# Scope of variables

- In C, constants and variables have a defined scope.
  - i.e Accessibility and visibility of constants and variables at different points in the program.
- A variable or a constant has four types of scope:
  - Block, Function, Program and File

**i) Block scope:**

- If a variable is declared within a statement block {pair of curly braces} , then if the control exits that block, all variables within that block cease to exist.
- Such variables are known as *local variables*.
- The *local variables* have block scope.
  - A program may have nested blocks.
    - ✓ For Ex: while loop inside main( )
    - ✓ If an integer x is declared and initialized in main( ), and then re-declared and re-initialized in a while loop, then variable x will occupy two different memory locations.

**Example:**

```
#include<stdio.h>
void main( )
{
int x=20, i=0;
printf("\n Value of x outside while loop =
        %d", x);

while(i<3)
{
  int x=i;

  printf("\n Value of x inside while loop =
          %d", x);
  i++;
 }

printf("\n Value of x outside while loop =
        %d", x);
}
```

**Output**

```
Value of x outside while loop = 20

Value of x inside while loop = 0

Value of x inside while loop = 1

Value of x inside while loop = 2

Value of x outside while loop = 20
```

***In nested blocks, variables declared outside the inner blocks are accessible to the nested blocks provided these variables are not re-declared within the inner block.***

## 2. Function scope:

***Function scope*** **indicates that the variable is active and visible from beginning to end of the function.**

- **In C, only goto label has *Function scope*.**

**Example:**

```
#include<stdio.h>
void main( )
{
  .....
  .....
  Loop: printf("Hi !");          /* A goto label has function scope
  .....
  .....
  goto Loop;
  .....
  .....
}
```

- **Here, the label Loop is visible from beginning to end of main( ) function.**
- **Therefore, there should not be more than one label having same name in main( ) function.**

## 3. Program scope:

- **The variables declared within a function are called *local variable*.**
- **The local variables are not known to other functions in the program.**
- **The local variables cease to exist if the control goes out of that function in which they are declared.**
- **However, if the function wants to access the variables which are not declared in that function, they have to be declared outside all the functions [*before main( )*] and such variables are called *global variables*.**
- **The *global variables* remain existing throughout the period of execution of the program.**
- **All the functions in a program can access *global variables. They exist even when a function calls another function.***

**Note:** ***If the local variables and global variables have the same name, then the function uses the local variable declared within it and ignores the global variable.***

**Example:**

```
#include<stdio.h>
int x=50;

void print( );     //Function declaration

void main( )
{
   printf(" \nValue of x in the main( ) = %d ", x);

   int x=20;     //Local variable to main

   printf(" \nValue of local variable x in the main( ) = %d ", x);
   print( );
}

void print( )     //Function
{
  printf(" \nValue of local variable x in the print( ) = %d ", x);

}
```

**OUTPUT**

```
Value of x in the main( ) = 50

Value of local variable x in the main( ) = 20

Value of local variable x in the print( ) = 50
```

**Note: The *local variables overwrite global variables***

**4. File scope:**

- **The global variable is accessible until the end of the file, then it is said to have the *file scope*.**
- **To allow the file to have a *file scope*, it must be declared with the *static* keyword.**

**Example:** ***static*** **int x=50;**

- **The global static variable can't be accessed by any other files except the file in which it is declared.**
- **They are useful to write own header files.**

## STORAGE CLASSES

- **Variables in C can have *storage class* in addition to data type.**
- **The storage class defines the scope (visibility) and life-time of the variables declared in the function.**

- **Consider the example:**

```
/* Example of storage class */
int m ;
void main ( )
{
    int i ;
    float balance ;
    ......
    ......
    function1( );
}
```

```
function1 ( )
{
    int i ;
    float sum ;
    ......
    ......
}
```

- **Here, variable m is called *global variable* because it is declared before main( ).**
- **It can be used in all the functions defined in the program.**
- **It need not be declared in other functions.**
- **The *global variable* is also known as *external variable*.**
- **The variables i, balance, and sum are called *local variables* because they are declared inside the functions.**
- ***Local variables* are visible and meaningful only inside the functions in which they are declared.**
- **Note that the variable i is declared in both the functions.**
    - **Any change in the value of i in one function does not affect its value in another function.**

- **The storage class gives the following information about the variables or the functions.**
    - ***It determines the part of memory where storage space will be allocated for that variable or function (i.e a register or RAM).***
    - ***It specifies how long the storage allocation will continue to exist for that variable or function.***

- ***It specifies whether the function or the variable can be referenced throughout the program or only within the function / block / source file where it is declared.***
- ***It specifies whether the function or the variable has internal or external or no linkage.***
- ***It specifies whether the variable will be automatically initialized to zero or any indeterminate value.***

➢ **C provides number of storage class specifiers that can be used to declare explicitly the *scope* and *lifetime* of variables.**

**There are 4 storage class specifiers as shown in the following table.**

| **Storage Class** | **Meaning** |
|---|---|
| ***auto*** | **Local variable known to the function in which it is declared. Default is auto.** |
| ***static*** | **Local variable which exists and retains its value even after the control is transferred to the calling function.** |
| ***extern*** | **Global variable known to all functions in the file.** |
| ***register*** | **Local variable which is stored in the register** |

**Example:**

```
auto int count;
register char ch;
static int x;
extern long total;
```

- **static and extern variables are automatically initialized to zero.**
- **auto variables contain undefined values (*garbage values*) if they are not initialized explicitly.**

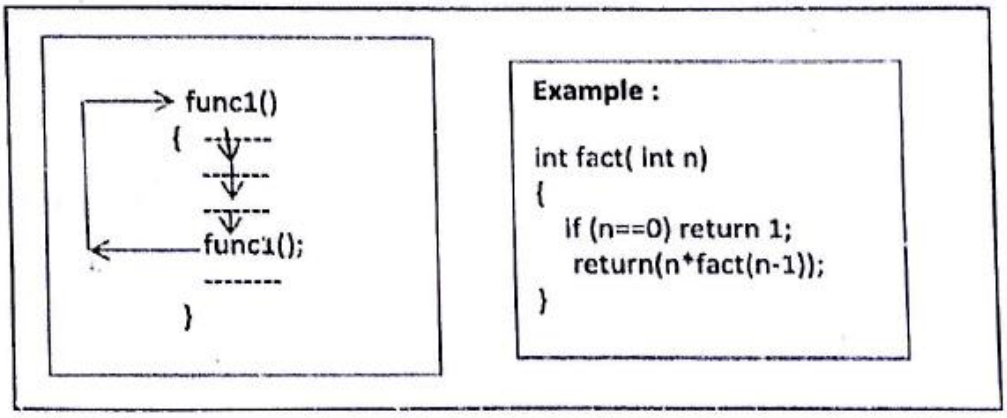
# Recursive functions

**Recursion**: The process of a function calling itself is called recursive. The recursion is a method of solving the problem where the solution to a problem depends on solutions to smaller instances of the same problem. Thus a ***recursive function is a function that calls itself during execution.*** The recursion refers to the process where a function calls itself either directly or indirectly. There are two types of recursion as given below:

1. Direct Recursion
2. Indirect Recursion

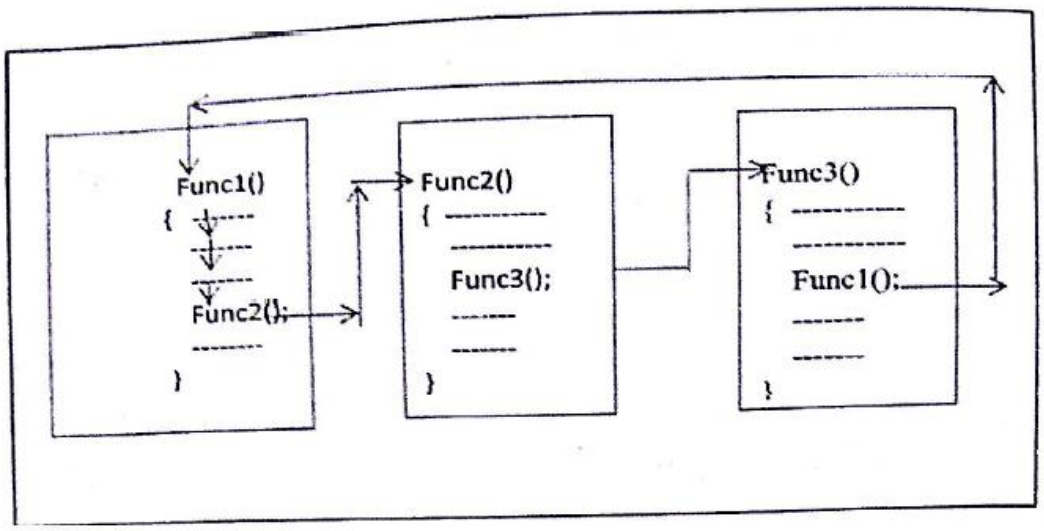


**Direct Recursion** : It refers to a process where a function calls itself directly as illustrated in the figure below :




As illustrated in the figure above **func1()** calls **func1()** . This type of calling itself is called direct recursion and the recursion will takes place until the given condition is satisfied. In the example given above the **fact(n)** calls itself and the called function returns **n*fact(n-1)** until the value of n becomes equal to **0**.

**Indirect Recursion**: It refers to a process where a function calls another function and that function calls back the original function. The process of indirect recursion takes place as illustrated in the figure below:




Here the Func1() calls Func2() and Func2() calls Func3() and Func3() calls Func1().

**Example : C Program to find the factorial of a given number using recursion .**



```
#include <stdio.h>
#include<conio.h>
int factorial(int);
int main()
{ int   num;
  int   result;
  clrscr();
   printf("Enter a number to find it's Factorial: ");
  scanf("%d", &num);
  if (num < 0)
  { printf("Factorial of negative number  is  not possible\n");
  }
  else
  {
     result = factorial(num);
     printf("The Factorial of %d is %d.\n", num, result);
  }
  getch();
  return 0;
}
```

```
int factorial (int num)
{
  if (num==1)
     return 1;
  else
     return(num*factorial(num-1));
}
```

**Output 1**

**Enter number to find its factorial: 0**
**Factorial of negative number is not possible**

**Output 2**

**Enter number to find its factorial: 6**
**The factorial of 6 is 620**

**Write a program to compute Fibonacci series upto n terms using recursion**

```
/*Program to compute Fibonacci Series using Recursion */
#include<stdio.h>
int fibo(int m);
void main( )
{
   int n, i=0, c;

   clrscr( );
   printf("Enter number of terms :  ");
   scanf("%d", &n);

   printf("FIBONACCI SERIES : \n ");

   for( c=1; c<=n; c++)
   {
       printf(" %d   ", fibo(i));    //Function call
       i++;
   }
} // End of main

/* Function to compute Fibonacci series */
int fibo(int m)
{
   if (m==o)
       retutn 0;
   else
       if (m==1)
          return 1;
       else
          return( fibo(m-1) + fibo(m-2));    //Function call
} //End of function
```

**Output**

**Enter number of terms: 7**
**FIBONACCI SERIES :**
**0 1 1 2 3 5  8**

# ARRAYS

**Array: Array is the collection of elements of same datatype.**
**It is a derived datatype.**
**Arrays are of two types:**

1. **One-dimensional arrays**
2. **Multi-dimensional arrays**

one Dimensional arrays.

**Single Dimensional Arrays**: Single dimensional array is one in which the array variable uses single subscript and single pair of square brackets to store multiple values of similar data types .

**Example:** x[10], std[10], usn[10] etc. are examples for single dimensional array . The elements are stored contiguously one after the other as illustrated below:

| 10 | 20 | 15 | 30 | 35 |
|---|---|---|---|---|
| x[0] | x[1] | x[2] | x[3] | x[4] |

The elements of single dimensional array can be used in programs just like any other C variable. For example the following are valid statements.

- a = x[0] +10;
- x[4] = p[0] + r[2] ;
- x[2] = n[i]*3

**Declaration of one Dimensional Arrays:** To use an array, we must declare it. Like any other variable, arrays must be declared before they are used so that the compiler can allocate space for them in memory. The general form of array declaration is

**type variable_name[SIZE];**

Here, type is data type like **int** or **char** or **float** . Size is the maximum number of elements that can be stored inside the array

**Example** : float height[50];
int group[10];
char name[10];

**Note: When declaring character arrays, we must allow one extra space for null character.**

## Initializtion of One –Dimensional Arrays:

**After an array is declared, its elements must be initialized. Otherwise, they will contain *garbage* values.**

**An array can be initialized in two ways:**

1. **At compile time**
2. **At run time**

1. **Compile time Initialization:** In this type, the initialization is done during declaration or anywhere inside the program before running a program. The general form of initialization of arrays during compile time is :

**type variable_name[SIZE] = { list of values };**

**Examples :**

```
int    number[3] = {0,0,0};
float  total[5] = {0.0,15.75,-10};
int    counter[ ] = {1,1,1,1};
char   name[ ] = { 'J','o','h','n','\0'};
char   name[ ] = "John";
```



**Arrays can also be partially initialized as illustrated below:**

→ int n[5] = {10,20}; Here the remaining elements will be initialized to zero.

→ char city[5] ={'B'}; Remaining elements will be initialized to null .

**Note:** The type of initialization n[3] = {10,20,30,40} is illegal in C programming language. [∵ size is 3 but 4 elements are initialized]

## 2.Run-time initialization:

**Here, the initialization is done during program execution.**

**This type is usually used to initialize large arrays.**

**Example:**

```
float sum[50];
for(i=0; i<50; i++)
{
  if (i<25)
     sum[i]=0.0;
  else
     sum[i]=1.0;
}
```

## Using constants as array size:

- **The constants can also be used to define the array size.**
- **But the constants must have been defined earlier before the usage of constants in array subscript as follows:**

```
#define NUM 100
int test[NUM];
```

**Note: The following way of declaration is illegal.**

```
int NUM=100;
int test[NUM];
```

## Reading One Dimensional arrays:

**To read one dimensional arrays, we use for loop and scanf() statement as follows:**

```
int a[10];
for(i=0; i<5; i++)
{
   scanf("%d", &a[i]);
}
```

**The above segment reads 5 elements into array called 'a'.**

## Writing One Dimensional arrays:

**To write one dimensional arrays, we use for loop and printf() statement as follows:**

```
int a[10];
for(i=0; i<5; i++)
{
   printf("%d", a[i]);
}
```

**The above segment writes 5 elements from array called 'a'.**

## Program to Read and Write a One Dimensional Array

```
/* Reading and Writing a one dimensional array */
#include<stdio.h>
void main ( )
{
   int a[10], n, i;

   clrscr( );
   printf (" Enter number of elements :");
   scanf (" %d", &n);
   /* Reading an array */
   printf (" \n Enter array elements:\n");
   for (i = 0 ; i <n ; i++)
        scanf ("%d", &a[i]);

   /* Writing an array */
   printf (" \n The given array is : \n");
   for (i=0 ; i<n ; i++)
        printf (" %d  ", a[i]);
} /* End of main */.
```



## Write a program to find Sum of array elements

```
/*pgm to find sum of array elements */
void main ( )
{  int a[10], n, i, sum = 0;
   clrscr( );
   printf(" Enter size of array :");
   scanf (" %d", &n );
  /* Reading the array */
  printf (" \n Enter array elements: \n");
  for (i=0 ; i<n ; i++)
       scanf (" %d", &a[i]);

  /* Writing the array */
  printf (" \n The given array is : \n");
  for (i=0 ; i<n ; i++)
       printf (" %d  ", a[i];
```

```
  /* Computing Sum */
  for ( i = 0 ; i <n ; i++)
      sum = sum + a[i];

  printf (" The sum of array
                elements = %d", sum);

} /* End of main */
```