KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Website: www.ktunotes.in

MODULE IV

Working with functions-Introduction to modular programming, writing functions, formal parameters, actual parameters, Pass by Value, Recursion, Arrays as Function Parameters structure, union, Storage Classes, Scope and life time of variables, simple programs using functions

MODULAR PROGRAMMING

Modular programming is the process of subdividing a computer program into separate sub-programs. It is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

Advantages

- **Ease of Use :**This approach allows simplicity, as rather than focusing on the entire thousands and millions of lines code in one go, we can access it in the form of modules.
- **Programming errors are easy to detect:** Minimizes the risks of ending up with programming errors and also makes it easier to spot errors, if any.
- **Allows re-use of codes:** A program module is capable of being re-used in a program which minimizes the development of redundant codes
- **Improves manageability:** Having a program broken into smaller sub-programs allows for easier management.

FUNCTION

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and can have additional functions.

Types of Functions

There are two types of functions in C programming:

- Standard library functions
- User-defined functions

STANDARD LIBRARY FUNCTIONS

The standard library functions are built-in functions in C programming. These functions are defined in header files. For example,

- The printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the stdio.h header file.
- The sqrt() function calculates the square root of a number. The function is defined in the math.h header file.
- The strlen() function calculates the length of a given string. The function is defined in the string.h header file.

USER-DEFINED FUNCTIONS

Functions that we define ourselves to do certain specific task are referred as user-defined functions.

Advantages of user-defined functions

- The program will be easier to understand, maintain and debug.
- Reusable codes that can be used in other programs
- A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

Three parts of a user defined functions are:

1) Function Declaration or Prototype
2) Function Definition
3) Function Call

Function Declaration

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body. A function prototype gives information to the compiler that the function may later be used in the program.

Syntax

return_type function_name(parameter list);

Note:

- ✓ If function definition is written after main, then only we write prototype declaration in global declaration section
- ✓ If function definition is written above the main function then ,no need to write prototype declaration

Function Definition

A function definition in C programming consists of a *function header* and a *function body*. Function header consist of return type,function name,arguments(parameters).

- **Return Type**: A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**
- **Function Name**: The actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters**: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. Parameters are optional; that is, a function may contain no parameters.
- **Function Body**: The function body contains a collection of statements that define what the function does.

Syntax:

return_type function_name(argument list) {

body of the function

}

Function Call

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

Syntax:

functionName(parameter list);

Example:

```
int add(int,int,int);                    // function declaration

void main()

{
        int x=10,y=20,z=30,res;
        res = add(x,y,z);                // function call
        printf("Result=%d",res);
}


int add(int a, int b, int c)             // function definition
{
        int sum;
        sum = a+ b + c;
        return sum;                      //return statement
}
```

Return Statement

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after the return statement.

In the above example, the value of the **sum** variable is returned to the main function. The **res** variable in the main() function is assigned this value.

Formal and Actual Parameters

There are different ways in which parameters can be passed into and out of functions. Let us assume that a function *B()* is called from another function *A()*. In this case *A* is called the ***"caller function"*** and *B* is called the ***"called function or callee function"***. Also, the arguments which *A* sends to *B* are called *actual arguments* and the parameters of *B* are called *formal arguments*.

- **Formal Parameter:** A variable and its type as they appear in the prototype of the function or method.
- **Actual Parameter:** The variable or expression corresponding to a formal parameter that appears in the function call in the calling environment.

In the above example, x, y and z in the main function are the actual parameter of add function. Formal parameter of add function are a, b and c.

PASS BY VALUE

In this parameter passing technique, a copy of actual parameter is passed to formal parameter. As a result, any changes or modification happened to formal parameter won't reflect back to actual parameter. This can be explained with an example of swapping program.

```
#include<stdio.h>
int swap(int a,int b)
{
        int temp;
        temp=a;
        a= b;
        b=temp;
}

void main()
{
        int x,y;
        printf("Enter the numbers:");
        scanf("%d%d",&x,&y);
        printf("Before swapping : x=%d\ty=%d\n",x,y);
        swap(x,y);
        printf("After swapping : x=%d\ty=%d",x,y);
}
```


Output
Enter the numbers:10 20
Before swapping: x=10 y=20
After swapping : x=10 y=20

In above program we expect the program to swap the value of x and y after calling the function swap with x and y as actual parameter. But swapping does not take place as this uses call by value as parameter passing technique.

During Swap call, a copy of actual parameters are created and changes are made on that copy. So the value of x and y does not changes.

After reading the value of x and y

x: 10	y: 20

During swap call a copy is created and passed to formal parameter

x: 10	a: 10

y: 20	b: 20

After the swap call, formal parameters get swapped but actual parameters remains the same.

x: 10	a: 20
y: 20	b: 10

TYPES OF USER DEFINED FUNCTIONS

- Functions with no arguments and no return values
- Functions with arguments and no return values
- Functions with no arguments and but return a value
- Functions with arguments and one return values
- Function that have multiple return values(work based on condition)

- **Functions with no arguments and no return values**

// function to read two numbers and print its sum

```
void add()
{
        int a,b,sum;
        printf("Enter the values of a & b");
        scanf("%d%d",&a,&b);
        sum = a+b;
        printf("Sum=%d",sum);
}
```

- **Functions with arguments and no return values**

// function that takes two arguments and print their sum.

```
void add(int a,int b)
{
        int sum;
        sum= a + b;
```

```
        printf("Sum=%d",sum);
    }
```

- **Functions with no arguments and return one value**

  // function to read two numbers and return its sum

```
int add()
{
        int a,b,sum;
        printf("Enter the values of a & b");
        scanf("%d%d",&a,&b);
        sum = a+b;
        return sum;
}
```

- **Functions with arguments and one return value**

  // function that takes two arguments and print their sum.

```
int add(int a,int b)
{
        return (a + b);
}
```


Function that have multiple return values

```
#include <stdio.h>
int even(num)
 {
        if( num%2==0)
                return 1;
         else
                return 0;
         }
        void main()
        {
                int s,a;
                printf("enter a");
                scanf("%d",&a);
                printf("%d",even(a));
        }
```

- **Write a program to perform arithmetic operations using function**

```
#include<stdio.h>
int add(int a,int b)
{
```

```
    return (a+b);
}
int diff(int a,int b)
{
    return (a-b);
}
int mul(int a,int b)
{
    return (a*b);
}
float div(int a,int b)
{
    return (float)a/b;
}
int mod(int a,int b)
{
    return (a%b);
}
void main()
{
    int x,y;
    printf("Enter the values:");
    scanf("%d%d",&x,&y);
    printf("Sum=%d\n",add(x,y));
    printf("Difference=%d\n",diff(x,y));
    printf("Multiply=%d\n",mul(x,y));
    printf("Division=%f\n",div(x,y));
    printf("Modulo=%d\n",mod(x,y));
}
```

OUTPUT

```
Enter the values:21 8
Sum=29
Difference=13
Multiply=168
Division=2.625000
Modulo=5
```

- **Write a program to display prime numbers upto a range using function.**

```
// function takes an argument n, if n is prime it will return 1 otherwise 0

#include<stdio.h>
#include<math.h>
int checkPrime(int n)
{
    int flag=0,i;
```

```
    for(i=2;i<=sqrt(n);i++)
    {
        if(n%i == 0)
        {
            flag =1;
            break;
        }
    }
    if(flag == 0)
        return 1;
    else
        return 0;
}
void main()
{
    int n,i;
    printf("Enter the range:");
    scanf("%d",&n);
    for(i=2;i<=n;i++)
    {
        if(checkPrime(i) == 1)
            printf("%d\t",i);
    }
}
```


OUTPUT

```
Enter the range:25
2    3    5    7    11    13    17    19    23
```

RECURSIVE FUNCTION

In some problems, it may be natural to define the problem in terms of the problem itself. Recursion is useful for problems that can be represented by a simpler version of the same problem.

Example: the factorial function

6! = 6 * 5 * 4 * 3 * 2 * 1

We could write:

6! = 6 * 5!

A function that calls itself is known as a recursive function. And, this technique is known as recursion. While using recursion, programmers need to be careful to define a termination condition from the function; otherwise it will go into an infinite loop. Termination condition will be the base case where the problem can be solved without

further recursion. Recursion makes program elegant. However, if performance is vital, use loops instead as recursion is usually much slower.

- **Write a program to find factorial of a number using recursive function**

```
#include<stdio.h>
int fact(int n)
{
    if( n == 0)
            return 1;
    else
            return n*fact(n-1);
}
void main()
{
    int n;
    printf("Enter the number:");
    scanf("%d",&n);
    printf("Factorial=%d",fact(n));
}
```

OUTPUT

```
Enter the number:5
Factorial=120
```


- **Write a program to find nCr and nPr.**

```
#include<stdio.h>
int fact(int n)
{
    if( n == 0)
            return 1;
    else
            return n*fact(n-1);
}
void main()
{
    int n,r;
    float C,P;
    printf("Enter the numbers:");
    scanf("%d%d",&n,&r);
    P = (float) fact(n) / fact(n-r);
    C = (float)fact(n) / (fact(r) * fact(n-r));
    printf("nCr=%f\n",C);
    printf("nPr=%f",P);
}
```

OUTPUT

Enter the numbers:5 3

nCr=10.000000

nPr=60.000000

- **Write a program to find the sum of series 1 + 1/2! + 1/ 3!+ ..**

```
#include<stdio.h>
int fact(int n)
{
    if( n == 0)
            return 1;
    return n*fact(n-1);
}
void main()
{
    int n,i;
    float sum=0.0;
    printf("Enter the limit:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
            sum = sum + 1.0 / fact(i);
    printf("Sum=%f",sum);
}
```

OUTPUT

Enter the limit:5

Sum=1.716667

- **Write a recursive function to print Fibonacci series**

```
#include<stdio.h>
int fib(int n)
{
    if(n == 1)
            return 0;
    else if(n == 2)
            return 1;
    return fib(n-2)+fib(n-1);
}

void main()
{
    int n,i;
```

```
    printf("Enter the limit:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
            printf("%d\t",fib(i));
}
```

OUTPUT

```
Enter the limit:7
0    1    1    2    3    5    8
```

- **Write a recursive program to print sum of digit of a number**

```
#include<stdio.h>
int sod(int n)
{
    if(n<=0)
            return 0;
    else
            return n%10 + sod(n/10);
}

void main()
{
    int n;
    printf("Enter the number:");
    scanf("%d",&n);
    printf("Sum=%d",sod(n));
}
```

OUTPUT

```
Enter the number:124
Sum=7
```

- **Write a recursive function to find the sum of first n natural numbers**

```
#include<stdio.h>
int sum(int n)
{
    if(n<=0)
            return 0;
    else
            return n + sum(n-1);
}
void main()
{
    int n;
    printf("Enter the number:");
    scanf("%d",&n);
```

```
    printf("Sum=%d",sum(n));
}
```

OUTPUT

```
Enter the number:5
Sum=15
```

PASSING AN ARRAY AS PARAMETER

Like the value of simple variables, it is also possible to pass the values of an array to a function. To pass a single dimensional array to a function, it is sufficient to list the name of the array without any subscripts and the size of the array as arguments.

Rules to pass an Array to Function

- The function must be called by passing only the name of the array.
- In function definition, formal parameter must be an array type; the size of the array does not need to be specified.
- The function prototype must show that the argument is an array.

- **Write a function to sort an array.**

```
#include<stdio.h>
int sort(int A[],int n)
{
    int i,j,temp;
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(A[j]>A[j+1])
            {
                temp  = A[j];
                A[j]  = A[j+1];
                A[j+1]= temp;
            }
        }
    }
}

void main()
{
    int A[30];
    int i,n;
```

```
    printf("Enter the limit:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
            printf("Enter the element:");
            scanf("%d",&A[i]);
    }
    sort(A,n);
    printf("Sorted Array\n");
    for(i=0;i<n;i++)
            printf("%d\t",A[i]);
}
```

OUTPUT

```
Enter the limit:5
Enter the element:17
Enter the element:23
Enter the element:5
Enter the element:2
Enter the element:9
Sorted Array
2     5     9     17     23
```


- **Write a program to sort the matrix row wise.**

```
#include<stdio.h>
int sort(int A[],int n)
{
    int i,j,temp;
    for(i=0;i<n-1;i++)
    {
            for(j=0;j<n-i-1;j++)
            {
                    if(A[j]>A[j+1])
                    {
                            temp  = A[j];
                            A[j]  = A[j+1];
                            A[j+1]= temp;
                    }
            }
    }
}

void main()
{
    int A[30][30];
```

```
    int i,j,m,n;
    printf("Enter the order of matrix:");
    scanf("%d%d",&m,&n);
    for(i=0;i<m;i++)
    {
            for(j=0;j<n;j++)
            {
                    printf("Enter the element:");
                    scanf("%d",&A[i][j]);
            }
    }
    for(i=0;i<m;i++)
    {
            sort(A[i],n);
            for(j=0;j<n;j++)
            {
                    printf("%d\t",A[i][j]);
            }
            printf("\n");
    }
}
```

OUTPUT

```
Enter the order of matrix:2 4
Enter the element:12
Enter the element:4
Enter the element:25
Enter the element:7
Enter the element:8
Enter the element:5
Enter the element:16
Enter the element:2
4     7     12     25
2     5     8     16
```

PASSING 2D ARRAY AS PARAMETER

Note: While passing 2D array as parameter we need to mention the maximum size of element of each row that is column.

- **Write a program to pass a 2D matrix as parameter and find its sum of all the elements.**

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

```
// function that takes a 2D array and its order
int sum(int A[][30],int m,int n)
{
    int i,j,sum =0;
    for(i=0;i<m;i++)
    {
            for(j=0;j<n;j++)
            {
                    sum = sum + A[i][j];
            }
    }
    printf("\nSum=%d",sum);
}

void main()
{
    int A[30][30];
    int i,n,m,j;
    printf("Enter the order of matrix:");
    scanf("%d%d",&m,&n);
    for(i=0;i<m;i++)
    {
            for(j=0;j<n;j++)
            {
                    printf("Enter the element:");
                    scanf("%d",&A[i][j]);
            }
    }
    sum(A,m,n);
}
```


OUTPUT

```
Enter the order of matrix:2 2
Enter the element:1
Enter the element:2
Enter the element:3
Enter the element:4
Sum=10
```

STRUCTURE

A structure is a user defined data type in C. A Structure is a collection of related data items, possibly of different types. A Structure is ***heterogeneous*** in that it can be composed of data of different types. In contrast, array is ***homogeneous*** since it can

contain only data of the same type.The keyword 'struct' is used to create a structure. A structure variable can either be declared with structure declaration or as a separate declaration like basic types.

Consider we want to create the structure of a person with following variables name, age and address. Then such a structure can be created as

```
struct Person
{
        char name[30];
        int age;
        char addr[50];
};
```

The general format of a structure definition is as follows

```
struct structure_name{
        data_type       member1;
        data_type       member2;
        ---------------------
        ---------------------
};
```


In defining a structure you may note the following syntax:

- The template is terminated with a semicolon.
- While the entire definition is considered as a statement, each member is declared independently for its name and type in a separate statement inside the template.

Difference between Structure and Array

Array	Structure
An array is a collection of related data elements of same type.	Structure can have elements of different types.
An array is derived data type	Structure is a programmer defined one
Any array behaves like built in data type. All we have to do is to declare an array variable and use it	In Structure we have to design and declare a data structure before the variable of that type are declared and used.

Declaring Structure Variable

After defining a structure format we can declare variables of that type. A structure variable declaration is similar to the declaration of variables of any other data type. It includes the following elements.

1. The keyword struct
2. The structure tag name(structure name)
3. List of variable names separated by commas.
4. A terminating semicolon.

Example:

```
struct Person p1,p2,p3;          // created structure variable for person Structure.
```

Accessing Structure Members

Members of a structure is accessed as structure variable followed by dot(.) operator (also called period or member access operator.)

- **Write a program to create a structure employee with member variables name, age, bs, da, hra and tsalary. Total Salary is calculated by the equation tsalary= (1+da+hra)* bs. Read the values of an employee and display it.**

```
#include<stdio.h>
struct Employee{
    char name[30];
    int age;
    float bs;
    float da;
    float hra;
    float tsalary;
};
void main()
{
    struct Employee e;
    printf("Enter the name:");
    scanf("%s",e.name);
    printf("Enter the age:");
    scanf("%d",&e.age);
    printf("Enter the basic salary:");
    scanf("%f",&e.bs);
    printf("Enter the da:");
    scanf("%f",&e.da);
    printf("Enter the hra:");
    scanf("%f",&e.hra);
    e.tsalary=(1+e.da+e.hra)*e.bs;

    printf("Name=%s\n",e.name);
```

```
    printf("Age=%d\n",e.age);
    printf("Basic Salary=%.2f\n",e.bs);
    printf("DA=%.2f\n",e.da);
    printf("HRA=%.2f\n",e.hra);
    printf("Total Salary=%.2f\n",e.tsalary);
}
```

OUTPUT

```
Enter the name:John
Enter the age:31
Enter the basic salary:10000
Enter the da:12
Enter the hra:7.5
Name=John
Age=31
Basic Salary=10000.00
DA=12.00
HRA=7.50
Total Salary=205000.00
```

- **Write a program to create a structure Complex with member variables real and img. Perform addition of two complex numbers using structure variables.**


```
#include<stdio.h>
struct Complex
{
    int real;
    int img;
};

void main()
{
    struct Complex a,b,c;
    printf("Enter the real and img part of a:");
    scanf("%d%d",&a.real,&a.img);
    printf("Enter the real and img part of b:");
    scanf("%d%d",&b.real,&b.img);
    c.real = a.real + b.real;
    c.img = a.img + b.img;
    printf("c = %d + %di\n",c.real,c.img);
}
```

OUTPUT

```
Enter the real and img part of a:10 20
```

```
Enter the real and img part of b:30 40
c = 40 + 60i
```

ARRAY OF STRUCTURES

Structure is used to store the information of one particular object but if we need to store the information of many such objects then Array of Structure is used.

```
Example :
struct Bookinfo
{
        char[20] bname;
        int pages;
        int price;
}Book[100];
```

- **Declare a structure namely Student to store the details (roll number, name, mark_for_C) of a student. Then, write a program in C to find the average mark obtained by the students in a class for the subject Programming in C (using the field mark_for_C). Use array of structures to store the required data.**

```
#include<stdio.h>
struct Student{
    char name[30];
    int rollnum;
    int mark_for_C;
};

void main(){
    struct Students[30];
    int i,n,sum=0;
    float avg;
    printf("Enter the no of Student:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
            printf("Enter the Student name:");
            scanf("%s",s[i].name);
            printf("Enter the Student rollnum:");
            scanf("%d",&s[i].rollnum);
            printf("Enter the Student Mark for C:");
            scanf("%d",&s[i].mark_for_C);
    }
    printf("Name\tRoll Number\tMark for C\n");
    for(i=0;i<n;i++)
    {
```

```
            printf("%s\t%d\t%d\n",s[i].name,s[i].rollnum,s[i].mark_for_C);
            sum = sum + s[i].mark_for_C;
    }
    avg = (float)sum / n;
    printf("Average Mark=%.2f\n",avg);
}
```

OUTPUT

```
Enter the no of Student:3
Enter the Student name:John
Enter the Student rollnum:27
Enter the Student Mark for C:35
Enter the Student name:Miya
Enter the Student rollnum:24
Enter the Student Mark for C:40
Enter the Student name:Anu
Enter the Student rollnum:26
Enter the Student Mark for C:45
Name    Roll Number    Mark for C
John        27              35
Miya        24              40
Anu         26              45
Average Mark=40.00
```


- **Write a program to create a structure employee with member variables name, age, bs, da, hra and tsalary. Total Salary is calculated by the equation tsalary= (1+da+hra)* bs. Read the values of 3 employees and display details based on descending order of tsalary.**

```
#include<stdio.h>
struct Employee{
    char name[30];
    int age;
    float bs;
    float da;
    float hra;
    float tsalary;
};
void sort(struct Employee e[],int n)
{
    int i,j;
    struct Employee t;
    for(i=0;i<n-1;i++)
    {
            for(j=0;j<n-i-1;j++)
            {
```

```
                    if(e[j].tsalary < e[j+1].tsalary)
                    {
                        t = e[j];
                        e[j]=e[j+1];
                        e[j+1]=t;
                    }
                }
            }
        }
        void main()
        {
            struct Employee e[5];
            int i;
            for(i=0;i<3;i++)
            {
                printf("Enter the name:");
                scanf("%s",e[i].name);
                printf("Enter the age:");
                scanf("%d",&e[i].age);
                printf("Enter the basic salary:");
                scanf("%f",&e[i].bs);
                printf("Enter the da:");
                scanf("%f",&e[i].da);
                printf("Enter the hra:");
                scanf("%f",&e[i].hra);
                e[i].tsalary=(1+e[i].da+e[i].hra)*e[i].bs;
            }
            sort(e,3);
            printf("Name\t    Age\tBasic Salary\tDA    \t HRA    \t Total Salary\n");
            for(i=0;i<3;i++)
            {
                printf("%s\t%d\t%.2f\t",e[i].name,e[i].age,e[i].bs);
                printf("%.2f\t%.2f\t%.2f\n",e[i].da,e[i].hra,e[i].tsalary);
            }

        }
```

OUTPUT

```
Enter the name:John
Enter the age:31
Enter the basic salary:14000
Enter the da:6
Enter the hra:7.5
Enter the name:Miya
Enter the age:28
Enter the basic salary:15000
```

```
Enter the da:7.6
Enter the hra:8
Enter the name:Anu
Enter the age:29
Enter the basic salary:15000
Enter the da:8
Enter the hra:9

Name      Age    Basic Salary   DA     HRA    Total Salary
Anu       29     15000.00       8.00   9.00   270000.00
Miya      28     15000.00       7.60   8.00   249000.00
John      31     14000.00       6.00   7.50   203000.00
```

UNION

A union is a user-defined type similar to structure in C programming. We use the keyword 'union' to define unions. When a union is defined, it creates a user-defined type. However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.

Example of Employee Union creation and declaration


```
union Employee
{
    char name[30];
    int age;
    double salary;
} ;

union Employee e;
```

Differences between structure and union

Structure	Union
struct keyword is used to define a structure	**union** keyword is used to define a union
Members do not share memory in a structure.	Members share the memory space in a union
Any member can be retrieved at any time in a structure	Only one member can be accessed at a time in a union.
Several members of a structure can be initialized at once.	Only the first member can be initialized.

Size of the structure is equal to the sum of size of the each member.	Size of the union is equal to the size of the largest member.

Predict the output

```
#include<stdio.h>
struct Person
{
    char pincode[6];                // Size = 6 bytes
    int age;                        // Size = 4 bytes
    double salary;                  // Size = 8 bytes
};
union Employee
{
    char pincode[6];
    int age;
    double salary;
};
void main()
{
    struct Person p;
    union Employee e;
    printf("Size of Structure Person=%d\n",sizeof(p));
    printf("Size of Union Employee=%d",sizeof(e));
}
```


OUTPUT

```
Size of Structure Person=18
Size of Union Employee=8
```

SCOPE, VISIBILITY AND LIFETIME OF A VARIABLE

In C, not only do all the variables have a data type, they also have a storage class. A storage class is used to represent additional information about a variable.

The storage class of a variable in C determines

- variable's storage location (memory or registers)
- The default initial value of the variable
- The scope (visibility level) of the variable
- Life time of the variable-how long the variable exists .

The following storage classes are most relevant in C

- Automatic

- External or Global
- Static
- Register

AUTOMATIC VARIABLES

Automatic variables are declared inside a function in which they are to be utilized. They are created when the function is called and destroyed automatically when the function is exited, hence the name automatic. Automatic variables are therefore private or local to the function in which they are declared. Because of this property, automatic variables are also referred to as local or internal variables.

- ✓ Storage location –main Memory
- ✓ Default initial value – An unpredictable value, which is often called a garbage value.
- ✓ Scope – Local to the block in which the variable is defined.
- ✓ Life – Till the control remains within the block in which the variable is defined.
- ✓ Keyword auto is used to declare these variables

Example:

auto int i;

Note: A variable declared inside a function without storage class specification is by default an automatic variable.

```
void main(){
        int num;
        ----------
        ----------
}
```

is same as

```
void main(){
        auto int num;
        ----------
        ----------
}
```

Example

#include<stdio.h>

```
void func1()
{
        int  max=10;
        printf("Max in func1()=%d\n",max);
}
void func2()
{
        int  max=20;
        printf("Max in func2()=%d\n",max);
}
void main()
{
        int max=30;
        func1();
        func2();
        printf("Max in main()=%d\n",max);
}
```

Output

```
Max in func1()=10
Max in func2()=20
Max in main()=30
```


REGISTER VARIABLES

We can tell the compiler that a variable should be kept in one of the machine's registers instead of keeping in the memory. Since a register access is faster than a memory access, keeping the frequently accessed variables in the register will lead to faster execution of programs. This is done as follows

register int count;

Since only a few variables can be placed in the register, it is important to carefully select the variables for these purposes. However C will automatically convert register variables into non register variable once limit is reached.

- ✓ Storage - CPU registers.
- ✓ Default initial value - Garbage value.
- ✓ Scope - Local to the block in which the variable is defined.
- ✓ Life - Till the control remains within the block in which the variable is defined.
- ✓ Keyword register is used to declare these variables

Example:

```
register int i;
```

Example

```
void main( )
{
        register int i ;
        for ( i = 1 ; i <= 10 ; i++ )
                printf ( "\n%d", i ) ;
}
```

Here, even though we have declared the storage class of i as register, we cannot say for sure that the value of i would be stored in a CPU register. Because the number of CPU registers are limited, and they may be busy doing some other task. In such situations, the variable works as if its storage class is auto.

STATIC VARIABLES

As the name suggests, the value of static variables persists until the end of the program. A variable can be declared static using the keyword **static** like

static *int x;*

- ✓ Storage –main Memory.
- ✓ Default initial value – Zero.
- ✓ Scope – Local to the block in which the variable is defined.
- ✓ Life – Value of the variable persists between different function calls. ie they retain the latest value
- ✓ Keyword static is used to declare these variables

Example:

```
static int i;
```

A static variable tells the compiler to persist the variable until the end of the program. That is , they retain the latest value. Instead of creating and destroying a variable every time when it comes into and goes out of scope, static is initialized only once and remains into existence till the end of program.

```
#include<stdio.h>

void func1()
{
        static int  x=10; //static variable
        x++;
        printf("x in func1()=%d\n",x);
}
```

```
void func2()
{
        int x=10;       // local variable
        x++;
        printf("x in func2()=%d\n",x);
}
void main(){
        func1();
        func1();
        func2();
        func2();
}
```

OUTPUT

```
x in func1()=11
x in func1()=12
x in func2()=11
x in func2()=11
```

EXTERNAL VARIABLES

Variables that are both alive and active throughout the entire program are known as external variables. They are called global variables. Unlike local variables, global variables can be accessed by any function in the program. External variables are declared outside a function.

- ✓ Storage –main Memory(RAM).
- ✓ Default initial value – Zero.
- ✓ Scope – Global.
- ✓ Life – till the end of program.
- ✓ Keyword extern is used to declare these variables

Example: extern int i;

```
#include<stdio.h>
float pi=3.14;          // One way of declaring external variable

float area(int r)
{
        return (pi*r*r);
}
float perimeter(int r){
        return (2 * pi * r);
}
```

```
void main()
{
        // extern float pi=3.14;  Another way of declaring external variable
        int r;
        float a,p;
        printf("Enter the radius:");
        scanf("%d",&r);
        a=area(r);
        p=perimeter(r);
        printf("Area=%f\n",a);
        printf("Perimeter=%f\n",p);
}
```

OUTPUT

```
Enter the radius:5
Area=78.500000
Perimeter=31.400002
```

More example to show the property of global variable.

```
#include<stdio.h>
int max; // global variable
void func1()
{
        int  max=10; // local variable
        printf("Max in func1()=%d\n",max);
}
void func2()
{
        max=20; // resets max value to 20
        printf("Max in func1()=%d\n",max);
}
void main()
{
        max=40; //set max value to 40
        func1();
        func2();
        printf("Max in main()=%d\n",max);
}
```


OUTPUT

```
Max in func1()=10
Max in func1()=20
Max in main()=20
```

Why the value of max in main() print as 20? [Hint: main uses global scope of max.]

Properties / Storage Class	Storage	Default Initial Value	Scope	Life
Automatic	Memory	Garbage Value	Local to the block in which the variable is defined	Till the control remains within the block in which the variable is defined
Register	CPU Registers	Garbage Value	Local to the block in which the variable is defined	Till the control remains within the block in which the variable is defined
Static	Memory	Zero	Local to the block in which the variable is defined	Value of the variable continues to exist between different function calls
External	Memory	Zero	Global	Till the program's execution doesn't come to an end

PREVIOUS YEAR UNIVERSITY QUESTIONS

1. What are the advantages of using functions in a program? ***[KTU, MODEL 2020]***
2. With a simple example program, explain scope and life time of variables in C. ***[KTU, MODEL 2020]***
3. Write a function namely myFact in C to find the factorial of a given number. Also, write another function in C namely nCr which accepts two positive integer parameters n and r and returns the value of the mathematical function C(n,r)(n! / (r! x (n - r)!)). The function nCr is expected to make use of the factorial function myFact. ***[KTU, MODEL 2020]***
4. What is recursion? Give an example. ***[KTU, MODEL 2020], [KTU, JULY 2017]***
5. With a suitable example, explain the differences between a structure and a union in C. ***[KTU, MODEL 2020], [KTU, MAY 2017], [KTU, JULY 2017], [KTU, APRIL 2018], [KTU, JULY 2018]***
6. Declare a structure namely Student to store the details (roll number, name, mark_for_C) of a student. Then, write a program in C to find the average mark obtained by the students in a class for the subject Programming in C (using the field mark_for_C). Use array of structures to store the required data. ***[KTU, MODEL 2020]***

7. Write a function in C which takes a 2-Dimensional array storing a matrix of numbers and the order of the matrix (number of rows and columns) as arguments and displays the sum of the elements stored in each row. ***[KTU, MODEL 2020]***
8. **Describe the output generated by the following program**

```
#include<stdio.h>
int a=3;
fun(int x)
{
        a+= x;
        return(a);
}
void main( )
{
        int i;
        for(i=1;i<=6;++i)
        {
                a=fun(i);
                printf("%d",a);
        }
}
```

[KTU, DECEMBER 2019]

9. Write a C program using function to find the decimal equivalent of a binary number. ***[KTU, DECEMBER 2019***]
10. Define recursion with an example. Differentiate between iteration and recursion. ***[KTU, DECEMBER 2019]***
11. Write a C program to find the GCD of two numbers using recursive function. ***[KTU, DECEMBER 2019]***
12. Write the output of the program

```
#include<stdio.h>
#define prod(a ,b) a*b
int main( )
{
   int x=3,y=4;
   printf("%d",prod(x+2,y-1));
   printf("%d",prod(y+1,x-2));
   return 0;
}
```

[KTU, MAY 2019]

13. A student database stores following information about students in a class: Rollno, name, gender, CGPA. Write a program to prepare a rank list based on CGPA Also prepare a list of students having CGPA less than 7. ***[KTU, MAY 2017]***
14. A library database maintains following information about books:- book_id, name, author, no_of_copies. Write a program to sort the books based on the decreasing order of number of copies available. ***[KTU, JULY 2017]***
15. Write the output of the program. Justify the answer

```
#include<stdio.h>
int fun( )
{
```

```
    static int count=0;
    count++;
return count;
}
int main( )
{
    printf("%d", fun( ));
    printf("%d", fun( ));
    return 0;
}
```

[KTU, MAY 2019]

16. What is a structure? How is a structure member accessed? Explain with an example. ***[KTU, MAY 2019]***
17. What do you meant by scope of a variable in C? ***[KTU, MAY 2019], [KTU, MAY 2017]***
18. Explain the purpose of typedef construct. ***[KTU, MAY 2019], [KTU, MAY 2017]***
19. Explain the meaning of each of the following function prototypes.
    i) int f1 (int a);
    ii) double f2(double a, int b); ***[KTU, MAY 2019]***
20. Give the syntax and use of external storage class. ***[KTU, MAY 2019]***
21. Differentiate static and automatic variables. ***[KTU, MAY 2019]***
22. Write a C program to find largest and smallest element in an integer array using function. ***[KTU,DECEMBER 2018]***
23. Write a C program to create a structure student with detail rollno, name, marks and grade. Read the details of n students and display the details of the student if student name is given as input. ***[KTU,DECEMBER 2018***]
24. What is the use of a function prototype? Give the function prototype of a function accepting one float value and an integer array and return a float array. ***[KTU, MAY 2017]***
25. Discuss the difference between call by value and call by reference parameter passing techniques with the help of suitable examples. ***[KTU, MAY 2019], [KTU, MAY 2017], [KTU, JULY 2017], [KTU, APRIL 2018], [KTU, JULY 2018]***
26. Write a recursive function to perform binary search on a set of sorted numbers. ***[KTU, MAY 2017]***
27. What are library functions? ***[KTU, MAY 2017]***
28. What are formal arguments and actual arguments in a function? ***[KTU, JULY 2017], [KTU, JULY 2018]***
29. What are function prototypes? Why do we use function prototypes? ***[KTU, JULY 2017]***
30. How is an array name interpreted when it is passed to a function? ***[KTU, JULY 2017]***
31. Write a C program to find the length of a given string recursively, without using any standard string library function. ***[KTU, APRIL 2018]***
32. What are function prototypes? Is Function prototype mandatory for every user defined function in C? Justify your answer. ***[KTU, APRIL 2018]***
33. Write a C program to print Fibonacci series using recursion. ***[KTU, DECEMBER 2018]***

34. List out main components which comprise a function definition. ***[KTU, JULY 2018]***
35. Write a function in C to check whether the given number is prime or not. Call the function from main function. ***[KTU, JULY 2018]***
36. Consider the following function declarations
    void display(int);
    int display (int);
37. Differentiate between the two given declarations ***[KTU, JULY 2018]***
38. Write a recursive function with an integer parameter n to return the factorial of n ***[KTU, JULY 2018],***
39. Explain static storage class with the help of example. ***[KTU, MAY 2017], [KTU, DECEMBER 2019]***
40. Explain register storage class with the help of an example. ***[KTU, JULY 2017], [KTU, DECEMBER 2018]***
41. How does array differ from a structure? ***[KTU, DECEMBER 2018], [KTU, JULY 2018]***
42. What are function prototypes and what is its purpose? ***[KTU, APRIL 2018]***
43. Write a program to read an array of integer numbers and display its mean and standard deviation. Note: Computation of mean and standard deviation needs to be performed in a separate function. ***[KTU, APRIL 2018]***
44. Develop a recursive C program to print first N Fibonacci terms. ***[KTU, JUNE 2017]***
45. Write a function big to find largest of two numbers and use this function in the main program to find largest of three numbers. ***[KTU, DECEMBER 2017]***
46. Create a structure for an employee with following information:- empid, name and salary. Write a program to read the details of 'n' employees and display the details of employees whose salary is above 10000. Use pointer to structure. [***KTU, DECEMBER 2019]***