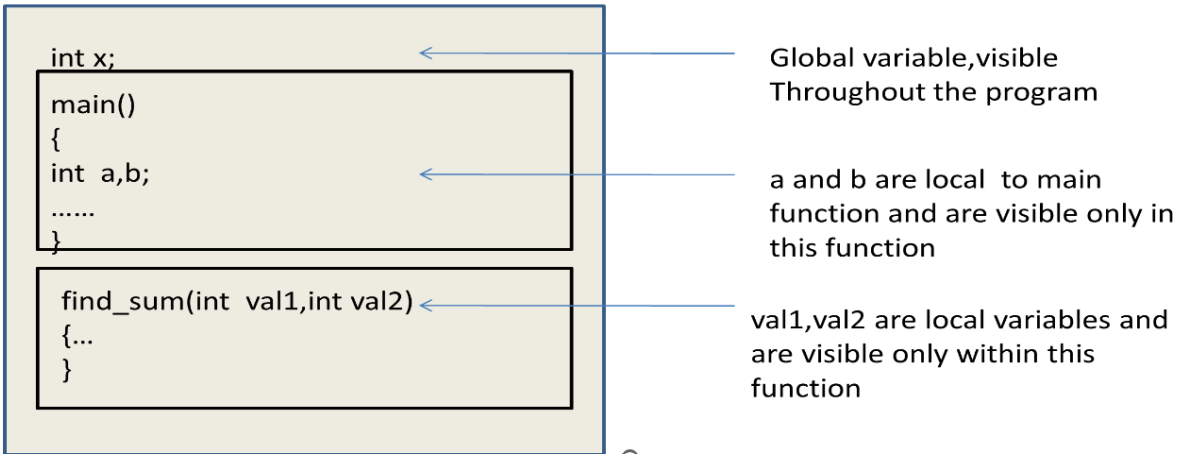
Scope and Lifetime of variables

Scope determines where a variable or constant can be used and where a function can be referenced or invoked.When the visibility of a variable, constant or function is confined to a section of the program, the scope is said to be local.Otherwise the scope is said to be global. When a program starts executing,all of its global variables are allocated storage in main memory.All the local variable are allocated storage in the main memory when a function starts executing.The memory allocated to the global variables remains for the entire duration of the program.Whereas ,memory allocated for local variables is released as soon as the function finishes its execution.The term lifetime of a variable or constant refers to the duration of its storage allocation during the execution of the program.Thus,the lifetime of global variables is for the entire duration of the program.The lifetime of local variables is limited to the execution of the function that defines them.



Local variables declared with the static keyword retain their values even after the function in which they are declared has finished executing.
Static variable are only visible when the function in which they are defined is executing.

Place of declaration	**Keyword**	**Visibility**	**Lifetime**
Before all functions		Throughout the program	For the entire duration of the program
Inside a function		Only within the function	While function is executing
Inside a function	Static	Only within the function	For the entire duration of the program

Program Based on local and global variable

```
#include<stdio.h>
int x=10;  //global variable
int main()
{
  int x=5;//local variable
  int y =15;//local to main
  printf("x=%d",x);

  sample();

}
void sample()
{  int temp=5;//local to sample
 printf("\nx=%d",x);
//printf("%d",y);
printf("%d",temp);
}
```

Functions

Functions break large computing tasks into smaller ones, and enable people to build on what others have done instead of starting over from scratch. Appropriate functions hide details of operation from parts of the program that don't need to know about them, thus clarifying the whole,and easing the pain of making changes.C has been designed to make functions efficient and easy to use; C programs generally consist of many small functions rather than a few big ones.

Each function definition has the form

```
return-type function-name(argument declarations)
{
declarations and statements
}
```

Function for addition of 2 numbers

```
#include<stdio.h>
int addnum(int,int); //function prototype or declaration
void main()
{
int a,b,c;
printf("enter 2 num");
scanf("%d%d",&a,&b);

c=addnum(a,b); //function call
printf("The addition is %d",c);
getch();
}

//function definition
int addnum(int a,int b)
{
  return (a+b);
}
```

Call by value, Call by reference

The arguments passed to function can be of two types namely
1. Values passed
2. Address passed

The first type refers to call by value and the second type refers to call by reference.

consider **program1**

```
main()
{
int x=50, y=70;
interchange(x,y);
printf("x=%d y=%d",x,y);
}

interchange(int x1, int y1)
{
int z1;
z1=x1;
x1=y1;
y1=z1;
printf("x1=%d y1=%d",x1,y1);
}
```

Here the value to function interchange is passed by value.

Consider **program2**

```
main()
{
int x=50, y=70;
interchange(&x,&y);
printf("x=%d y=%d",x,y);
}

interchange( int *x1, int *y1)
{
int z1;
z1=*x1;
*x1=*y1;
*y1=z1;
printf("*x=%d *y=%d",x1,y1);
}
```

Here the function is called by reference. In other words address is passed by using symbol & and the value is accessed by using symbol *.
The main difference between them can be seen by analyzing the output of program1 and program2.
The output of program1 that is call by value is
x1=70 y1=50
x=50 y=70
But the output of program2 that is call by reference is
*x=70 *y=50
x=70 y=50

This is because in case of call by value the value is passed to function named as interchange and there the value got interchanged and got printed as

x1=70 y1=50
and again since no values are returned back and therefore original values of x and yas in main function namely
x=50 y=70 got printed.

But in case of call by reference address of the variable got passed and therefore what ever changes that happened in function interchange got reflected in the address location and therefore got reflected in original function call in main also without explicit return value.
So value got printed as *x=70 *y=50 and x=70 y=50

Program based on nesting of functions

```
#include<stdio.h>
int check_positive(int,int);
int divide(int,int);
int main()
{
  int numerator,denominator,result;
  printf("Enter the numerator:-");
  scanf("%d",&numerator);
  printf("\nEnter the denominator:-");
  scanf("%d",&denominator);

  result=divide(numerator,denominator);
  if(result==0)
  printf("error");
  else
  printf("Quotient is %d",result);
  return 0;
}
int divide(int n,int d)
{

  if(check_positive(n,d))
  {
    return n/d;
  }
  else
  return 0;

}
int check_positive(int n,int d)
{
  if(n >0 && d>0)
  return 1;
  else
  return 0;
}
```