MODULE 1

sequential, conditional, unconditional and control structures in C

In a C program, the flow of execution is **sequential** by default, that is, one statement after another in their order as in the source code of a function.

With various **control structures**, the order of execution can be different from the sequential order.

There are two major types of non-sequential control structures: **selection statements** and **loops**

Selection statements in C

Simple If
If else
If else if ladder
Switch

Study the syntax for all the above.

All the above are also called **conditional control structures** since the flow of control is decided based on the condition which is either true or false.

Unconditional control structure in C

A special statement ***goto*** is an unconditional control statement in C
syntax

Goto label;

label:

Loops in C

Three types of loops in C
While loop, do while loop and for loop

Sturdy the syntax for each of them

Concept of preprocessing

A C Preprocessor is just a text substitution tool and it instructs the compiler to do required pre-processing before the actual compilation.

All preprocessor commands begin with a hash symbol (#).

Eg: # include <stdio.h>

This will include the all the input , output function definitions to the current program

Other preprocessors are

Sr.No.	Directive & Description
1	#define Substitutes a preprocessor macro.
2	#include Inserts a particular header from another file.
3	#undef Undefines a preprocessor macro.
4	#ifdef Returns true if this macro is defined.
5	#ifndef Returns true if this macro is not defined.
6	#if Tests if a compile time condition is true.

Modular programming concepts in C

Modularization is a method to organise large programs in smaller parts, i.e. the modules

Every module has an implementation part that hides the code and any other private implementation detail. C is a language that lends itself readily to modular programming. Because the functions of a C program can be compiled separately,all we have to do is to put each function, or group of related functions, in its own file. Each file can then be treated as a component, or a module, of your program.

Develop programs using functions

Functions in C

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main** (), and all the most trivial programs can define additional functions. The main advantage of function is that they make the programming simple. Once a function is declared and defined, the user can call the same function with different parameters and hence making the programming simple.

There are basically two types of functions in C.

1. Standard Library functions or built in functions
   These functions are provided by the C language itself. Eg: clrscr(), printf(), scanf(),getch(),strlen(), strcpy() etc….
2. User Defined Functions
   User can define their own functions.

There are basically three parts for a user defined function.

1) Function declaration (Function prototype)
   A function declaration tells the computer about a function's name, return type, and no of parameters and their type. Usually we declare function at the top of the main programme.
   Eg: int sum (int, int);

2).Function calling

   Usually we make the function call form the main programme.

   Eg: s=sum(2,3)

   The parameters which are provided in the function calling statements are called actual parameters, here 2 and 3. When the function is called the programme control is transferred to the function definition.

3. Function Defenition.

   A function definition provides the actual body of the function. That is it contains the code for what the function should do. The parameters which occur in the function definition is known as formal parameters.

   Eg: int sum (int a, int b)

   {
   int sum;
   sum=a+b
   return (sum);
   }

And whenever a function call occurs, the values of the actual parameters are copied to the formal parameters. Here a and be are the formal parameters and when the function sum(2,3) is called, a gets the value of and b gets the value of 3.

This type of function calling is known as **call by value method**, where the value of actual parameters gets copied to formal parameters. And **whatever the changes that we make to the formal parameters are not going to reflected back to the actual parameters.** Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. In the above example variables sum, a and be are local variables and not visible from any other functions, including main function.

When the codes in the function definition are completed the program control is transferred back to the main programme.

There are two types of function calling mechanism in C.

Call by value and call by reference.

The call by value method is already discussed above. In call by reference method instead of passing the value, the reference or the address of the variable is passed. And this address is copied to the formal parameters. So the formal parameters should be pointers so that they can store the addresses. The main concept behind the call by reference method is that since the address of the variable is passed instead of value, whatever the changes that the function makes at this address, will be visible from any functions.

Eg:

```
#include<stdio.h>
void swap(int *x, int *y);    ## function decleration
 void  main () {
   int a = 100;
   int b = 200;
   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );
   swap(&a, &b);   ## function calling, &a-> address of a
   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );
}
```

```
 void swap(int *x, int *y) ## function definition , x and y are pointer variables

{
```

```
   int temp;
   temp = *x;    ## * operator gets the value saved at a particular address
   *x = *y;      ## put y into x
   *y = temp;    ## put temp into y

   return;
}
```

Output

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :200
After swap, value of b :100
```

Recursion with examples

A function that calls itself is known as a recursive function. And, this technique is known as recursion

Example: Factorial of a Number Using Recursion

```
#include <stdio.h>
int factorial(int n);
void  main()
{
    int n;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    printf("Factorial=%d",factorial(n));
    }
int factorial(int n)
{
    if (n >= 1)
        return n*factorial(n-1);## recursive call,
    else
        return 1;
}
```

The above program works in the following way.
factorial(3)
return (3*factorial(2)) ##since 3>1, return n * factorial of(n-1)= return 3* factorial(2)
return (3* return(2*factorial(1)))
return(3*return(2*1))
return(3*2)
=6

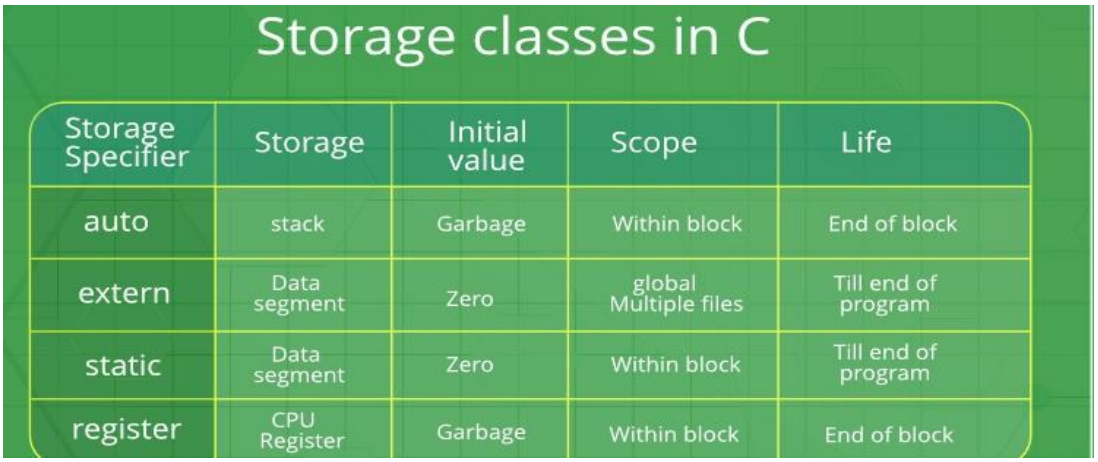
Explain Scope,Visibility and Life of Variables

Scope
scope is basically the region of code in which a variable is available to use.
Eg: local variables have availability inside the function definition only, hence local variables scope is only inside the function.
There are four types of scope: file scope, block scope, function scope and prototype scope.

Visibility
Visibility and scope are similar concepts.
Visibility of a variable is defined as if a variable is accessible or not inside a particular region of code. Scope is the region of code where a variable is accessible, visibility is a yes or no question that whether the variable is available or not.

Lifetime of a variable is the time for which the variable is taking up a valid space in the system's memory,
It is of three types:static lifetime, automatic lifetime and dynamic lifetime.

Storage Classes in C
Based on the scope, visibility , lifetime etc the variables in c can be of the following four storage classes.



.

MODULE 2

Summarize the definition, initialization and accessing of single and multi dimensional arrays.

ARRAY

An array is a collection of data that holds fixed number of values of same type. For example: if you want to store marks of 100 students, you can create an array for it.

```
float marks[100];
```

The above marks array can store a maximum of 100 elements of the type int

The size and type of arrays cannot be changed after its declaration, trying to read/write an element from the array beyond its limits causes errors in the program (Array out of bound errors)

Arrays are of two types:

1. One-dimensional arrays
2. Multidimensional arrays

Elements of an Array and How to access them?

You can access elements of an array by indices.

Suppose you declared an array `mark` as above. The first element is `mark[0]`, second element is `mark[1]` and so on.

Few key notes:

- Arrays have 0 as the first index not 1. In this example, `mark[0]`
- If the size of an array is `n`, to access the last element, `(n-1)` index is used. In this example, `mark[4]`

How to insert and print array elements?

```
int mark[5] = {19, 10, 8, 17, 9}

// insert different value to third element
mark[3] = 9;

// take input from the user and insert in third element
scanf("%d", &mark[2]);
```

```
•
• // print first element of an array
• printf("%d", mark[0]);
•
• // print ith element of an array
• printf("%d", mark[i-1]);
```

Basic Array operations.

Array insertion

Deleting an element from array

Array searching

Array sorting

Study the programs for array operations, which we have done in the lab,

Also study the programs for passing arrays to functions, ie array operation using function.

Matrix is an example for 2 dimensional array.

Eg : int A[10]1[10]

This array can store a maximum of 100 elements and the array index varies from A[0][0] to A[99][99]

Study the various matrix operations that we have done in the lab.

Read and print a matrix

Row sum of matrix

Colum sum of matrix

Sum of diagonal elements of a matrix

Transpose of a matrix

Matrix Addition

Matrix multiplication

illustrate the concept of divide and conquer method in solving problems

In computer science, divide and conquer is an algorithm design paradigm. A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly.

One should think of a divide-and-conquer algorithm as having three parts:
1. Divide the problem into a number of subproblems that are smaller instances of the same problem.
2. Conquer the subproblems by solving them recursively.
3. Combine the solutions to the subproblems into the solution for the original problem.

LInear Search,Binary search, selection sort, Quick sort

Study the programs that we have done in the lab. And the assignment programs

Strings in C

In C programming, array of characters is called a string. A string is terminated by a null character '/0'

Declaration of strings

Before we actually work with strings, we need to declare them first.

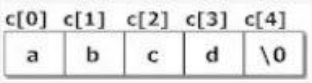
Strings are declared in a similar manner as arrays. Only difference is that, strings are of char type.

Eg: `char s[5];`

Initialization of strings

In C, string can be initialized in a number of different ways.

```
char c[] = "abcd";

     OR,

char c[50] = "abcd";

     OR,

char c[] = {'a', 'b', 'c', 'd', '\0'};

     OR,

char c[5] = {'a', 'b', 'c', 'd', '\0'};

OR

Char c[5];

C=”abcd”
```

And the array will be saved as 

Reading the Strings:

```
char c[20];

scanf("%s", c);
```

or

```
#include<stdio.h>
int main()
{
char name[30];
printf("Enter name: ");
gets(name);//Function to read string from user.
printf("Name: ");
puts(name);//Function to display string.
return0;
}
```

String Handling Functions in C

Function	Work of Function
strlen()	Calculates the length of string
strcpy()	Copies a string to another string
strcat()	Concatenates(joins) two strings
strcmp()	Compares two string
strlwr()	Converts string to lowercase
strupr()	Converts string to uppercase

Study the program for each of the user defined string handling functions that we have written in record.

illustrate passing arrays as parameters to a function

Refer Lab Record : readArray(), printArray() programs

MODULE 3 : POINTERS

POINTER

A pointer is a variable which contains the address in memory of another variable. We can have a pointer to any variable type.

The ***address of operator,*** **&** gives the ``address of a variable''.

The ***indirection*** or dereference operator * gives the ``contents of an object ***pointed to*** by a pointer''.

We can declare a pointer to an integer variable as

```
int a=5;
int  *p;
p=&a
```

Now the pointer variable p is pointing to address of the integer variable a.

We can print the value of p as ***printf("%x",p)*** . If ***a*** was at the address 1000H, it will print 1000H. (we use %x to denote that the p stores an address(Hexadecimal))

Now applying the indirection operator *, we can get the content of the address pointed by p. Hence ***printf("%d",*p)*** will be printing 5, the value of a.

if we write *p=6, then now the value of a will be changed to 6.

Pointer Arithmetic

The following are valid pointer arithmetic operations

1. Addition of integer to a pointer

p=p+1 will increment the value of p by one. Hence if p was an integer pointer and p was initially pointing to 1000H, p=p+1 or ++p or p++ will change the value of p to 1002 H, since the size of an integer is 2 bytes.

if p was a pointer to a floating point value, then ++p will make the address to be incremented by 4 bytes. (ie if initiall p was 1000H, the p++ will make p to point to 1004 H)

if p was a pointer to a character variable then address will be incremented by 1 byte

2. Subtraction of integer to a pointer

Decrementing the pointer values also will be done in the same fashion.

3. Subtracting one pointer from another of the same type

If we have two pointers p1 and p2 of base type pointer to int with addresses 1002 and 1000 respectively, then p2 - p1 will give 1, since the size of int type is 2 bytes. If you subtract p2 from p1 i.e p1 - p2 then answer will be negative i.e -1.

4. comparision of pointers

we can compare two pointers using relational operator. You can perform six different type of pointer comparison <, >, <=, >=, == and !

This is known as pointer arithmetic or address arithmetic

Relation between pointers and arrays

Pointers and arrays in C are closely related.

The basic idea is that any array name in C is a pointer to its base address.

Lets consider the integer array a

int a[10];

by definition a=a[0] (Array name is the pointer to its base address (starting address))

Hence a+1 will point to a[1], a+2 will point to a[2] and so on.

*a(a+1) will get the value at a[1]

So we can write *(a+1) in place of a[1]

Therefore any array operations can be done using pointers as described below.

Write the program for array operation using pointers, that we have done in lab.

Also when passing array to function, we are passing array name, since array name is a pointer, passing array to function uses pass by reference method, not pass by value.

***** Note: you can add increment (++) and decrement(--) operator also in valid pointer arithmetics**

Advantages of passing pointers to function

1) When we are passing pointers to functions, the address gets passed, hence any change that we make on the formal parameters are also going to be affected on the actual parameters, which is not possible in the case normal function calls where we pass the parameter values
2) In general a function can return only one value, here by making use of pointers we can make multiple values returned back to the code where the function is called.
3) Pointers can be used to pass information back and forth between the calling function and the called function.

Explain dynamic memory allocation concepts in C

The concept of dynamic memory allocation in c language enables the C programmer to allocate memory at runtime. Dynamic memory allocation in c language is possible by 4 functions listed below which are part of the <stdlib.h> header file.

malloc()	allocates single block of requested memory.
calloc()	allocates multiple block of requested memory.
realloc()	reallocates the memory occupied by malloc() or calloc() functions.
free()	frees the dynamically allocated memory.

Develop programs for single and multi-dimensional arrays using pointers.

Refer the reading and printing of arrays using pointer program from record

MODULE 4: STRUCTURES

Structure is a user-defined datatype in C language which allows us to combine data of different types together. Structure helps to construct a complex data type which is more meaningful. It is somewhat similar to an Array, but an array holds data of similar type only. But structure on the other hand, can store data of any type, which is practical more useful.

Defining a structure
struct keyword is used to define a structure. struct defines a new data type which is a collection of primary and derived data types.

Syntax:
Syntax

```
struct structurename
{
    //structure members

} variables;
```

Example

```
struct car
{
      char  name[100];
      float price;
} car1;
```

We can also declare many variables using comma (,) like below,

Example

```
struct car
{
      char  name[100];
      float price;
} car1, car2, car3;
```

We can also declare structure variables using the keyword struct as shown below

```
struct car
{
      char  name[100];
      float price;
};
struct car car1;
```

Initializing structure members and accessing it

We can initialize the structrue members directly like below,

Example

```
struct car
{
	char  name[100];
	float price;
};

struct car car1 ={"xyz", 987432.50};
```

How to access the structure members?

To access structure members, we have to use the dot (.) operator. It is also called the member access operator.

To access the price of car1,

```
car1.price
```

To access the name of car1,

```
car1.name
```

****For the rest of the topics related to structure refer to the structure program that I sent in the group. It is very important**

M4.02 Develop programs using structure to solve problems

M4.03 Illustrate the array of structure with examples

M4.04 Illustrate passing of structure as parameters to a function.

M4.05 Utilize pointers to process structure data type.

All the above portions are given in the structure program that you have did in the lab. Study the program

Union Data type

A union is a user-defined type similar to structs in C except for one key difference. Structures allocate enough space to store all their members, whereas unions can only hold one member value at a time.

How to define a union?

We use the union keyword to define unions. Here's an example:

```
union car
{
  char name[50];
  int price;
};
```

The above code defines a derived type union car.

Here's how we create union variables.

```
union car
{
  char name[50];
  int price;
};
void main()
{
  union car car1, car2, car3;
  }
```

Another way of creating union variables is:

```
union car
{
```

```
  char name[50];
  int price;
} car1, car2, car3;
```

Example: Accessing Union Members

```
#include <stdio.h>
union Job {
   float salary;
   int workerNo;
} j;
void main() {
   j.salary = 12.3;
   // when j.workerNo is assigned a value,
   // j.salary will no longer hold 12.3
   j.workerNo = 100;
   printf("Salary = %.1f\n", j.salary);
   printf("Number of workers = %d", j.workerNo);
}
```

As we said above unlike structure only one union member can hold value at a time, hence the output of the above program will be

```
Salary = 0.0
Number of workers = 100
```

Note: value of salary is set to zero when workNo was assigned a value

Memory allocated for structure and union

Size of the memory allocated for structure will be the sum of all its member elements. If the above union **Job was a structure** total memory allocated would be size of salary (4 since it is floating point)+size of workerNo(2 since it is int)=**6**, but since it is a union , the size of the union will be the maximum size of any of its members, which is **4**

Enumeration (or enum) in C

Enumeration (or enum) is a user defined data type in C. It is mainly **used to assign names to integer constants**, the names make a program easy to read and maintain.

```
Syntax:

enum week{Mon, Tue, Wed,Thur,Fri,Sat,Sun};
```

The above statement assigns the values Mon=0,Tue=1,Wed=2 by default (start with the value 0, then increment by one for the rest of the members)

Now u can use the enumerated data types for printing the day numbers as

```
void main(){
      for(int i=Mon;i<=Sun; i++){
      printf("%d, ",i);
  }
```

This will output 0,1,2,3,4,5,6

Variables of type enum can also be defined just as we did in structure and union

Eg: `enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};`

enum week day;

Here the enumerated variable is **day**, which can be assigned any of the values such as Mon, Tue, Wed, Thur, Fri, Sat, Sun etc, by default Mon gets the value 0.

```
Program

enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};

void main()

{

    enum week day;

    day = Wed;

    printf("%d",day);

    }
```

The out will be the integer corresponding to the day Wed, which is **2**

I have skipped two portions from last module

Which are file operations and command line arguments

If u study all other things from module 4 , you may omit file operation and command line argument portion

U can find the documents

For File operations click here

For command line arguments click here