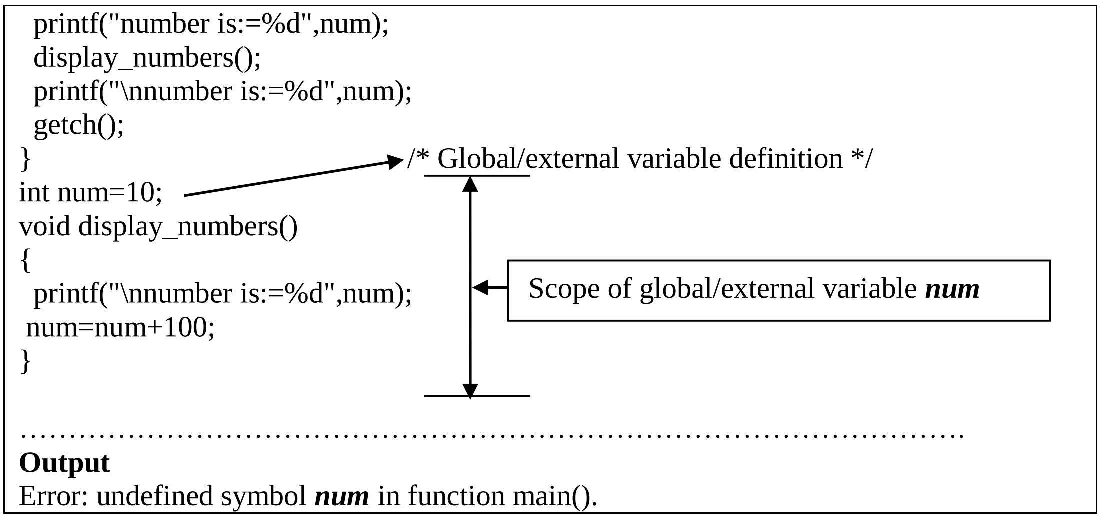
Here, in the above example, variable ***num*** is declared on top of all the functions. It is global to both the functions main( ) and display_numbers( ). The change in its value ***(num=num+100)*** in the function display_nnumbers() reflected the change in main() because ***num*** is a common variable to both the functions.

Note it!

The main scope concept of global variables is that, these variables are visible only to the functions, which are down to their definition; these can't be accessed from the functions above to their definition

Consider the following another example:

```
#include<stdio.h>
#include<conio.h>
void display_numbers();
voidt main()
{
```



```
  printf("number is:=%d",num);
  display_numbers();
  printf("\nnumber is:=%d",num);
  getch();
}                                 /* Global/external variable definition */
int num=10;
void display_numbers()
{
  printf("\nnumber is:=%d",num);      Scope of global/external variable num
 num=num+100;
}

…………………………………………………………………………………….
Output
Error: undefined symbol num in function main().
```

Figure 7.10.5: Global variable scope example-2

Here, in the above example ***num*** is an global/external variable because declared outside both the function main() and display_numbers(). But it is declared below the main() and above the display_numbers() function, so it is global to display_numbers( ) but, not to the main() function. Therefore, ***num*** can't be accessed from the main(), so compiler produces an error. Global variables are initialized with **zero** if not assigned with any value by the program.

Life time of global variable

We already knew that, the time period for which a variable exists in the memory is known as lifetime of variable. Global variables survive in the memory as long as the program is running. These variables are destroyed from the memory when the program terminates. Therefore, these variables occupy memory longer than local variables.

Note it!

The time period for which a variable exists in the memory is known as lifetime of variable

Activity

1. What are the benefits of local and global variables?
   ………………………………………………………………………………………
   ………………………………………………………………………………………

Study skills

1. Mention the lines which are belongs as scope of variable *test* from the following program segment

```
#include<stdio.h>
..................
void main()
{
  ..........................
 float x,y;
 test();
}
void test()
```

```
{
    int test=200;
     if(test<500)
         {
           test=300;
           printf("the value of test variable is:=%d",test);
         }
    else
        test=400;
  }
```

2. Mention the lines which are belongs as scope of variable *count* from the following program segment

```
#include<stdio.h>
int count= 5;
Void main()
{
  ..........................
 float x,y;
 test(count);
}
void test(int Z)
{
    count=count++;
     if(count>=10)
         {
          count=0;
           printf("the value of count variable is:=%d",count);
         }
    else
        count++;
}
```

Summary Summary	
In this lesson we have ▪ Learned about local and global variables. ▪ Learned how local and global variables are executed in a program. ▪ Learned about scope and life time of variables.	

ASSIGNMENT

Assignment

1. Write a program to evaluate f(x) =$x+\frac{x^3}{3!}+\frac{x^5}{5!}+\frac{x^7}{7!}+-------$ using local and global variable concept.
   ……..……………………………………………………………………………..
   ………………………………………………………………………………….
2. Write a multifunction program to illustrate the properties of global variables
   ………..……………………………………………………………………
   ………..……………………………………………………………………

Assessment

Assessment

Write 'T' for true and 'F' for false for the following sentences

1. A global variable can be used only in main function.
2. Global variables cannot survive in the memory as long as the program is running.
3. A *local variable* is a variable that is declared outside a particular function.
4. Life time is the time interval for which a variable exists in the memory.
5. We can declare and use the same variable name in different functions in same program.

Multiple Choice Questions (MCQ)

1. Local variables are also referred to as-
   a) Automatic variables. b) External variables c) Fixed type variables d) None of these
2. a global variable is a variable that is declared-
   a) Outside all functions b) Inside of main function c) Outside only a function d) Only outside of main function
3. Global variables are
   a) Only alive in the entire program b) Both alive and active throughout the entire program
   c) Only active throughout the entire program d) None of these
4. The area where a variable can be accessed is known as
   a) Block of variable b) Non block of variable c) Scope of variable d) Life time of variable
5. Block variables are declared
   a) within a pair of braces { } b) within a pair of [ ] c) within a pair ( ) d) None of these

Exercise

1. Define local and global variable and explain with their properties with an example.
2. Explain the necessity of global variables in a program.
3. Distinguish between global and local variables.
4. What do you mean by scope and life time of variable? Explain with proper example.