SYLLABUS

UNIT-I: INTRODUCTION TO C LANGUAGE, VARIABLES, DATA TYPES AND OPERATORS

Introduction:

- Types of Languages
- History of C language
- Basic Structure
- Programming Rules
- Flow charts & Algorithms
- Commonly used library functions
- Executing the C Program
- Pre-processors in "C"
- Keywords & Identifiers
- Constants

Variables:

- Rules for defining variables
- Scope and Life of a Variable–

Data types

- Type Conversion
- Formatted Input and Output operations.

Operators: Introduction

- Arithmetic
- Relational
- Logical
- Assignment
- Conditional
- Special - Bitwise
- Increment / Decrement operator.

Q1. What is a computer language? Explain Types of Languages with example.

A) When we want to make communication between user and computer or between two or more computers, we need a language through which user can give information to computer and vice versa. When user wants to give any instruction to the computer the user needs a specific language and that language is known as computer language. User interacts with the computer using programs and that programs are created using computer programming languages like C, C++, Java etc.,

Computer languages have been evolved from Low Level to High Level Languages. In the earliest days of computers, only Binary Language was used to write programs. The computer languages are classified as follows.

1) Low level language
2) Middle level language
3) High level language

i) Low Level Language (Machine Language)

Low level language is considered as the First-Generation Language (1GL). Binary Language is an example of low-level language. The binary language contains only two symbols 1 & 0. All the instructions of binary language are written in the form of binary numbers 1's & 0's.

ii) Middle Level Language (Assembly language)

1. A middle level language is one which is closer to machine (computer) as well as to human (programmer).
2. A language that has the features of both low level and high-level languages.
3. A high level language that allows to write low level programs in it is called as middle level language.

This level language uses English like words to represent individual operations. For example: ADD, MOV, SUB,

A translator first converts the assembly language program into machine language program. Translator used with assembly language is called Assembler

iii) High Level Language

1) High level language is a computer language which can be understood by the users.
2) High level language is very similar to the human languages and have a set of grammar rules that are used to make instructions more easily.

3) Every high-level language has a set of predefined words known as Keywords and a set of rules known as Syntax to create instructions.
4)High level language needs to be converted into low level language to make it understandable by the computer. using converter like compiler and interpreter
Example C,C++, Java, Python

Q2) What is translator? Explain Types of translator?

A) Computers only understand machine code (binary), this is an issue because programmers prefer to use a variety of high and low-level programming languages instead.
To get around the issue, the high-level and low-level program code (source code) needs to pass through a translator. A translator will convert the source code into machine code (object code).

Language translator: We have 3 types of translators: 1) Assembler
2) Compiler 3) interpreter to convert high level language to low level language.

A **compiler** takes entire program and converts it into object code which is typically stored in a file. The object code is also refereed as binary code and can be directly executed by the machine after linking. Examples of compiled programming languages are C and C++.
An **Interpreter** directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code. Examples of interpreted languages are Perl, Python and MATLAB.
An **assembler** is a program for converting instructions written in low-level symbolic code into machine code.

Languages like COBOL, FORTRAN, BASIC, C, C++ JAVA etc., are the examples of high-level languages. These instructions are converted to low level language by the compiler so that it can be understood by the computer.

Q3) What is C language? Explain history of C language.

User can use c language to do **System Programming** (for writing operating system) as well as **Application Programming** (for generate menu driven customer billing system). That's why it is called middle level language.

Brief History of C

- The C programming language is a structure-oriented programming language, developed at Bell Laboratories in 1972 by Dennis Ritchie.

- C programming language features were derived from an earlier language called "B" (Basic Combined Programming Language – BCPL)
- C language was invented for implementing UNIX operating system.
- In 1978, Dennis Ritchie and Brian Kernighan published the first edition "The C Programming Language" and is commonly known as K&R C.
- In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or "ANSI C", was completed late 1988.
- Many of C's ideas & principles were derived from the earlier language B, thereby naming this new language "C".

USES

The C programming language is used for developing system applications that forms a major portion of operating systems such as Windows, UNIX and Linux.

Below are some examples of C being used:

- Database systems
- Graphics packages
- Word processors
- Spreadsheets
- Operating system development
- Compilers and Assemblers
- Network drivers
- Interpreters

Q4) Explain features or characteristics of c language?

A) Characteristics of C Programming Language

1) C is a **General-Purpose Programming Language** This means C can be used to write a variety of applications. It is often referred to as a "system programming language."
2) C is a **middle level language**, which means it combines the features of high-level language with the functionality of an assembly language.
3) C is a **structured programming language,** which means as a programmer, you are required to divide a problem into a several different modules or functions.
4) C is renowned for its **simplicity** and is easy to use because of its structured approach. It has a vast collection of keywords, operators, built-in functions and data types which make it efficient and powerful.
5) C is **portable**, which means a C program runs in different environments. C compilers are available for all operating systems and hardware platforms. Additionally, you can easily write code on one system and port it to another.

6) C is a very **flexible language**; it is convenient and portable, like a high-level language and flexible like a low-level language. It can be interfaced with other programming languages.
8) C is **super-fast**. The compilation and execution of programs is much faster on C than with most other languages.
9) C is **modular**, which means C programs can be divided into small modules, which are much easier to understand.
10) C is **easily available**. The C software is easy to access and can be easily installed on your computer. The installation of C hardly takes a few minutes.
11) C is easy to **debug**. The C compiler detects syntax errors quickly and easily and displays the errors along with the line numbers of the code and the error message.
12) C makes available a number **of in-built memory management** functions that save memory and improve the efficiency of the program such as malloc(), calloc() and alloc().
13**) Recursion** is one of the common techniques used in C, where in a function calls itself again and again.
.

Q 5) Explain structure of "C language "program?

A) Any C program is consisting of 6 main sections. Below you will find brief explanation of each of them.

Basic Structure of C Programs
Documentation Section
Link Section
Definition Section
Global Declaration Section
main() Function Section { Declaration Part Executable Part }
Subprogram Section Function 1 Function 2 Function 3 - - - Function n

Basic Structure of C Program

1. **Documentation Section**

This section consists of comment lines which include the name of programmer, the author and other details like time and date of writing the program. Documentation section helps anyone to get an overview of the program.

2. **Link Section**

The link section consists of the header files of the functions that are used in the program. It provides instructions to the compiler to link functions from the system library.

3. **Definition Section**

All the symbolic constants are written in definition section. Macros are known as symbolic constants.

4. **Global Declaration Section**

The global variables that can be used anywhere in the program are declared in global declaration section. This section also declares the user defined functions.

5. **main() Function Section**

It is necessary have one main() function section in every C program. This section contains two parts, declaration and executable part. The declaration part declares all the variables that are used in executable part. These two parts must be written in between the opening and closing braces. Each statement in the declaration and executable part must end with a semicolon (;). The execution of program starts at opening braces and ends at closing braces.

6. **Subprogram Section**

The subprogram section contains all the user defined functions that are used to perform a specific task. These user defined functions are called in the main() function.

Sample "C program :-

```
#include <stdio.h>
int main()
{
  // printf() displays the string inside quotation
  printf("Hello, World!");

}
```

Output

Hello, World!

Q 6) Explain C-Programming Rules

- All lines of C statements must end with a semicolon.

- C is case-sensitive. That is, uppercase and lowercase characters are different. Usually, the statements are typed in lowercase.
- A C Program statement can be split into many lines or can be written in one line.
- Braces must always match upon pairs, i.e., every opening brace { must have a matching closing brace }.
- All C program starts with void main( ) function.
- Comments cannot be nested. For example,

  /* Welcome to C World ,/* programming*/ */

- Comment can be split into more than one line as per by the user preference

Q7: Explain ALGORITHM and FLOWCHART in C-language?

The word "algorithm" relates to the name of the mathematician Al-khowarizmi, which means a procedure or a technique.
An algorithm is a sequence of steps to solve a particular problem or algorithm is an ordered set of unambiguous steps that produces a result and terminates in a finite time.

Algorithm has the following characteristics

• **Input**: An algorithm may or may not require input

• **Output:** Each algorithm is expected to produce at least one result

• **Definiteness**: Each instruction must be clear and unambiguous.

• **Finiteness**: If the instructions of an algorithm are executed, the algorithm should terminate after finite number of steps

HOW TO WRITE ALGORITHM

EXAMPLE:

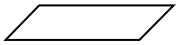
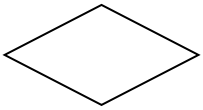
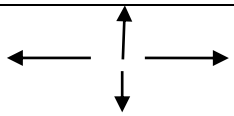
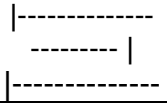
Step 1 Define your algorithm input: e.g. to calculate the area of rectangle input may be the rectangle height and rectangle width.
Step 2 Define the variables: We can define two variables for rectangle height and rectangle width as HEIGHT and WIDTH (or H & W). We should use meaningful variable name e.g. instead of using H & W use HEIGHT and WIDTH as variable name.
Step 3 Outline the algorithm's operations: Use input variable for computation purpose, e.g. to find area of rectangle multiply the HEIGHT and WIDTH variable and store the value in new variable (say) AREA.
Step 4 Output the results of your algorithm's operations: In case of area of rectangle output will be the value stored in variable AREA. if the input variables described a

rectangle with a HEIGHT of 2 and a WIDTH of 3, the algorithm would output the value of 6.

FLOWCHART:

The first design of flowchart goes back to 1945 which was designed by John Von Neumann. Unlike an algorithm, Flowchart uses different symbols to design a solution to a problem. It is another commonly used programming tool. Flowchart is often considered as a blueprint of a design used for solving a specific problem.

Flowchart is diagrammatic /Graphical representation of sequence of steps to solve a problem. To draw a flowchart following standard symbols are use.

Symbol Name	Symbol	function
Oval		Used to represent start and end of flowchart
Parallelogram		Used for input and output operation
Rectangle		Processing: Used for arithmetic operations and data-manipulations
Diamond		Decision making. Used to represent the operation in which there are two/three alternatives, true and false etc
Arrows		Flow line Used to indicate the flow of logic by connecting symbols
Circle		Page Connector
		Comments

Algorithm & Flowchart to find the sum of two numbers

Algorithm

Step-1 Start

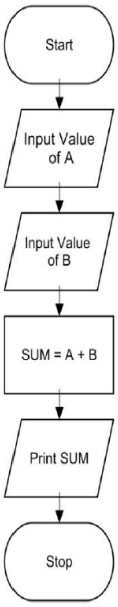
Step-2 Input first numbers say A

Step-3 Input second number say B

Step-4 SUM = A + B

Step-5 Display SUM

Step-6 Stop



Algorithm & Flowchart to find Area and Perimeter of Circle

R : Radius of Circle
AREA : Area of Circle
PERIMETER : Perimeter of Circle

Algorithm

Step-1 Start

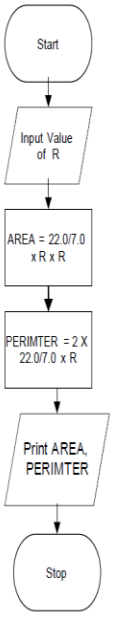
Step-2 Input Radius of Circle say R

Step-3 Area = 22.0/7.0 x R x R

Step-4 PERIMETER = 2 x 22.0/7.0 x R

Step-5 Display AREA, PERIMETER

Step-6 Stop



Q8) Explain Commonly used library functions?

Standard Library Functions in C

Standard Library Functions are basically the inbuilt functions in the C compiler that makes things easy for the programmer.

As we have already discussed, every C program has at least one function, that is, the **main() function**. The main() function is also a standard library function in C since it is inbuilt and conveys a specific meaning to the C compiler.

Header Files in C

In order to access the standard library functions in C, certain header files need to be included before writing the body of the program.

Here is a tabular representation of a list of header files associated with some of the standard library functions in C:

HEADER	MEANING	ELUCIDATION
<stdio.h>	Standard input-output header	Used to perform input and output operations like scanf() and printf().
<string.h>	String header	Used to perform string manipulation operations like strlen and strcpy.
<conio.h>	Console input-output header	Used to perform console input and console output operations like clrscr() to clear the screen and getch() to get the character from the keyboard.

<stdlib.h>	Standard library header	Used to perform standard utility functions like dynamic memory allocation using functions such as malloc() and calloc().
<math.h>	Math header	Used to perform mathematical operations like sqrt() and pow() to obtain the square root and the power of a number respectively.
<ctype.h>	Character type header	Used to perform character type functions like isaplha() and isdigit() to find whether the given character is an alphabet or a digit. respectively.

Let us discuss some of the commonly used Standard library functions in C in detail:

1. <stdio.h>

This is the basic header file used in almost every program written in the C language.
It stands for standard input and standard output used to perform input-output functions, some of which are:

- **printf()**– Used to display output on the screen.
- **scanf()**– To take input from the user.
- **getchar()**– To return characters on the screen.
- **putchar()**– To display output as a single character on the screen.
- **fgets()**– To take a line as an input.
- **puts()**– To display a line as an output.
- **fopen()**– To open a file.
- **fclose()**– To close a file.

2. <stdlib.h>

Functions such as malloc(), calloc(), realloc() and free() can be used while dealing with dynamic memory allocation of variables

2.1 malloc() :*malloc() stands for* memory allocation.
2.2 calloc(): *calloc stands for contiguous allocation*
2.3 realloc():*realloc stands for reallocation.*
2.4 free(): *free is responsible to free the dynamically allocated memory done by malloc(), calloc() or realloc() to the system.*

3. <math.h>

The math header is of great significance as it is used to perform various mathematical operations such as:

- **sqrt()** – This function is used to find the square root of a number
- **pow()** – This function is used to find the power raised to that number.
- **fabs()** – This function is used to find the absolute value of a number.
- **log()** – This function is used to find the logarithm of a number.
- **sin()** – This function is used to find the sine value of a number.
- **cos()** – This function is used to find the cosine value of a number.
- **tan()** – This function is used to find the tangent value of a number.

4. <ctype.h>

This function is popularly used when it comes to character handling.

Some of the functions associated with <ctype.h> are:

- **isalpha()** – Used to check if the character is an alphabet or not.
- **isdigit()** – Used to check if the character is a digit or not.
- **isalnum()** – Used to check if the character is alphanumeric or not.
- **isupper()** – Used to check if the character is in uppercase or not
- **islower()** – Used to check if the character is in lowercase or not.
- **toupper()** – Used to convert the character into uppercase.
- **tolower()** – Used to convert the character into lowercase.

5. <conio.h>

It is used to perform console input and console output operations like **clrscr()** to clear the screen and **getch()** to get the character from the keyboard.

Q9) Explain creating, compiling, linking, and executing the "C program".

A) **Step by Step Execution of C Program**

Step 1: Create & Saving

- This is First Step i.e. **Creating and Editing Program**.
- First Write C Program using Text Editor, such as [**Borland C/C++ 3.0**, **turbo C**]
- Save Program by using **[.C] Extension**.
- File Saved with [.C] extension is called "**Source Program** ".

Step 2: **Compilation** is the process of converting high level language instructions into low level language instructions. We use the shortcut key **Alt + F9** to compile a C program in **Turbo C**. Compiling

- Compiling C Program: C Source code with [.C] Extension is given as input to compiler and compiler convert it into **Equivalent Machine Instruction**.
- In Borland C/C++ 3.0 or turbo c, program can be compiled using key **[Alt + F9]**.
- **Compiler Checks for errors**. If source code is error-free then Code is converted into Object File [. Obj].

Step 3: Checking Errors

- During **Compilation Compiler will check for error**, If compiler finds any error then it will report it.
- User have to **re-edit the program**.
- After **re-editing program, Compiler again check for any error**.
- If program is error-free then program is linked with appropriate libraries.

Step 4: linker: Linker combines the code from specified header file into object file and generates executable file as Sample.exe. With this compilation process completes.

Linking Libraries

- Program is linked with **included header files**.
- Program is linked with other libraries.
- This process is **executed by Linker**.

Step 5: Executing / Running Executable File (Ctrl + F9)

1. After completing compilation successfully, an executable file is created with **.exe** extension.
2. The processor can understand this **.exe** file content so that it can perform the task specified in the source file.
3. We use a shortcut key **Ctrl + F9** to run a C program. Whenever we press **Ctrl + F9**, the **.exe** file is submitted to the **CPU**.
4. On receiving **.exe** file, **CPU** performs the task according to the instruction written in the file. The result generated from the execution is placed in a window called **User Screen**.

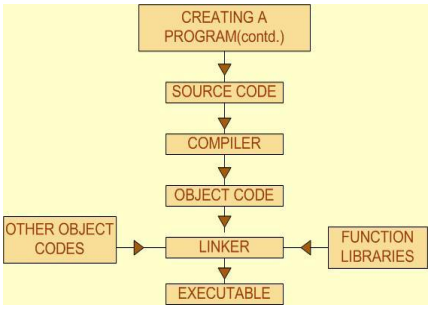
Step 6: Check Result (Alt + F5)

1. After running the program, the result is placed into **User Screen**.
2. Just we need to open the User Screen to check the result of the program execution.
3. We use the shortcut key **Alt + F5** to open the User Screen and check the result.

Overall process in simple steps:

- Type the program in C editor and save with **.c extension** (Press **F2** to save).
- Press **Alt + F9** to compile the program.

- If there are errors, correct the errors and recompile the program.
- If there are no errors, then press **Ctrl + F9** to execute / run the program.
- Press **Alt + F5** to open User Screen and check the result.

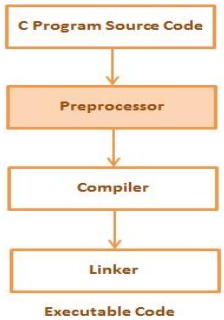


Q10) Write about Preprocessors in "C" language.

A) C PREPROCESSOR DIRECTIVES:

A Preprocessor is a system software (a computer program that is designed to run on computer's hardware and application programs). It performs preprocessing of the High-Level Language (HLL). Preprocessing is the first step of the language processing system. Language processing system translates the high-level language to machine level language or absolute machine code (i.e. to the form that can be understood by machine).

- Before a C program is compiled in a compiler, source code is processed by a program called preprocessor. This process is called preprocessing.
- Commands used in preprocessor are called preprocessor directives and they begin with **"#"** symbol.
- It removes all the comments. A comment is written only for the humans to understand the code. So, it is obvious that they are of no use to a machine. So, preprocessor removes all of them as they are not required in the execution and won't be executed as well.



TYPES OF PREPROCESSOR:

I) **Macro Expansion**: It is used to assign names to different constants or statements which are to be used repeatedly in a program. These defined values or statement can be used by main or in the user defined functions as well. They are used for

a. defining a constant. This macro defines constant value and can be any of the basic data types.

b. defining a statement

c. defining a mathematical expression

Syntax: #define

For example:

#define PI 3.141593

II) File Inclusion: The include directive is used to include files like as we include header files in the beginning of the program using **#include** directive like

Syntax: #include<header file>

ex: #include<stdio.h>

#include<conio.h>

III) Conditional Compilation: Set of commands are included or excluded in source program before compilation with respect to the condition. To use conditional, #ifdef, #if, #defined, #else and #elseif directives are used.

A program in C language involves into different processes. Below diagram will help you to understand all the processes that a C program comes across.

Q11) Write about keywords and identifiers?

Keywords are preserved words that have special meaning in C language. The meaning of C language keywords has already been described to the C compiler. These meaning cannot be changed. Thus, keywords cannot be used as variable names because that would try to change the existing meaning of the keyword, which is not allowed. There are total 32 keywords in C language.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Identifiers:

In C language identifiers are the names given to variables, constants, functions and user-define data. These identifiers are defined against a set of rules.

Rules for an Identifier

- An Identifier can only have alphanumeric characters (a-z, A-Z, 0-9) and underscore(_).
- The first character of an identifier can only contain alphabet (a-z, A-Z) or underscore (_).
- Identifiers are also case sensitive in C. For example, **name** and **Name** are two different identifiers in C.
- Keywords are not allowed to be used as Identifiers.
- No special characters, such as semicolon, period, whitespaces, slash or comma are permitted to be used in or as Identifier.

Character set

In C language characters are grouped into the following categories,

- Letters (all alphabets a to z & A to Z).
- Digits (all digits 0 to 9).
- Special characters, (such as colon: semicolon;, period ., underscore _, ampersand &etc).
- White spaces.

Q12) What is a constant?

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like

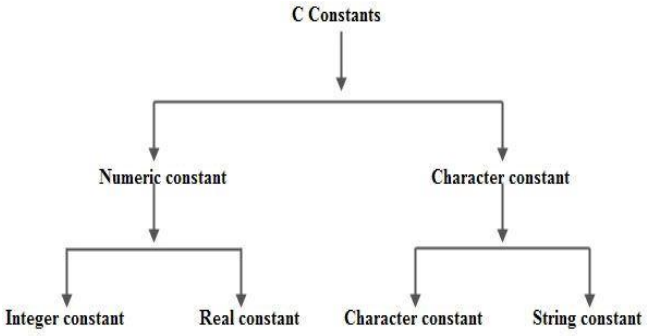
1) an integer constant,

2)a floating constant,

3)a character constant,

4) string literal.

There are enumeration constants as well.



1)Integer constant

An integer literal or integer constant is a numeric constant (associated with number) without any fractional or exponential part. can be a decimal, octal, or hexadecimal

constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

examples of integer literals −

```
212        /* valid */
215u       /* valid */
```

- Embeded spaces and non-digit character are not permitted between the digits.

```
Example :   12,342      /* invalid */
            $234        /* invalid */
            34 465      /* invalid */
```

2)Floating-point

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

While representing decimal form, you must include the decimal point, the exponent, or both; and while representing exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

examples of floating-point literals −

```
3.14159       /* Legal */
314159E-5L    /* Legal */
```

3)Character Constants

Character literals are enclosed in single quotes, e.g., 'x' can be stored in a simple variable of **char** type.

A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

There are certain characters in C that represent special meaning when preceded by a backslash for example, newline (\n) or tab (\t).

Here, you have a list of such escape sequence codes –

Following is the example to show a few escape sequence characters −

```
#include <stdio.h>
int main()
{
   printf("Hello\tWorld\n\n");
   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Hello World
```

4)String Literals

String literals or constants are enclosed in double quotes " ". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separating them using white spaces.

examples of string literals.

"hello, dear"

"hello, \
dear"

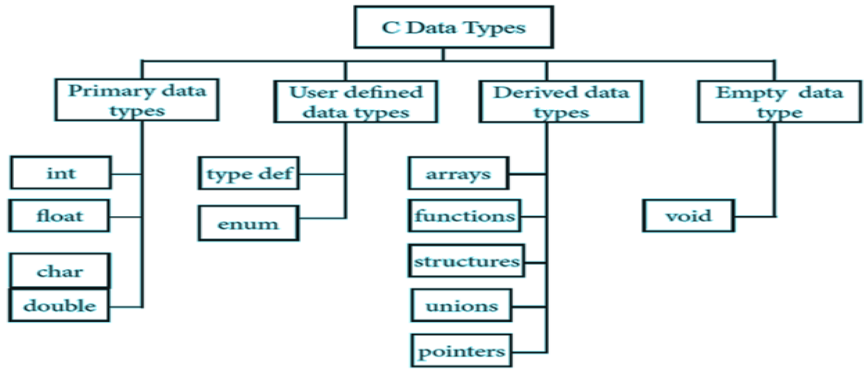
"hello, " "d" "ear"

13Q) Explain data types "C" language in detail.

A) C data types are defined as the data storage format that a variable can store a data to perform a specific operation.

- Data types are used to define a variable before to use in a program.
- Size of variable, constant and array are determined by data types.

C – DATA TYPES:



There are four data types in C language. They are,

Types	Data Types
Basic data types / primary datatypes	int, char, float, double
User defined data type/ Enumeration data type	enum
Derived data type	pointer, array, structure, union
Void data type	void

1. BASIC DATA TYPES/PRIMARY DATA TYPE IN C LANGUAGE:

1.1. INTEGER DATA TYPE:

- Stores numeric value(whole number)
- Int keyword used
- Storage size = 2, 4, 8 bytes
- (2 byte) can store values from -32,768 to +32,767

- If you want to use the integer value that crosses the above limit, you can go for "long int" and "long long int" for which the limits are very high.
- Types of interger variables
  - short
  - long
  - signed
  - unsigned
  - long long

1.2. CHARACTER DATA TYPE:

- Character data type allows a variable to store only one character.
- Storage size of character data type is 1. We can store only one character using character data type.
- "**char**" keyword is used to refer character data type.
- For example, 'A' can be stored using **char** datatype. You can't store more than one character using char data type.

1.3. FLOATING POINT DATA TYPE:

Floating point data type consists of 2 types. They are,

1. float
2. double

1. FLOAT:

- Float data type allows a variable to store decimal values.
- "float" is a keyword to refer float datatype.
- Storage size of float data type is 4. This also varies depend upon the processor in the CPU as "int" data type.
- We can use up-to 6 digits after decimal using float data type.
- For example, 10.456789 can be stored in a variable using float data type.

2. DOUBLE:

- Double data type is also same as float data type which allows up-to 10 digits after decimal.

- The range for double datatype is from 1E–37 to 1E+37.

2. USER DEFINED DATA TYPE OR ENUMERATION DATA TYPE IN C LANGUAGE:

- Enumeration data type consists of named integer constants as a list.
- It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.
- Enum syntax in C:
  enum identifier [optional{ enumerator-list }];
- Enum example in C:

3. DERIVED DATA TYPE IN C LANGUAGE:

Types of derived data type :
1)Array, 2) pointer, 3)structure and 4) union

- " Array" : C Array is a collection of variables belongings to the same data type. You can store group of data of same data type in an array.
  Ex : int a[10];
- " Pointer" : Pointers in C language is a variable that stores/points the address of another variable.
  Ex: int *p; char *p;
- "Structure": C Structure is a collection of different data types which are grouped together and each element in a C structure is called member.

```
Ex : struct student
{
int a;
char b[10];
}
```

- " Union" : C Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called member.

```
Ex:  union tag_name
      {
      data type var_name1;
      data type var_name2;
     data type var_name3;
      };
```

4. VOID DATA TYPE IN C LANGUAGE:

- Void is an empty data type that has no value.
- This can be used in functions and pointers.

Q 14) What is a variable? Explain declaration and initialization of variable?

A) C variables are names used for storing a data value to locations in memory. The value stored in the c variables may be changed during program execution The value of the C variable may get change in the program.

C variable might be belonging to any of the data type like int, float, char etc.

RULES FOR NAMING C VARIABLE:

Variable name must begin with letter or underscore.

Variables are case sensitive

They can be constructed with digits, letters.

No special symbols are allowed other than underscore.

sum, height, _value are some examples for variable name

DECLARING & INITIALIZING C VARIABLE:

Variables should be declared in the C program before to use.

Memory space is not allocated for a variable while declaration. It happens only on variable definition.

Variable initialization means assigning a value to the variable.

Declaration of Variable

Declaration of variable in c can be done using following syntax:

data_type variable_name;
or
data_type variable1, variable2,…,variablen;

where *data_type* is any valid c data type and *variable_name* is any valid identifier.

For example,

```
int a;
float variable;
float a, b;
```

Initialization of Variable

C variables declared can be initialized with the help of assignment operator '='.

Syntax

data_type variable_name=constant/literal/expression;
or
variable_name=constant/literal/expression;

Example

```
int a=10;
 int a=b+c;
 a=10;
 a=b+c;
```

Multiple variables can be initialized in a single statement by single value, for example, a=b=c=d=e=10;

NOTE: C variables must be declared before they are used in the c program. Also, since c is a case sensitive programming language, therefore the c variables, abc, Abc and ABC are all different.

PROGRAM FOR DECLARATION AND INITIALIZATION OF VARIABLES:

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int a = 10;
int b = 30;
int C;
clrscr();
C = a+ B;
Printf("the addition of two no.s is :%d",C);
}
```

Q 15) Explain scope of variables in C language?

A) **TWO TYPES OF SCOPE OF VARIABLES IN C PROGRAM THEY ARE,**

1. Local variable
2. Global variable

1. LOCAL VARIABLE:

- The scope of local variables will be within the function only.
- These variables are declared within the function and can't be accessed outside the function.
- In the below example, m and n variables are having scope within the main function only. These are not visible to test function.
- Likewise, a and b variables are having scope within the test function only. These are not visible to main function.

```
#include<stdio.h>
void test();
 int main()
```

```
{
  int m = 22, n = 44;
  printf("\nvalues : m = %d and n = %d", m, n);
  test();
}

void test()
{
  int a = 50, b = 80;
  printf("\nvalues : a = %d and b = %d", a, b);
}
```

```
values : m = 22 and n = 44

values : a = 50 and b = 80
```

2. GLOBAL VARIABLE IN C:

- The scope of global variables will be throughout the program.
- These variables can be accessed from anywhere in the program.
- This variable is defined outside the main function. So that, this variable is visible to main function and all other sub functions.

```
#include<stdio.h>
void test();
int m = 22, n = 44;
int a = 50, b = 80;
int main()
{
  printf("All variables are accessed from main function");
  printf("\nvalues: m=%d:n=%d:a=%d:b=%d", m,n,a,b);
  test();
}
 void test()
{
  printf("\n\nAll variables are accessed from" \
  " test function");
  printf("\nvalues: m=%d:n=%d:a=%d:b=%d", m,n,a,b);
}
```

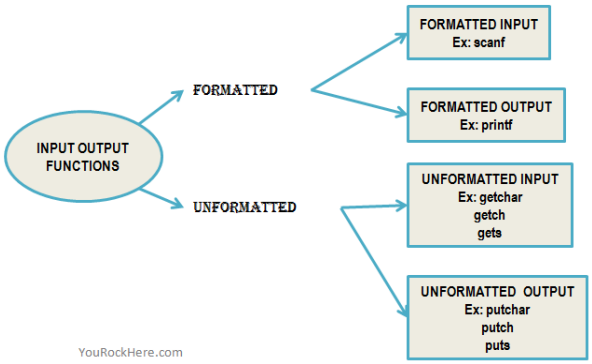
COMPILE & RUN

OUTPUT:

```
All variables are accessed from main function
values : m = 22 : n = 44 : a = 50 : b = 80
All variables are accessed from test function
values : m = 22 : n = 44 : a = 50 : b = 80
```

Input and output operations in "C" language

Input and output operations are of two types:
1) Formatted
2) Unformatted



Q 16) Explain formatted input / output operations?

A) Formatted input and output operations are of two types.

1) scanf ()
2)print()

- printf() and scanf() functions are inbuilt library functions in C programming language which are available in C library by default. These functions are declared and related macros are defined in "stdio.h" which is a header file in C language.
- We have to include "stdio.h" file as shown in below C program to make use of these printf() and scanf() library functions in C language.
- C language is case sensitive. For example, printf() and scanf() are different from Printf() and Scanf(). All characters in printf() and scanf() functions must be in lower case.

1) Scanf():-In C programming language, scanf() function is used to read character, string, numeric data from keyboard.

The *scanf* function is used to read formatted input data. The format in which input data is to be provided is specified by the scanf function itself as it's first parameter or argument. The scanf function is written as -

Syntax : scanf("format specifiers", &variable1, &variable2, . . . ,);

Here

1) The first parameter contains a list of format specifiers indicating the format and type of data to be read.
2) The remaining parameters - &variable1, &variable2, ..., are addresses of the variables where the read data will be stored.
3) *scanf* reads the input data as per the format specifiers and stores them (i.e., assigns them) to the corresponding addresses.
4) An **&** is pre-fixed to the variable name to denote its address.

Note that there must be the same number of format specifiers and addresses as there are input data

The scanf function uses format specifiers,

- **int** uses **%d**
- **float** uses **%f**
- **char** uses **%c**
- **character strings** (discussed later) use **%s**

printf():-The usually used output statement is printf (). It is one of the library functions.

- In C programming language, printf() function is used to print the "character, string, float, integer, octal and hexadecimal values" onto the output screen.
- We use printf() function with %d format specifier to display the value of an integer variable.
- Similarly %c is used to display character, %f for float variable, %s for string variable, %lf for double and %x for hexadecimal variable.
- To generate a newline,we use "\n" in C printf() statement.

Syntax : printf("text");

Ex: printf("welcome to "C" language");

Syntax : printf("format string",&variable1,&variable2…)

Ex: printf("%d %f",&a,&b);

Sample program for printf and scanf operations :

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

```
Void main()
{
int  x, y, reslut;
 printf("enter first number:");
scanf("%d",&x);
printf("enter second number:");
scanf("%d",&y);
result=x+y;
printf("sum of 2 numbers:%d ",result);
}
```

Q 17) Explain unformatted input /output functions in C language.

A) 1**) putchar() :-**putchar() function is a file handling function in C programming language which is used to write a character on standard output/screen. Please find below the description and syntax for above file handling function.

Declaration: int putchar(int char)

putchar() function is used to write a character on standard output/screen.
Ex:- putchar(char);

2**) getchar() :-** getchar() function is used to get/read a character from keyboard input.

Declaration: int getchar(void)

getchar() function is used to get/read a character from keyboard input. In a C program, we can use getchar function as below.

ex:- getchar(char);

where, char is a character variable/value.

Sample C program for putchar() and getchar() :

```
#include <stdio.h>
#include <ctype.h>
int main()
{
  char c;
  printf("Enter some character. Enter $ to exit...\n");
  while (c != '$');
  {
    c = getchar();
```

```
        printf("\n Entered character is: ");
        putchar(c);
        printf("\n")
      }
      return 0;
    }
```

1) **gets()** : The gets() function reads string from user and
   syntax : gets(variablename)

2) **puts()** : puts() function prints the string.
   Syntax : puts(variable name)

Both functions are defined in <stdio.h> header file.

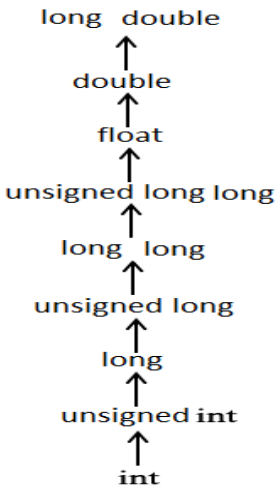
Sample program

```
#include<stdio.h>
#include <string.h>
int main()
{
char name[50];
   printf("Enter your name: ");
   gets(name); //reads string from user
   printf("Your name is: ");
   puts(name);  //displays string
 return 0;
}
```

Q 18)Explain Type Conversion in C

A type cast is basically a conversion from one type to another. There are two types of type conversion:

1. **Implicit Type Conversion** Also known as 'automatic type conversion'.
   - Done by the compiler on its own, without any external trigger from the user.
   - Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
   - All the data types of the variables are upgraded to the data type of the variable With the data.



- It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

Program for implicit type conversions

```
#include<stdio.h>
int main()
{
    int x = 10;    // integer x
    char y = 'a';  // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

Explicit Type Conversion– This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.

The syntax in C:

Syntax: (type) expression

Type indicated the data type to which the final result is converted.

Program for explicit type conversion:

```
#include<stdio.h>

int main()
{
   double x = 1.2;

   // Explicit conversion from double to int
   int sum = (int)x + 1;

   printf("sum = %d", sum);

   return 0;
}
```

Advantages of Type Conversion

- This is done to take advantage of certain features of type hierarchies or type representations.
- It helps us to compute expressions containing variables of different data types.

Q19) WHAT ARE OPERATORS? EXPLAIN TYPES OF OPERATORS IN "C" LANGUAGE.

A) An operator is a symbol which operates on a value or a variable. For example: `+` is an operator to perform addition.

C programming has wide range of operators to perform various operations. For better understanding of operators, these operators can be classified as:

Operators in C programming
1) Arithmetic Operators
2) Increment and Decrement Operators
3) Assignment Operators

4) Relational Operators
5) Logical Operators
6) Conditional Operators
7)Special Operators

1) Arithmetic Operators: An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined together to form expressions.
- Consider the expression A + B * 5 where, +, * are operators, A, B are variables, 5 is constant and A + B * 5 is an expression.

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division(modulo division)

Sample program for arithmetic operators:

```
#include <stdio.h>
int main()
{
   int a = 9,b = 4, c;

   c = a+b;
   printf("a+b = %d \n",c);
```

```
    c = a-b;
    printf("a-b = %d \n",c);

    c = a*b;
    printf("a*b = %d \n",c);

    c=a/b;
    printf("a/b = %d \n",c);

    c=a%b;
    printf("Remainder when a divided by b = %d \n",c);

    return 0;
}
```

2. Increment and decrement operator :

Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.

Syntax:

Increment operator: ++var_name; (or) var_name++;

Decrement operator: – -var_name; (or) var_name – -;

Example:

Increment operator : ++ i ; i ++ ;

Decrement operator : – – i ; i – – ;

Operator	Operator/Description
Pre increment operator (++i)	value of i is incremented before assigning it to the variable i
Post increment operator (i++)	value of i is incremented after assigning it to the variable i
Pre decrement operator (–i)	value of i is decremented before assigning it to the variable i

Post decrement operator (i–)	value of i is decremented after assigning it to variable i

Sample program for increment and decrement operator

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

void main()
{
int x,i;
i=10;
x=++i;
printf("x: %d",x);
printf("i: %d",i);
getch();
}
```

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

void main()
{
int  x,i;
i=10;
x=i++;
printf("x: %d",x);
printf("i: %d",i);
getch();
}
```

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

void main()
{
int x,i;
i=10;
x=--i;
printf("x: %d",x);
printf("i: %d",i);
getch();
}
```

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

void main()
{
int  x,a,b,c;
a = 2;
b = 4;
c = 5;
x = a-- + b++ -
++c;
printf("x: %d",x);
getch();
}
```

3.Relational Operators:

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.

>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Example:

```
#include <stdio.h>
Void main()
{
   int a = 5, b = 5, c = 10;

   printf("%d == %d = %d \n", a, b, a == b); // true
   printf("%d == %d = %d \n", a, c, a == c); // false
   printf("%d > %d = %d \n", a, b, a > b); //false
   printf("%d > %d = %d \n", a, c, a > c); //false
   printf("%d < %d = %d \n", a, b, a < b); //false
   printf("%d < %d = %d \n", a, c, a < c); //true
   printf("%d != %d = %d \n", a, b, a != b); //false
   printf("%d != %d = %d \n", a, c, a != c); //true
   printf("%d >= %d = %d \n", a, b, a >= b); //true
   printf("%d >= %d = %d \n", a, c, a >= c); //false
   printf("%d <= %d = %d \n", a, b, a <= b); //true
   printf("%d <= %d = %d \n", a, c, a <= c); //true

   return 0;
}
```

4. Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning of Operator	Example
&&	Logial AND. True only if all	If c = 5 and d = 2 then, expression ((c ==

Operator	Meaning of Operator	Example
	operands are true	5) && (d > 5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c == 5) || (d > 5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression ! (c == 5) equals to 0.

Program for logical operators:

```
#include <stdio.h>
int main()
{
   int a = 5, b = 5, c = 10, result;

   result = (a == b) && (c > b);
   printf("(a == b) && (c > b) equals to %d \n",
result);

   result = (a == b) && (c < b);
   printf("(a == b) && (c < b) equals to %d \n",
result);

   result = (a == b) || (c < b);
   printf("(a == b) || (c < b) equals to %d \n", result);

   result = (a != b) || (c < b);
   printf("(a != b) || (c < b) equals to %d \n", result);

   result = !(a != b);
   printf("!(a == b) equals to %d \n", result);

   result = !(a == b);
   printf("!(a == b) equals to %d \n", result);
   return 0;
}
```

5. Conditional operators:

A conditional operator is a ternary operator, that is, it works on 3 operands.

Conditional Operator Syntax

conditionalExpression ? expression1 : expression2

example: (A > 100 ? 0 : 1);

In above example, if A is greater than 100, 0 is returned else 1 is returned. This is equal to if else conditional statements.

The conditional operator works as follows:

- The first expression conditionalExpression is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false.
- If *conditionalExpression* is true, *expression1* is evaluated.
- If *conditionalExpression* is false, *expression2* is evaluated.

Program for C conditional Operator

```
#include<stdio.h>

int main()
{
  int age;

  printf(" Please Enter your age here: \n ");
  scanf(" %d ", &age);

 (age >= 18) ? printf(" You are eligible to Vote ") :  printf(" You are not eligible to Vote
");

 return 0;
}
```

6. ASSIGNMENT OPERATORS IN C:

In C programs, values for the variables are assigned using assignment operators.
For example, if the value “10” is to be assigned for the variable “sum”, it can be assigned as “sum = 10;”

There are 2 categories of assignment operators in C language. They are,
1. Simple assignment operator (Example: =)
2. Compound assignment operators (Example: +=, -=, *=, /=, %=, &=, ^=)

Operators	Example/Description
=	sum = 10; 10 is assigned to variable sum
+=	sum += 10; This is same as sum = sum + 10
-=	sum -= 10; This is same as sum = sum – 10
*=	sum *= 10; This is same as sum = sum * 10
/=	sum /= 10; This is same as sum = sum / 10
%=	sum %= 10; This is same as sum = sum % 10
&=	sum&=10; This is same as sum = sum & 10
^=	sum ^= 10; This is same as sum = sum ^ 10

Sample program for Assignment operators

```
#include <stdio.h>
int main()
{
   int a = 5, c;

   c = a;
   printf("c = %d \n", c);

   c += a; // c = c+a
   printf("c = %d \n", c);
```

```
    c -= a; // c = c-a
    printf("c = %d \n", c);

    c *= a; // c = c*a
    printf("c = %d \n", c);

    c /= a; // c = c/a
    printf("c = %d \n", c);

    c %= a; // c = c%a
    printf("c = %d \n", c);

    return 0;
}
```

7. Special operator:

Operators	Description
,	
&	This is used to get the address of the variable. Example : &a will give address of a.
*	This is used as pointer to a variable. Example : * a where, * is pointer to the variable a.
Sizeof ()	This gives the size of the variable. Example : size of (char) will give us 1.

1.Comma operator is a set of expression separated by comma is a valid constant in the C language. For example i and j are declared by the statements

```
int i , j;
i=(j=10,j+20);
```

In the above declaration, right hand side consists of two expressions separated by comma. The first expression is j=10 and second is j+20. These expressions are evaluated from left to right. ie first the value 10 is assigned to j and then expression j+20 is evaluated so the final value of i will be 30.

2.Reference or ampersand operator

The **reference operator** is denoted by & symbol. It is used to return the **address** of the variable. If **x** is a variable then **&x** will return the address of the **x**.

Program example of reference operator

```
#include<stdio.h>
int main()
{
   int a;
   printf("address of a %p\n",&a);
   return 0;
}
```

Output

address of a 0x7fff58dfcae8

3. Dereference operator

The **dereference operator** is also known as an **indirection operator**. It is denoted by * symbol. It is used to return the **value pointed by** the **pointer variable**. If **x** is a pointer, then ***x** represents the value contained in the address pointed by **x**.

Program example of dereference operator

```
#include<stdio.h>
int main()
{
   int a = 10, *x;
   x = &a;
   printf("value pointed by pointer x is %d\n",*x);
   return 0;
}
```

Output

value pointed by pointer x is 10

4. Sizeof () operator

The **sizeof() operator** is used to find out the size of the variables in C program. For example:

consider 'a' as an integer then

sizeof(a) would **return** 2 or 4 based on the compiler or sys

Program example of sizeof() operator

```
#include<stdio.h>
int main()
{
   char a;
   int b;
   float c;
   double d;

   printf("size of char %lu\n",sizeof(a));
   printf("size of int %lu\n",sizeof(b));
   printf("size of float %lu\n",sizeof(c));
   printf("size of double %lu\n",sizeof(d));
   return 0;

}
```

Output:

```
size of char 1
size of int 4
size of float 4
size of double 8
```

UNIT -2

WORKING WITH CONTROL STATEMENTS, LOOPS

Syllabus

Conditional statements:

- Introduction
- If statements
- If-else statements
- nested if-else
- break statement
- continue statement
- go to statement
- Switch statements

Looping statements:

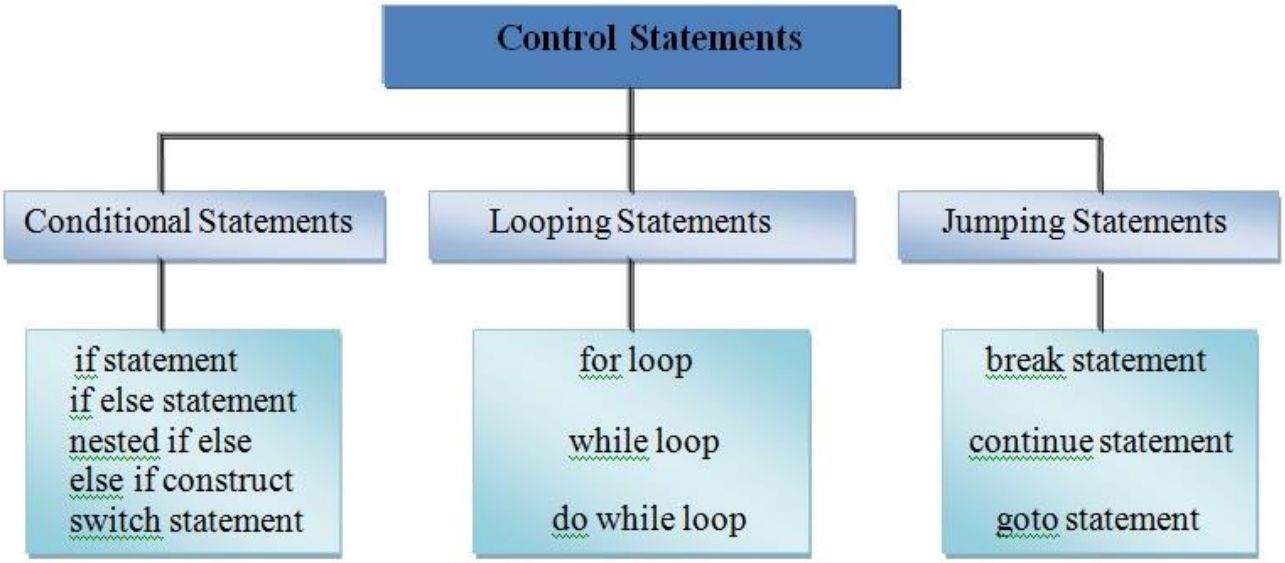
- Introduction
- While statements
- Do-while statements
- For Statements
- Nested loop statements.

Q1: What is Control Statements?

Control statements enable us to specify the flow of program control; ie, the order in which the instructions in a program must be executed. They make it possible to make decisions, to perform tasks repeatedly or to jump from one section of code to another.

There are three types of control statements in C:

1. Conditional (OR)Decision making statements
2. Iteration statements (OR) Looping statement
3. Jumping statements



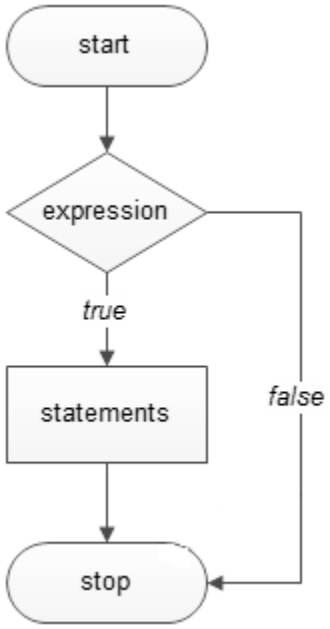
CONDITIONAL STATEMENTS OR DECISION MAKING

Q2: Explain Conditional Statements (OR) Decision making in "C" language?

A) C conditional statements allow you to make a decision, based upon the result of a condition. These statements are called as **Decision-Making Statements** or **Conditional Statements**.

Decision making statement is depending on the condition block need to be executed or not which is decided by condition.
If the condition is "true" statement block will be executed, if condition is "false" then statement block will not be executed.

The flowchart of Decision-making technique in C can be expressed as:



C languages have such decision-making capabilities within its program by the use of following the decision-making statements.

Conditional Statements in C

1. If statement
2. switch statement
3. Conditional Operator

IF STATEMENT

If statements in C is used to control the program flow based on some condition, it's used to execute some statement code block if the expression is evaluated to true. Otherwise, it will get skipped. This is the simplest way to modify the control flow of the program.

The if statement in C can be used in various forms depending on the situation and complexity.

There are four different types of if statement in C. These are:

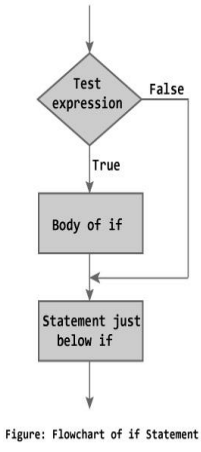
- Simple if Statement
- if-else Statement
- Nested if-else Statement
- else-if Ladder

Simple if statement:

The expression is evaluated and depending on whether the value of the expression is true (non zero) or false (zero) it transfers the control to a particular statement. The if statement checks whether condition expression is true or false. If the condition expression is true, then the block of code inside the if statement is executed but if condition expression is false, then the block of code inside the if statement is ignored and the next statement after if block is executed. If the expression is evaluated and found to be true, the single statement following the "if statement" is executed. If false, the following statement is skipped. Here a compound statement composed of several statements bounded by braces can replace the single statement.

Keywords: **if**

The general form of simple if statement is:

Syntax	**Example**	
if (expression) **{** **statement block;** **}** **statement n;**	if(x > y) { printf("x is greater than y"); }	 Figure: Flowchart of if Statement

write a C Program which demonstrate use of if statement.

```
#include<stdio.h>
 #include<conio.h>
 void main()
{
  int x,y;
   x = 7;
 y = 5 ;
 if(x > y)
  {
 printf("x is greater
than y");
   }
getch();
}
```

```
#include<conio.h>
int main()
{
int  number;
clrscr();
printf("Enter a number
\n");
scanf("%d"&number);
if(number>0)
printf("Given number is
positive \n");
getch();
return 0;
}
```

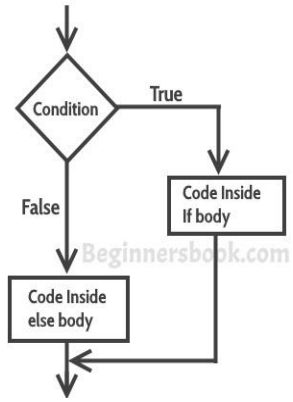
if...else statement

The if else statement in C programming language is used to execute a set of statements if condition is true and execute another set of statements when condition is false. Only either if block or else block of code gets executed (not both) depending on the outcome of condition.

The if else statement checks whether condition expression is true or false. If the condition expression is true, then the block of code inside the if statement is executed but if condition expression is false, then the block of code inside the else statement is executed.

Both if and else block of statements can never execute at a time.

Syntax:
The general form of a simple *if...else* statement is,

syntax	example	
if(expression) **{** **statement-block1;** **}** **else** **{** **statement-block2;** **}**	if (x > y) { printf("x is greater than y"); } else { printf("y is greater than x"); }	

If the '**expression**' is true, the '**statement-block1**' is executed,
else '**statement-block2**' is executed.
Keywords are: **if, else**

Program for if-else statement.
Write a C Program which demonstrate use of if else-statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int x,y;
clrscr();
   printf(“enter two no.s:”);
  scanf(“%d%d”,&x,&y);

   if (x > y)
   {
      printf("x is greater than y");
   }
   else
   {
      printf("y is greater than x");
   }
   getch();
}
```

Program for using if-else statement

```
#include <stdio.h>
int main()
{
   int number;
   printf("Enter an integer: ");
   scanf("%d",&number);

   // True if remainder is 0
   if( number%2 == 0 )
      printf("%d is an even integer.",number);
   else
      printf("%d is an odd integer.",number);
   return 0;
}
```

Nested if –else statement

Nested if else statment is same like if else statement, where new block of if else statement is defined in existing if or else block statment.

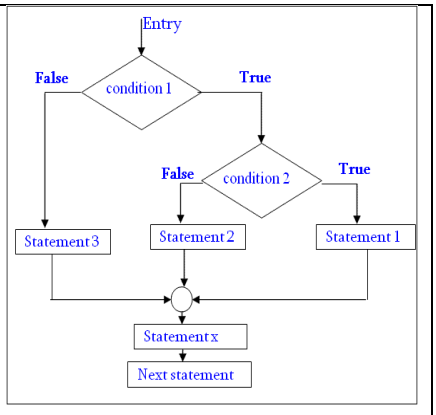
Used when user want to check more than one conditions at a time.

Syntax:

```
if(condition is true)
{
   if(condition is
true)
   {
      statement;
   }
   else
   {
      statement;
   }
}
else
{
   statement;
}
```

Example :

```
if(a>b)
  {
     if(a>c)  {
        printf("%d is greater ",a);
     }
     else     {
        printf("%d is greater ",c);
     }
  }
  else  {
     if (b>c)   {
        printf("%d is greater ",b);
     }
     else   {
        printf("%d is greater ",c);
     }
  }
```



Program for nested if-else statement

Largest of two numbers.

```
void main()
 {
    int a,b,c;
    printf("Enter three numbers
separated by space ");
    scanf("%d%d%d",&a,&b,&c);
    if(a>b)
    {
       if(a>c)
       {
          printf("%d is greater ",a);
       }
       else
       {
          printf("%d is greater ",c);
       }
    }
    else
    {
       if (b>c)
       {
          printf("%d is greater ",b);
       }
       else
       {
          printf("%d is greater ",c);
       }
    }
 }
```

To find negative or positive number.

```
#include<stdio.h>
int main()
{
   int n;
   printf("Enter a number:");
   scanf("%d",&n);
   if(n<0)
      printf("Number is negative");
   else if(n>0)
      printf("Number is positive");
   else
      printf("Number is equal to
zero");
   return 0;
}
```

else if ladder

The if else ladder statement in C programming language is used to test set of conditions in sequence. An if condition is tested only when all previous if conditions in if-else ladder is false. If any of the conditional expression evaluates to true, then it will execute the corresponding code block and exits whole if-else ladder.

Syntax:

```
if(condition_expression_One)
{
   statement1;
} else if (condition_expression_Two)
{
   statement2;
} else if (condition_expression_Three)
{
   statement3;
} else
{
   statement4;
}
```

- First of all, condition expression One is tested and if it is true then statement1 will be executed and control comes out of whole if else ladder.
- If condition expression One is false then only condition expression Two is tested. Control will keep on flowing downward, If none of the conditional expression is true.
- The last else is the default block of code which will gets executed if none of the conditional expression is true.

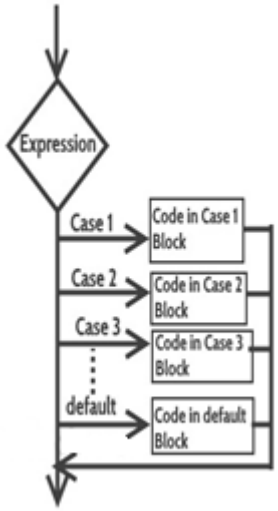
Program for else if ladder	Program for else if ladder
#include<stdio.h> #include<conio.h>	#include<stdio.h> #include<conio.h> void main()

```
void main()
{
   int marks;
    clrscr();
   printf("Enter your marks between 0-100\n");
   scanf("%d", &marks);
   if(marks >= 90)
{
      printf("YOUR GRADE : A\n");
   } else if (marks >= 70 && marks < 90)
    {
      printf("YOUR GRADE : B\n");
   } else if (marks >= 50 && marks < 70)
    {
           printf("YOUR GRADE : C\n");
    }    else
     {
           printf("YOUR GRADE : Failed\n");
     }

   getch();
   return(0);
}
```

```
{
   int a;
   printf("Enter a Number: ");
   scanf("%d",&a);
   if(a > 0)
   {
      printf("Given Number is Positive");
   }
   else if(a == 0)
   {
      printf("Given Number is Zero");
   }
   else if(a < 0)
   {
      printf("Given Number is Negative");
   }
   getch();
}
```

SWITCH STATEMENT

A switch statement allows a variable or value of an expression to be tested for equality against a list of possible case values and when match is found, the block of code associated with that case is executed.

Syntax :	***Example :***	
switch(expression) ***{*** ***case constant1 :*** ***statement;*** ***break;*** ***case constant2 :*** ***statement;*** ***break;*** ***statement;*** ***break;*** ***default : /* Optional */*** ***statement;*** ***}***	**switch**(num % 2) { *//If n%2 == 0* **case** 0: printf("Number is even"); **break**; *//Else if n%2 == 1* **case** 1: printf("Number is odd"); **break**; }	

Important Points about Switch Statement

- Switch case performs equality check of the value of expression/variable against the list of case values.
- The expression in switch case must evaluates to return an integer, character or enumerated type
- You can use any number of case statements within a switch. The expression value is compared with the constant after case.
- The data type of the value of expression/variable must be same as the data type of case constants.
- The break statement is optional. The break statement at the end of each case cause switch statement to exit. If break statement is not used, all statements below that case statement are also executed until it found a break statement.
- The default code block gets executed when none of the case matches with expression. default case is optional and doesn't require a break statement.

C Program using Switch case statement

```
#include <stdio.h>
#include<conio.h>
Void  main()
{
   int num;
clrscr();
     printf("Enter any number to check even or odd: ");
   scanf("%d", &num);

   switch(num % 2)
   {
      //If n%2 == 0
      case 0: printf("Number is even");
            break;

      //Else if n%2 == 1
      case 1: printf("Number is odd");
            break;
   }
   return 0;
getch();
}
```

- The complexity of the program increases as the number of constants to compare increases. Switch statement provides a way to write easy to manage and readable code.
- **Switch statement** provides a default case handler when no case matches.
- We can nest switch statements like if else statements.

CONDITIONAL OPERATOR

Conditional Operators in C Programming

Conditional Operator in C is a powerful Operator which can be used to implement if-then-else type of logic. This operator is also known as ternary operator as it takes three expressions in following form.

Conditional_Expression ? Expression_One : Expression_Two;

Ternary Operator will execute Expression_One if Conditional_Expression is true, otherwise it execute Expression_Two.
Ternary Operator is similar to if-else decision block as it evaluates only one code block depending on the result of Conditional_Expression
For Example
int X = 25;
int Y = (X > 20? 1: 2);
As X > 20, So after above statement Y's value becomes 1.

C Program to find larger number using Conditional Operator

Below program takes two numbers as input from user and stores them in integer variable 'A' and 'B'. Then using *ternary operator,* it prints whether A >= B or A < B.

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

int main()
{
    int A, B;
    printf("Enter two numbers\n");
   scanf("%d %d", &A, &B);
    (A>=B) ? printf("%d is >= %d\n",A,B) : printf("%d is < %d\n",A,B);
    getch();
   return(0);
}
```

Looping statements

Q3) What are the looping statements in C?

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are mainly three types of loops used in the C language.
It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using print f 10 times we can print inside a loop which runs up to 10 iterations.

Advantage with looping statement

- Reduce length of Code
- Take less memory space.
- Burden on the developer is reducing.
- Time consuming process to execute the program is reduced.
- Provides code re usability.
- Using loops we can traverse over the elements of data structure (array of linked list)

There are three types of loops in C.

1. While loop
2. do While loop
3. For loop

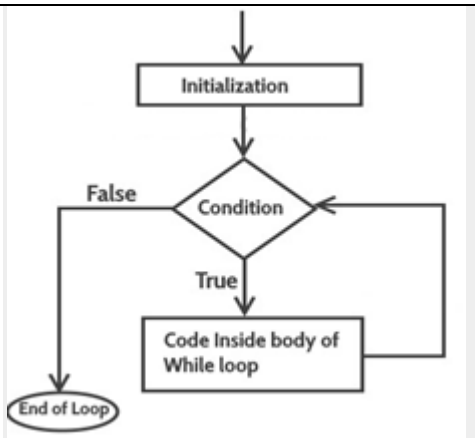
1. While Loop

It is Entry Controlled Loop . It means . In **while loop** First check the condition if condition is true then control goes inside the loop body other wise goes outside the body .The 'while' loop allows execution of statements inside block of loop only if condition in loop succeeds. The loop execution is terminated on the basis of test condition.

Loop is mainly consisted of three statements – **initialization expression, test expression, update expression**. The syntax of the three loops – For, while and do while mainly differs on the placement of these three statements.

Basic syntax to use 'while' loop is:

```
variable initialization;

while (condition to control loop)
{
statement 1;
statement 2;
..
iteration of variable;
}
```



In the above syntax :

- **Variable initialization** is the initialization of counter of loop before start of 'while' loop
- **Condition** is any logical condition that controls the number of times execution of loop statements
- **Iteration** is the increment/decrement of counter

Program:

```
#include <stdio.h>
int main()
{
  int count=1;
  while (count <= 4)
  {
       printf("%d ", count);
       count++;
  }
  return 0;
}
```

2. Do While Loop

It is also called **Exit Control Loop**(post tested loop). do-while loop allows execution of statements inside block of loop for one time for sure even if condition in loop fails.

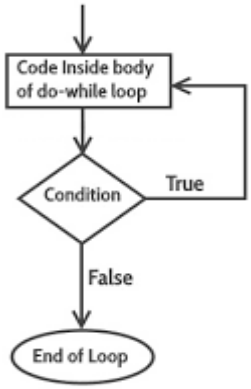
In do while loops also the loop execution is terminated on the basis of test condition. The main difference between do while loop and while loop is in do while loop the condition is tested at the end of loop body, i.e do while loop is exit controlled whereas the other two loops

are entry controlled loops.
Note: In do while loop the loop body will execute at least once irrespective of test condition.

Basic syntax to use 'do-while' loop is:

Syntax:

```
variable initialization;
do
 {
statement 1;
statement 2;
..
..
iteration of variable;
} while (condition to control loop);
```



- **Variable initialization** is the initialization of counter of loop before start of 'do-while' loop.
- **Condition** is any logical condition that controls the number of times execution of loop statements
- **Iteration** is the increment/decrement of counter

Program :

```
#include <stdio.h>
int main()
{
int j=0;
do
{
printf("Value of variable j is: %d\n", j);
j++;
}while (j<=3);
return 0;
}
```

3. For Loop

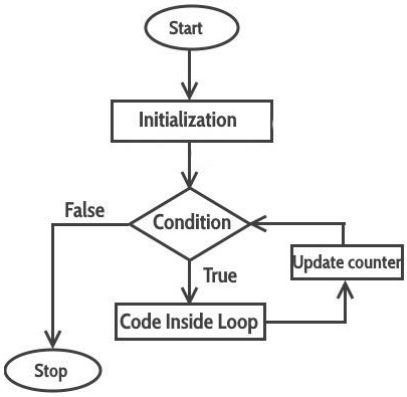
for-loops are **counter-controlled**, meaning that they are normally used whenever the number of iterations is known in advance.

for loop is a statement which allows code to be repeatedly executed.
For loop contains 3 parts Initialization, Condition and Increment or Decrements.
Basic syntax to use 'for' loop is:

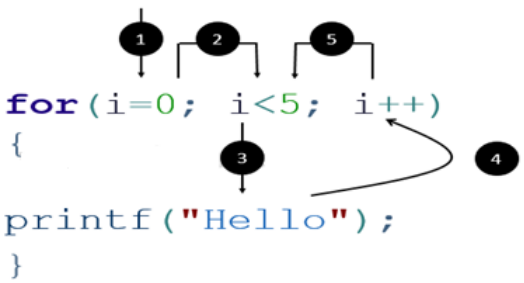
```
for (initialization expr; test expr; update expr)
{
    // body of the loop
    // statements we want to execute
}
```

```
for( a = 5;   a <= 10;   a++ )
```

Initilization — Condition — Increment (++) or Decrement (--)



- **Variable initialization** is the initialization of counter of loop.
- **Condition** is any logical condition that controls the number of times the loop statements are executed.
- **Iteration** is the increment/decrement of counter.
  - **Initialization Expression**: In this expression we have to initialize the loop counter to some value. for example: int i=1;
  - **Test Expression(condition)**: In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of loop and go to update expression otherwise we will exit from the for loop. For example: i <= 10;
  - **Update Expression(iteration)**: After executing loop body this expression increments/decrements the loop variable by some value. for example: i++;



Sequence of the execution will be as below –

Seq. No	Statement Flow	Explanation
01	Flow No 1 will be executed	i = 0
02	Flow No 2 will be executed	Condition Checking
03	Flow No 3 will be executed	True Condition
04	Flow No 4 will be executed	-
05	Flow No 5 will be executed	i = 1
06	Flow No 3 will be executed	True Condition
07	Flow No 4 will be executed	-
08	Flow No 5 will be executed	i = 2
09	Flow No 3 will be executed	True Condition
10	Flow No 4 will be executed	-
11	Flow No 5 will be executed	i = 3
12	Flow No 3 will be executed	True Condition
13	Flow No 4 will be executed	-
14	Flow No 5 will be executed	i = 4
15	Flow No 3 will be executed	True Condition
16	Flow No 4 will be executed	-
17	Flow No 5 will be executed	i = 5
18	Flow No 3 will be executed	False Condition

It is noted that when 'for' loop execution starts,

- first **variable initialization** is done, then condition is checked before execution of statements;
- if and only if **condition** is TRUE, statements are executed;

- after all statements are executed, **iteration** of counter of loop is done either increment or decrement.

Program 1 :

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

void main()
{
int i;
clrscr();
for(i=1;i<5;i++)
{
printf("\n%d",i);
}
getch();
}
```

Program 2 :

```
#include <stdio.h>

int main()
{
   int i, n, t1 = 0, t2 = 1, t3;

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

   printf("Fibonacci Series: ");

   for (i = 1; i <= n; ++i)
   {
      printf("%d, ", t1);
      t3 = t1 + t2;
      t1 = t2;
      t2 = t3;
   }
   return ();
}
```

Q4: Explain Nested For Loops in C with Examples

Nested loop means a loop statement inside another loop statement. That is why nested loops are also called as "**loop inside loop**".

Syntax for Nested For loop:

```
for ( initialization; condition; increment ) {
  for ( initialization; condition; increment ) {
     // statement of inside loop
  }
  // statement of outer loop
}
```

Example

Below program uses a nested for loop to print a 2D matrix of 3×3.

```
// C program that uses nested for loop to print a 2D matrix

#include <stdio.h>
#include <stdlib.h>
 int main()
{
   int i, j;
   int matrix[3][3] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
   printf("Given matrix is \n");
    // Print the matrix using nested loops
   for (i = 0; i < 3; i++) {
       for (j = 0; j < 3; j++)
         printf("%d ", matrix[i][j]);
       printf("\n");
   }
    return 0;
}
```

Output:

```
Given matrix is
1  2  3
4  5  6
7  8  9
```

Q 5: Explain Difference between for, while, do while loop statements

Sr. No	For loop	While loop	Do while loop
1.	Syntax: For(initialization; condition;updating), { . Statements; }	Syntax: While(condition), { . Statements; . }	Syntax: Do { . Statements; } While(condition);
2.	It is known as entry controlled loop	It is known as entry controlled loop.	It is known as exit controlled loop.
3.	If the condition is not true first time than control will never enter in a loop	If the condition is not true first time than control will never enter in a loop.	Even if the condition is not true for the first time the control will enter in a loop.
4.	There is no semicolon; after the condition in the syntax of the for loop.	There is no semicolon; after the condition in the syntax of the while loop.	There is semicolon; after the condition in the syntax of the do while loop.
5.	Initialization and updating is the part of the syntax.	Initialization and updating is not the part of the syntax.	Initialization and updating is not the part of the syntax

Jumping statements

Q6 : Discuss briefly about various jumping statements?

A: Jump statement makes the control jump to another section of the program unconditionally when encountered. It is usually sued to terminate the loop or switch case instantly or escape the execution of section of the program.

There are mainly 3 jump statements

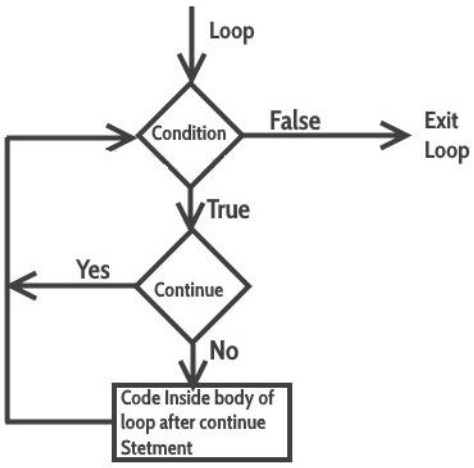
1. continue
2. break
3. goto

1.C – continue statement

The **continue statement** is used inside **loops**. When a continue statement is encountered inside a loop, control jumps to the beginning of the loop for next iteration, skipping the execution of statements inside the body of loop for the current iteration.
Syntax:

```
continue;
```

Flow diagram of continue statement



Example: Use of continue in While loop

In this example we are using `continue` inside while loop. When using while or do-while loop you need to place an increment or decrement statement just above the `continue` so that the counter value is changed for the next iteration. For example, if we do not place counter– statement in the body of "if" then the value of counter would remain 7 indefinitely.

Example

```
#include <stdio.h>
int main()
{
    int counter=10;
    while (counter >=0)
    {
        if (counter==7)
        {
             counter--;
             continue;
        }
        printf("%d  ", counter);
        counter--;
    }
    return 0;
}
```

Output:

```
10 9 8 6 5 4 3 2 1 0
```

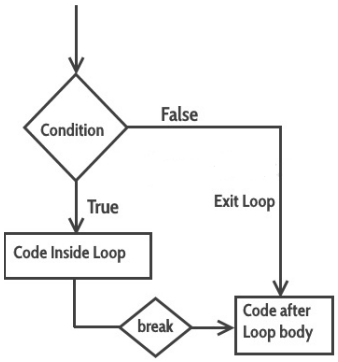
The print statement is skipped when counter value was 7.

2. C – break statement

The break statement is used inside loops and switch case.

1. It is used to come out of the loop instantly. When a break statement is encountered inside a loop, the control directly comes out of loop and the loop gets terminated. It is used with **if statement**, whenever used inside loop.
2. This can also be used in switch case control structure. Whenever it is encountered in switch-case block, the control comes out of the switch-case(see the example below).

Flow diagram of break statement



Syntax:

```
break;
```

Example – Use of break in a while loop

```
#include <stdio.h>
int main()
{
     int num =0;
     while(num<=100)
     {
        printf("value of variable num is: %d\n", num);
        if (num==2)
        {
            break;
        }
        num++;
     }
     printf("Out of while-loop");
     return 0;
}
```

Output:

```
value of variable num is: 0
value of variable num is: 1
value of variable num is: 2
Out of while-loop
```

3.C – goto statement

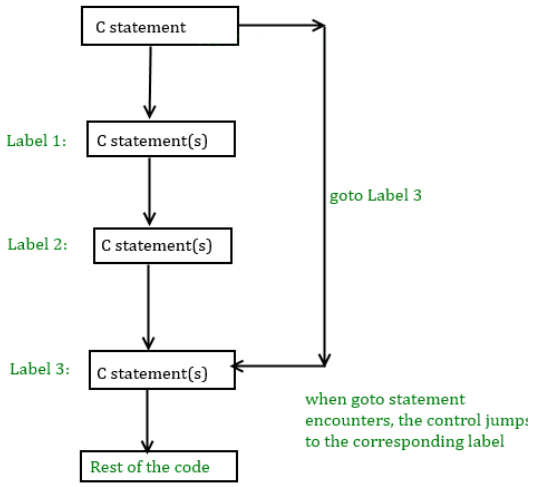
The goto statement is rarely used because it makes program confusing, less readable and complex. Also, when this is used, the control of the program won't be easy to trace, hence it makes testing and debugging difficult.

When a goto statement is encountered in a C program, the control jumps directly to the label mentioned in the goto stateemnt

Syntax of goto statement in C

```
goto label_name;
..
..
label_name: C-statements
```

Flow Diagram of goto



Example of goto statement

```
#include <stdio.h>
int main()
{
   int sum=0;
   for(int i = 0; i<=10; i++){
       sum = sum+i;
       if(i==5){
          goto addition;
       }
   }
   addition:
   printf("%d", sum);
   return 0;
}
```

Output:

```
15
```

Explanation: In this example, we have a label `addition` and when the value of i (inside loop) is equal to 5 then we are jumping to this label using goto. This is reason the sum is displaying the sum of numbers till 5 even though the loop is set to run from 0 to 10.

IMPORTANT QUESTIONS

1. Explain Conditional Statements (OR) Decision making in "C" language?

   (OR)

   a) Write the syntax of if statement and explain
   b) Explain if else statement with example
   c) Explain nested if else statement with example
   d) Explain Else- if ladder statement with as example

2. Explain switch statement with an example

3. Discuss briefly about various Jumping statements

   (OR)

   a) What is the use of **break** statement explain with example?
   b) What is the use of **continue** statement explain with an example?
   c) What is the use of **goto** statement explain with an example?

4. What are the looping statement in C?

   (OR)

   a) Discuss in detail about while loop with an example program?
   b) Discuss in detail about do-while loop with an example program?
   c) Discuss in detail about for loop with an example program?
   d) Discuss in detail about Nested for loop with an example program?

5. Explain difference between while loop and do while loop and for loop

UNIT-III

FUNCTIONS, ARRAYS AND STRINGS FUNCTIONS

SYLLABUS

- Definition and declaration of functions
- Function proto type
- return statement
- types of functions
- formatted
- Built in functions:
    - Mathematical functions
    - String functions
    - Character functions
    - Date functions.
- User defined functions:
    - Introduction
    - Need for user defined functions
    - Elements of functions
    - Function call
    - call by value
    - call by reference
    - Recursive functions.
- Arrays:
    - Introduction
    - Defining an array
    - Initializing an array
    - characteristics of an array
    - One dimensional array
    - Two dimensional array
    - Multi dimensional array.
- Strings:
    - Introduction
    - Declaring and initializing string
    - Reading and Writing strings
    - String standard functions

Q 1) What is a function in C language? Write uses of function.

A) A function is a set of statements that take inputs, do some specific computation and produces output.
The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.

Usage : Functions in a code can be (used)called several times, whenever we want to use them it returns the corresponding output.
Advantages:
1. Saves time
2. Optimises code.

A large C program is divided into basic building blocks called C function. C function contains set of instructions enclosed by "{ }" which performs specific operation in a C program. Actually, Collection of these functions creates a C program.

USES OF C FUNCTIONS:

- C functions are used to avoid rewriting same logic/code again and again in a program.
- There is no limit in calling C functions to make use of same functionality wherever required.
- We can call functions any number of times in a program and from any place in a program.
- A large C program can easily be tracked when it is divided into functions.
- The core concept of C functions are, re-usability, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.

C functions can be classified into two categories,

1. **Library functions OR Built-in function**
2. **User-defined functions**

LIBRARY FUNCTIONS/BUILT-IN FUNCTIONS

Q2) What are library functions? Explain types of library functions?

A) Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library.

- Each library function in C performs specific operation.

- We can make use of these library functions to get the pre-defined output instead of writing our own code to get those outputs.
- These library functions are created by the persons who designed and created C compilers.
- All C standard library functions are declared in many header files which are saved as file_name.h.
- Actually, function declaration, definition for macros are given in all header files.
- We are including these header files in our C program using "#include<file_name.h>" command to make use of the functions those are declared in the header files.
- When we include header files in our C program using "#include<filename.h>" command, all C code of the header files are included in C program. Then, this C program is compiled by compiler and executed.

Types of Library functions are:

1. C – math.h source code(mathematical functions)

All C inbuilt functions which are declared in math.h header file are given below. "math.h" header file supports all the mathematical related functions in C language. All the arithmetic functions used in C language are given below.

Function	Description
floor ( )	This function returns the nearest integer which is less than or equal to the argument passed to this function.
round ( )	This function returns the nearest integer value of the float/double/long double argument passed to this function. If decimal value is from ".1 to .5", it returns integer value less than the argument. If decimal value is from ".6 to .9", it returns the integer value greater than the argument.
ceil ( )	This function returns nearest integer value which is greater than or equal to the argument passed to this function.
sin ( )	This function is used to calculate sine value.
cos ( )	This function is used to calculate cosine.
cosh( )	This function is used to calculate hyperbolic cosine.

exp( )	This function is used to calculate the exponential "e" to the x^{th} power.
tan ( )	This function is used to calculate tangent.
tanh( )	This function is used to calculate hyperbolic tangent.
sinh( )	This function is used to calculate hyperbolic sine.
log ( )	This function is used to calculates natural logarithm.
log10 ( )	This function is used to calculates base 10 logarithm.
sqrt( )	This function is used to find square root of the argument passed to this function.
pow ( )	This is used to find the power of the given number.
trunc.(.)	This function truncates the decimal value from floating point value and returns integer value.

2. C – string.h source code(string functions)

All C inbuilt functions which are declared in string.h header file are given below. The source code for string.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN STRING.H FILE:

String functions	Description
strcat( )	Concatenates str2 at the end of str1
strncat( )	Appends a portion of string to another
strcpy( )	Copies str2 into str1
strncpy( )	Copies given number of characters of one string to another
strlen( )	Gives the length of str1
strcmp( )	Returns 0 if str1 is same as str2. Returns <0 if strl< str2. Returns >0 if str1 > str2

3. C – time.h source code(date functions)

All C inbuilt functions which are declared in time.h header file are given below. The source code for time.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN TIME.H FILE:

Below are the list of time related fuctions in C programming language.

Functions	Description
setdate()	This function used to modify the system date
getdate()	This function is used to get the CPU time
clock()	This function is used to get current system time
time()	This function is used to get current system time as structure
difftime()	This function is used to get the difference between two given times
strftime()	This function is used to modify the actual time format
mktime()	This function interprets tm structure as calendar time
localtime()	This function shares the tm structure that contains date and time informations
gmtime()	This function shares the tm structure that contains date and time informations
ctime()	This function is used to return string that contains date and time informations
asctime()	Tm structure contents are interpreted by this function as calendar time. This time is converted into string.

4. C – ctype.h source code(character functions)

All C inbuilt functions which are declared in ctype.h header file are given below.

LIST OF INBUILT C FUNCTIONS IN CTYPE.H FILE:

- "ctype.h" header file support all the below functions in C language. Click on each function name below for detail description and example programs.

Functions	Description
isalpha()	checks whether character is alphabetic
isdigit()	checks whether character is digit
isalnum()	Checks whether character is alphanumeric
isspace()	Checks whether character is space
islower()	Checks whether character is lower case
isupper()	Checks whether character is upper case
isxdigit()	Checks whether character is hexadecimal
iscntrl()	Checks whether character is a control character
isprint()	Checks whether character is a printable character
ispunct()	Checks whether character is a punctuation
isgraph()	Checks whether character is a graphical character
tolower()	Checks whether character is alphabetic & converts to lower case
toupper()	Checks whether character is alphabetic & converts to upper case

User Defined functions

Q 3) What is User defined functions? Explain need of user defined functions.

A)A large C program is divided into basic building blocks called C function. C function contains set of instructions enclosed by "{ }" which performs specific operation in a C program. Actually, Collection of these functions creates a C program.

C functions can be classified into two categories namely library functions and user defined functions. The main distinction between these two categories is that library functions are not required to be written by us whereas a user defined function has to be developed by the user at the time of writing a program.

Whenever we develop our program we write **main ()** function. This is a user defined function. Every program must have a main function to indicate where the program has to begin its execution.

Need of user defined functions:

- Every program must have a main function.
- It is possible to code any program utilizing only main function, it leads to a number of problems
- The program may become too large and complex and as a result the task of debugging, testing, and maintaining becomes difficult.
- If a program is divided into functional parts, then each part may be independently coded and later combined into a single unit
- These subprograms called 'functions' are much easier to understand, debug, and test
- There are times when some types of operation or calculation is repeated at many points throughout a program
- In such situations, we may repeat the program statements whenever they are needed
- Another approach is to design a function that can be called and used whenever required
- This saves both time and space.
- This sub-sectioning approach clearly results in a number of advantages
- It facilitates top-down modular programming.
- The length of a source program can be reduced by using functions at appropriate places.

- This factor is particularly critical with microcomputers where memory space is limited –
- It is easy to locate and isolate a faulty function for further investigations –
- A function may be used by many other programs, this means that a C program can build on what others have already done, instead of starting over, from scratch.

A MULTI-FUNCTION PROGRAM

• A function is a self-contained block of code that performs a particular task
• Once a function has been designed and packed, it can be treated as a 'black box' that takes some data from the main program and returns a value
• The inner details are invisible to the rest of the program

Example:

```
#include<conio.h>
#include<stdio.h>
void main()
 {
printline();
printf("This illustrated the use of C functions n");
printline();
 }
printline()
 {
inti;
for(i=1; i<40; i++)
pintf("-");
printf("/n");
 }
```

• Functions are classified as one of the derived data types in C
• Can define functions and use them like any other variables in C programs.

• Similarities between functions and variables in C – Both function name and variable names are considered identifiers and therefore they must adhere to the rules for identifiers.
– Like variables, functions have types (such as int) associated with them
– Like variables, function names and their types must be declared and defined before they are used in a program

Q4) What is a function? Explain function definition and function declaration(function prototype)
OR
Explain Elements of function

A)A large C program is divided into basic building blocks called C function. C function contains set of instructions enclosed by "{ }" which performs specific operation in a C program.In order to make use of a user-defined function, we need to establish **three** elements that are related to functions.

- Function definition
- Function call
- Function declaration

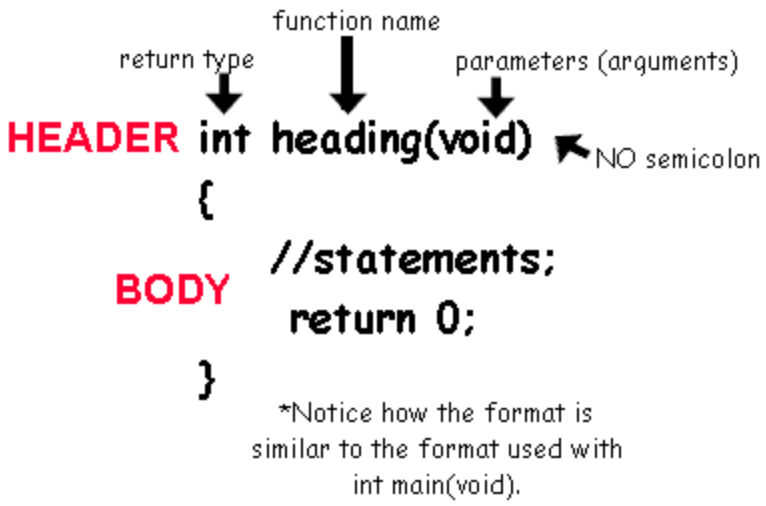
FUNCTION DEFINITION

Function definition: The function definition is an independent program module that is specially written to implement to the requirements of the function.

the general syntax of function definition is,

```
returntypefunctionName(type1 parameter1, type2 parameter2,...)
{
   // function body goes here
}
```

The first line *returntype* **functionName(type1 parameter1, type2 parameter2,...)** is known as **function header** and the statement(s) within curly braces is called **function body**.



Ex:**-int max(int num1, int num2)**

```
{
    /* local variable declaration */
int result;

    if (num1 > num2)
      result = num1;
  else
        result = num2;
   return result;
}
```

Note: While defining a function, there is no semicolon (;) after the parenthesis in the function header, unlike while declaring the function or calling the function.

functionbody

The function body contains the declarations and the statements(algorithm) necessary for performing the required task. The body is enclosed within curly braces **{...}** and consists of three parts.

- **local** variable declaration (if required).
- **function statements** to perform the task inside the function.
- a **return** statement to return the result evaluated by the function (if return type is void, then no return statement is required).

Function declaration (Function prototype)

The program or a function that has called a function is referred to as the calling function or calling program. The calling program should declare any function that is to be used later in the program. This is known as the function declaration.

General syntax for function declaration is,

Returntype functionName(type1 parameter1, type2 parameter2,...);

Ex: **int max(int num1, int num2);**

Like any variable or an array, a function must also be declared before its used. Function declaration informs the compiler about the function name, parameters is accept, and its return type. The actual body of the function can be defined separately. It's also called as **Function Prototyping**. Function declaration consists of 4 parts.

- returntype
- function name
- parameter list
- terminating semicolon

returntype

When a function is declared to perform some sort of calculation or any operation and is expected to provide with some result at the end, in such cases, a return statement is added at the end of function body. Return type specifies the type of value(int, float, char, double) that function is expected to return to the program which called the function.

Note: In case your function doesn't return any value, the return type would be void.

---

functionName

Function name is an identifier and it specifies the name of the function. The function name is any valid C identifier and therefore must follow the same naming rules like other variables in C language.

---

parameter list

The parameter list declares the type and number of arguments that the function expects when it is called. Also, the parameters in the parameter list receives the argument values when the function is called. They are often referred as **formal parameters**.

Function Call

Function call: In order to use the function we need to invoke it at a required place in the program. This is known as the function call.

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

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.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

Syntax: function_name(Parameter1 ,Parameter2 ,....Parameter n);

Ex:- **max(a,b);** is function call .

a,b are **parameters passed to function 'sum'**

1. Call a C function just by writing function name with opening and closing round brackets followed with semicolon.

2. If we have to supply parameters then we can write parameters inside pair of round brackets.

3. Parameters are optional.

Note: Function call should be made by ending Semicolon.

Sample program 1:

```
#include <stdio.h>

Int max(int num1, int num2);          /* function declaration */

int main ()
{
   /* local variable definition */
int a = 100;
int b = 200;
int c;
  ret = max(a, b);   /* function call  to get max value */
printf( "Max value is : %d\n", ret );
    return 0;
```

```
}

Int max(int num1, int num2)                    /* function returning the max between
two numbers */

{
  /* local variable declaration */
int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;
   return result;
}
```

Sample program 2:

```
#include<stdio.h>
#include<conio.h>
int sum(int,int);
void main()
{
inta,b,c;
printf("\nEnter the two numbers : ");
scanf("%d%d",&a,&b);
c = sum(a,b);
printf("\nAddition of two number is : %d",c);
getch();
}
int sum (int num1,int num2)
{
int num3;
num3 = num1 + num2 ;
return(num3);
}
```

The output will be like:-

```
Enter the two numbers : 12 12
Addition of two number is : 24
```

4) What is function call? Explain types of function call.

A)In order to use the function we need to invoke it at a required place in the program. This is known as the function call.

There are two ways to pass arguments/parameters to function calls.

1) ***call by value***

2) ***call by reference*.**

The major difference between call by value and call by reference is that in call by value a copy of actual arguments is passed to respective formal arguments. While, in call by reference the location (address) of actual arguments is passed to formal arguments, hence any change made to formal arguments will also reflect in actual arguments.

In C, all function arguments are passed "by value" because C does not support references like C++ and Java do. In C, the calling and called functions do not share any memory -- they have their own copy and the called function cannot directly alter a variable in the calling function; it can only alter its private, temporary copy.

The ***call by value*** scheme is an asset, however, not a liability. It usually leads to more compact programs with fewer extraneous variables, because parameters can be treated as conveniently initialized local variables in the called routine. Yet, there are some cases where we need ***call by reference***:

1. The called function communicates to the calling function only through return statement and return statement can only send only one value back to the calling function. If there are more than one value we want to alter, call by reference is required
2. If the size of data is large , copying actual arguments to formal arguments could be a time consuming operation and occupies more memory.

The ***call by value*** does not address above cases, hence we need call by reference. To achieve call by reference functionality in C language the calling function provides the address of the variable to be set (technically a pointer to the variable), and the called function declares the parameter to be a pointer and access the variable indirectly through it. Since the address of the argument is passed to the function, code within the called function can change the value of the actual arguments.

While studying call by value and call by reference in C it is important to note that the story is different for arrays. When the name of an array is used as an argument, the value passed to the function is the location or address of the beginning of the array --there is no copying of array elements. By subscripting this value, the function can access and alter any element of the actual array.

Difference between Call by Value and Call by Reference

Difference between *call by value* and *call by reference*	
call by value	**call by reference**
In *call by value*, a copy of actual arguments is passed to formal arguments of the called function and any change made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function.	In *call by reference*, the location (address) of actual arguments is passed to formal arguments of the called function. This means by accessing the addresses of actual arguments we can alter them within from the called function.
In call by value, actual arguments will remain safe, they cannot be modified accidentally.	In *call by reference*, alteration to actual arguments is possible within from called function; therefore the code must handle arguments carefully else you get unexpected results.

Example using Call by Value

The classic example of wanting to modify the caller's memory is a swapByValue() function which exchanges two values. For C uses call by value, the following version of *swap* swapByValue() will not work...

```
#include <stdio.h>
 voidswapByValue(int,int);/* Prototype */

intmain()/* Main function */
{
int n1 =10, n2 =20;

/* actual arguments will be as it is */
swapByValue(n1, n2);
printf("n1: %d, n2: %d\n", n1, n2);
}

voidswapByValue(int a,int b)
{
int t;
  t = a;
a = b;
```

```
 b = t;
}
```

OUTPUT

```
======
```

n1:10, n2:20

The swapByValue() does not affect the arguments n1 and n2 in the calling function it only operates on a and b local to swapByValue() itself. This is a good example of how local variables behave.

Example using Call by Reference

In call by reference, to pass a variable n as a reference parameter, the programmer must pass a pointer to n instead of n itself. The formal parameter will be a pointer to the value of interest. The calling function will need to use & to compute the pointer of actual parameter. The called function will need to dereference the pointer with *where appropriate to access the value of interest. Here is an example of a correct *swap* swapByReference() function. So, now you got the difference between call by value and call by reference!

```
#include <stdio.h>
 voidswapByReference(int*,int*);/* Prototype */
 intmain()/* Main function */
{
int n1 =10, n2 =20;
   /* actual arguments will be altered */
swapByReference(&n1,&n2);
printf("n1: %d, n2: %d\n", n1, n2);
}

voidswapByReference(int*a,int*b)
{
int t;
  t =*a;*a =*b;*b = t;
}
```

OUTPUT

```
======
```

n1:20, n2:10

Q) What is return statement in “C” language?

A)A function is a collection of statements grouped together to do some specific task. It may return a value. However, in no case a function will return more than one value.

The return statement is used to return some value or simply pass the control to the calling function.

The return statement can be used in the following two ways.

- return;
- return expression;

first form :**return;**
The first form of the return statement is used to terminate the function and pass the control to the calling function. No value from the called function is returned when this form of the return statement is used.

Second form: **return expression;**

- return keyword transfers program control to the caller.
- expression is optional, used to return result of a valid C expression to the caller.

Ex:int sum (int num1,int num2)
{
int num3;
num3 = num1 + num2 ;
return(num3);
}

Q) What is a recursive function? explain with example.

A) In C programming language, function calling can be made from main() function, other functions or from same function itself. The recursive function is definedd as follows...

A function called by itself is called recursive function.

The recursive functions should be used very carefully because, when a function called by itself it enters into infinite loop. And when a function enters into the infinite loop, the function execution never gets completed. We should define the condition to exit from the function call so that the recursive function gets terminated.

When a function is called by itself, the first call remains under execution till the last call gets invoked. Every time when a function call is invoked, the function returns the execution control to the previous function call.

```
Example: void main()
            {
        Printf(“welcome to C language”);
        main();
             }
```

Program using factorial of a given number using recursive:

```
#include<stdio.h>
longintfact(int n);
intmain()
{
int n;
printf("Enter a positive integer: ");
scanf("%d",&n);
printf("Factorial of %d = %ld", n,
fact(n));
return0;
}
longintfact(int n)
{
if(n >=1)
return n*fact(n-1);
else
return1;
}
```

Program for factorial of a given number

```
#include <stdio.h>
intmain()
{
int c, n, fact = 1;
   printf("Enter a number to calculate its
factorial\n");
scanf("%d", &n);
   for (c = 1; c <= n; c++)
    fact = fact * c;
   printf("Factorial of %d = %d\n", n, fact);
  return 0;
}
```

Q) What is an Array?

A) In C language, arrays are reffered to as structured data types. an array is a collection of one or more values of the same type. Each value is called an element of the array. The elements of the array share the same variable name but each element has its own unique index number (also known as a subscript). An array can be of any type, For example: int, float, char etc. If an array is of type intthen it's elements must be of type int only.

Example where arrays are used,

- to store list of Employee or Student names,
- to store marks of students,
- or to store list of numbers or characters etc.

Since arrays provide an easy way to represent data, it is classified amongst the data structures in C. Other data structures in c are **structure**, **lists**, **queues**, **trees** etc.

TYPES OF C ARRAYS:

There are 2 types of C arrays. They are,

1. One dimensional array
2. Multi dimensional array
   - Two dimensional array
   - Three dimensional array
   - four dimensional array etc…

1)One dimensional array : Arrays can be single or multidimensional. The number of subscript or index determines dimensions of the array. An array of one dimension is known as a one-dimensional array or 1-D array,

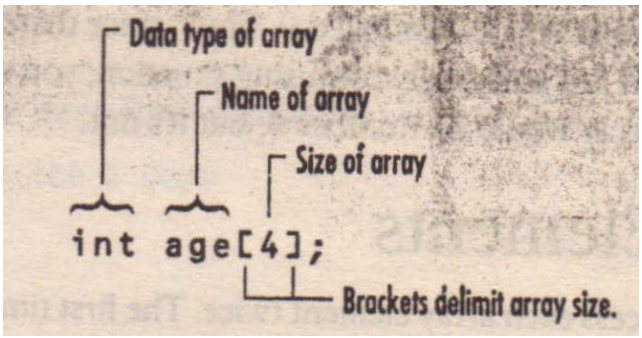
Declaration of an Array:

Like any other variable, arrays must be declared before they are used. Array declaration syntax is very simple. The syntax is the same as for a normal variable declaration except the variable name should be followed by subscripts to specify the size of each dimension of the array.

General form of array declaration is,

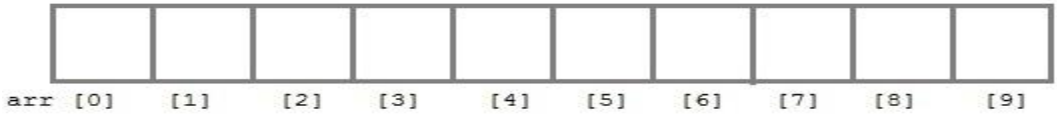
Syntax: data-type variable-name [size];

datatype: It denotes the type of the elements in the array.
variable_name: it is the name given to an array. It must be any valid identifier.
size: It is the number of elements an array can hold.



/* Example of array declaration */

intarr[10];



Here int is the data type, arr is the name of the array and 10 is the size of array. It means array arr can only contain 10 elements of int type.

Index of an array starts from 0 to **size-1** i.e first element of arr array will be stored at arr[0]address and the last element will occupy arr[9].

Initialization of an Array

When an array is declared inside a function the elements of the array have garbage value. If an array is global or static, then its elements are automatically initialized to 0. If you want you can explicitly initialize elements of the array at the time of declaration. The following is the syntax to do the same:

Syntax: datatype array_name[size] = { val1, val2, val3, ... valN };

data type is the type of elements of an array.
array_name is the variable name, which must be any valid identifier.
size is the size of the array.
val1, val2 ... are the constants known as initializers.

Each value is separated by a comma(,) and then there is a semi-colon(;) after the closing curly brace (}).

Here is are some examples:
floattemp[]={12.3,4.1,3.8,9.5,4.5};// an array of 5 float type
intarr[]={11,22,33,44,55,66,77,88,99};// an array of 5 float type

If the number of initializers is less than the specified size then the remaining elements of the array are assigned value of 0.
floattemp[5]={12.3,4.1};
here the size of temp array is 5 but there are only two initializers. After this initialization the elements of the array are as follows:

temp[0] is 12.3 temp[1] is 4.1 temp[2] is 0 temp[3] is 0 temp[4] is 0

If the number of initializers is greater than the size of the array then, the compiler will report an error. For example:

intnum[5]={1,2,3,4,5,6,7,8}// error

sample program 1:

```
#include<stdio.h>
void main()
{
intarr[4];
inti, j;
printf("Enter array element");
for(i=0;i<4;i++)
{
scanf("%d",&arr[i]);
}
for(j =0; j <4;j++)
{
printf("%d\n",arr[j]);
}
}
```

Program 2 :The following program uses for loop to take input and print elements of a 1-D array.

```
#include<stdio.h>
intmain()
{
intarr[5],i;
for(i=0;i<5;i++)
{
printf("Enter a[%d]: ",i);
scanf("%d",&arr[i]);
}
printf("\nPrinting elements of the array: \n\n");
for(i=0;i<5;i++)
{
```

```
printf("%d ",arr[i]);
}
// signal to operating system program ran fine
return0;
}
```

Expected Output:

```
Enter a[0]: 11
Enter a[1]: 22
Enter a[2]: 34
Enter a[3]: 4
Enter a[4]: 34

Printing elements of the array:

11 22 34 4 34
```

**How it works ? **

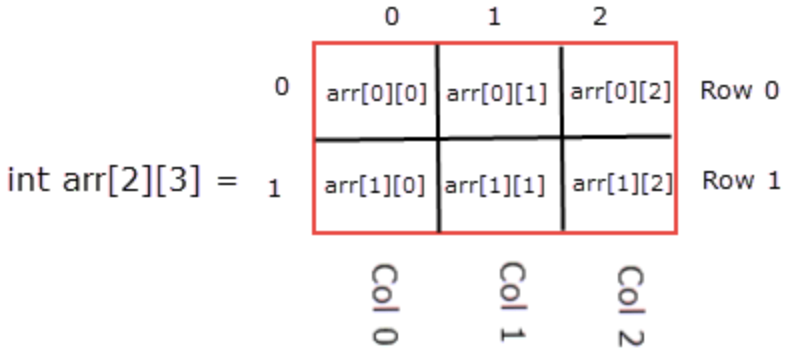
In Line 5, we have declared an array of 5 integers and variable i of type int. Then a for loop is used to enter five elements into an array. In scanf() we have used & operator (& is also known as the address of operator) on element arr[i] of an array, just like we had done with variables of type int, float, char etc. Line 13 prints "Printing elements of the array" to the console. The second for loop prints all the elements of an array one by one.

Two dimentionalarray :

An array of arrays is known as 2D array. The two dimensional (2D) array in C programming is also known as matrix. A matrix can be represented as a table of rows and columns.
The syntax declaration of 2-D array is not much different from 1-D array. In 2-D array, to declare and access elements of a 2-D array we use 2 subscripts instead of 1.

Syntax: datatype array_name [ROW][COL];

The total number of elements in a 2-D array is ROW*COL. Let's take an example.
Ex :intarr**[2][3];**
This array can store 2*3=6 elements. You can visualize this 2-D array as a matrix of 2 rows and 3 columns.



A conceptual representation of 2-D array

The individual elements of the above array can be accessed by using two subscript instead of one. The first subscript denotes row number and second denotes column number. As we can see in the above image both rows and columns are indexed from 0. So the first element of this array is at arr[0][0] and the last element is at arr[1][2]. Here are how you can access all the other elements:arr[0][0] - refers to the first element arr[0][1] - refers to the second element arr[0][2] - refers to the third element arr[1][0] - refers to the fourth element arr[1][1] - refers to the fifth element arr[1][2] - refers to the sixth element

Processing elements of a 2-D array

To process elements of a 2-D array, we use two nested loop. The outer for loop to loop through all the rows and inner for loop to loop through all the columns. The following program will clear everything.

```
#include<stdio.h>
#define ROW 3
#define COL 4

intmain()
{
intarr[ROW][COL],i, j;

for(i=0;i< ROW;i++)
{
for(j =0; j < COL;j++)
{
printf("Enter arr[%d][%d]: ",i, j);
```

```
scanf("%d",&arr[i][j]);
}
}

printf("\nEntered 2-D array is: \n\n");

for(i=0;i< ROW;i++)
{
for(j =0; j < COL;j++)
{
printf("%3d ",arr[i][j]);
}
printf("\n");
}
// signal to operating system everything works fine
return0;
}
```

Initializing 2-D array

Initialization of 2-D array is similar to a 1-D array. For e.g:

First Approach

intEmployees[4][3] = { 10, 20, 30, 15, 25, 35, 22, 44, 66, 33, 55, 77 };
First three elements will be 1st row, second three elements will be 2nd row, next 3 elements will be 3rd row and last 3 element will be 4th row. Here we divided them into 3 because our column size = 3.

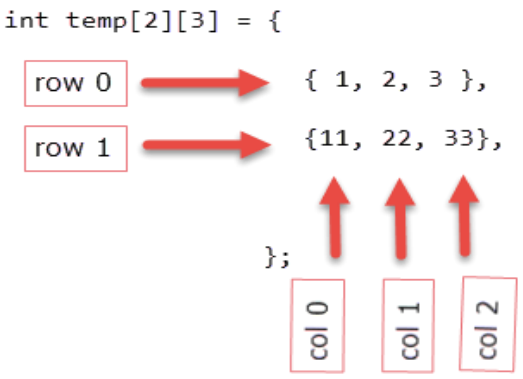
The above statement can also be written as
intEmployees[4][3] = { {10, 20, 30}, {15, 25, 35}, {22, 44, 66}, {33, 55, 77} };

Here, We surrounded each row with curly braces ({}). It is always good practice to use the curly braces to separate the rows.

Second Approach

intEmployees[ ][ ] = { {10, 20, 30}, {15, 25, 35}, {22, 44, 66}, {33, 55, 77} };

Here, We did not mention the row size and column size but the compiler is intelligent enough to calculate the size by checking the number of elements inside the row and column.
We can also write
intEmployees[ ][3] = { {10, 20, 30}, {15, 25, 35}, {22, 44, 66}, {33, 55, 77} };
inttemp[2][3]={ {1,2,3},// row 0
{11,22,33}// row 1
};



After this initialization, each element is as follows:
temp[0][0] : 1
temp[0][1] : 2
temp[0][2] : 3
temp[1][0] : 11
temp[1][1] : 22
temp[1][2] : 33

Consider another initialization.
intmy_arr[4][3]={ {10},{77,92},{33,89,44},{12,11} };
The size of my_arr is 4*3=12 , but in the initialization, we have only specified the value of 8 elements. In such cases, the remaining elements will be given the value of 0.
The individual elements are as follows:
my_arr[0][0] : 10
my_arr[0][1] : 0
my_arr[0][2] : 0

my_arr[1][0] : 77
my_arr[1][1] : 92
my_arr[1][2] : 0

my_arr[2][0] : 33
my_arr[2][1] : 89
my_arr[2][2] : 44

my_arr[3][0] : 12
my_arr[3][1] : 11
my_arr[4][2] : 0

In 2-D arrays, it is optional to specify the first dimension but the second dimension must always be present. This works only when you are declaring and initializing the array at the same time.

For example:

```
int  d[][3]={  {13,23,34},  {15,27,35} };
```

is same as

```
int  d[2][3]={  {13,23,34}, {15,27,35}};
```

sample program for initialization

```
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
for(i=0;i<4;i++){
 for(j=0;j<3;j++){
   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
return 0;
}
```

Accessing Two Dimensional Array Elements

We can access the Two Dimensional Array in C Programming elements using indexes. Using index we can access or alter/change each and every individual element present in the array separately. Index value starts at 0 and end at n-1 where n is the size of a row or column. For example, if an Array_name[8][5] will stores 8 row elements and 5 column elements. To access or alter 1st value use Array_name[0][0], to access or alter 2nd row 3rd column value then use Array_name[1][2] and to access the 8th row 5th column then use Array_name[7][4].

Transpose of a matrix:

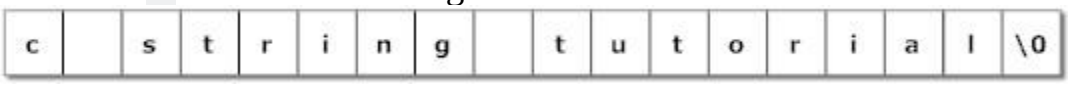
```
#include <stdio.h>
 intmain()
{
int m, n, c, d, matrix[10][10], transpose[10][10];

printf("Enter the number of rows and columns of matrix\n");
scanf("%d%d", &m, &n);

printf("Enter elements of the matrix\n");

   for (c = 0; c < m; c++)
for(d = 0; d < n; d++)
scanf("%d", &matrix[c][d]);

   for (c = 0; c < m; c++)
for( d = 0 ; d < n ; d++ )
        transpose[d][c] = matrix[c][d];

printf("Transpose of the matrix:\n");

   for (c = 0; c < n; c++)
{
     for (d = 0; d < m; d++)
printf("%d\t", transpose[c][d]);
printf("\n");
   }

   return 0;
}
```

STRINGS

Q) What is a string? Explain declaration and initialization of string?

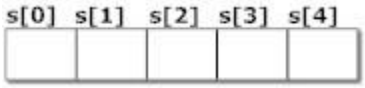
A) In C programming, array of characters is called a string. A string is terminated by a null character /0. For example:
"c string tutorial"

Here, "c string tutorial" is a string. When, compiler encounter strings, it appends a null character /0 at the end of string.



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.
Using arrays
char s[5];

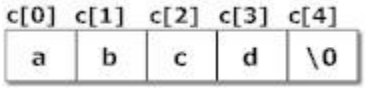


In the above syntax **"S"** is any name given to the string variable and size is used define the length of the string, i.e the number of characters strings will store. Please keep in mind that there is an extra terminating character which is the Null character ('\0') used to indicate termination of string which differs strings from normal character arrays.

Initialization of strings

In C, string can be initialized in a number of different ways.
For convenience and ease, both initialization and declaration are done in the same step.
Using arrays

```
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'};
```



The given string is initialized and stored in the form of arrays as above.

Sample program :

```
#include<stdio.h>
 intmain()
{
```

```
    // declare and initialize string
    char str[] = "Avanthi";

    // print string
    printf("%s",str);

    return 0;
}
```

We can see in the above program that strings can be printed using a normal printf statements just like we print any other variable. Unlike arrays we do not need to print a string, character by character. The C language does not provide an inbuilt data type for strings but it has an access specifier "**%s**" which can be used to directly print and read strings.

Q) What is a String? Explain Reading and writing of strings in C language.

A) Reading Strings from user

You can use the scanf() function to read a string like any other data types. Use "%S" as format specifier.
However, the scanf() function only takes the first entered word. The function terminates when it encounters a white space (or just space).

```
char c[20];
scanf("%s", c);
```

Writing string on screen :
We use printf() function to print a string . Use "%S" as format specifier.

```
Char c[20];
Scanf("%s",c);
Printf("%s",c);
```

```
#include<stdio.h>
 intmain()
{
```

```
    // declaring string
    char str[50];
        // reading string
    scanf("%s",str);
        // print string
    printf("%s",str);
     return 0;
}
```

You can see in the above program that string can also be read using a single scanf statement. Also you might be thinking that why we have not used the '&' sign with string name 'str' in scanf statement! To understand this you will have to recall your knowledge of scanf. We know that the '&' sign is used to provide the address of the variable to the scanf() function to store the value read in memory. As str[] is a character array so using str without braces '[' and ']' will give the base address of this string. That's why we have not used '&' in this case as we are already providing the base address of the string to scanf. And prinf() is used to print the string on the screen.

Another way of reading and writing asring

gets() & puts() to read and write a string in C.gets() & **puts()** functions defined into **stdio.h header file**.

- **gets()** reads string until **return/enter key** is pressed
  - Another method to read character string with white spaces from terminal is by using the gets()function.

Ex: char text[20];

```
gets(text);
printf("%s", text);
```

- **Return/Enter key** translates to new line character **'\n'** which in turn, translates to null character **'\0'**, thus last character stored in string is **'\0' as terminating character.**

- **puts()** writes string, followed by new line character **'\n'**, causes next output in new line.

Program using gets() and puts()

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

```
int main()
{
   char name[50];
   printf("Enter your name: ");
   gets(name); //reads string from user
   printf("Your name is: ");
   puts(name);  //displays string
   return 0;
}
```

Q) Explain string handling functions with example?

A) String Handling Functions

C language supports a large number of string handling functions that can be used to carry out many of the string manipulations. These functions are packaged in **string.h** library. Hence, you must include **string.h** header file in your programs to use these functions.

The following are the most commonly used string handling functions.

Method	Description
strcat()	**It is used to concatenate(combine) two strings**
strlen()	**It is used to show length of a string**
strrev()	**It is used to show reverse of a string**
strcpy()	**Copies one string into another**
strcmp()	**It is used to compare two string**
strupr()	**returns string characters in uppercase.**
strlwr()	**Returns string characters in lowercase**

strcat() function : It is used to concatenate(combine) two strings

syntax: strcat(s1,s2);
strcat("hello","world");
strcat() function will add the string **"world"** to **"hello"** i.e it will ouputhelloworld.

The strcat(first_string, second_string) function concatenates two strings and result is returned to first_string.

```
#include<stdio.h>
#include <string.h>
     int main()
{
    char ch[10]={'h', 'e', 'l', 'l', 'o', '\0'};
    char ch2[10]={'c', '\0'};
    strcat(ch,ch2);
    printf("Value of first string is: %s",ch);
   return 0;
}
```

---

strlen() function: strlen() function will return the length of the string passed to it. The strlen() function returns the length of the given string. It doesn't count null character '\0'.

Syntax: strlen(ch)

Ex: int j;
j =strlen("studytonight");
printf("%d",j);

program:

```
#include<stdio.h>
#include <string.h>
int main(){
char ch[20]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
  printf("Length of string is: %d",strlen(ch));
 return 0;
}
```

---

strcmp() function:

strcmp() function will return the ASCII difference between first unmatching character of two strings. The strcmp(first_string, second_string) function compares two string and returns 0 if both strings are equal.

Here, we are using *gets()* function which reads string from the console.

Syntax: strcmp(s1, s2)

Ex:

```
int j;
    j =strcmp("study","tonight");
printf("%d",j);
```

program:

```
#include<stdio.h>
#include <string.h>
int main(){
  char str1[20],str2[20];
  printf("Enter 1st string: ");
  gets(str1);//reads string from console
  printf("Enter 2nd string: ");
  gets(str2);
  if(strcmp(str1,str2)==0)
     printf("Strings are equal");
  else
     printf("Strings are not equal");
 return 0;
}
```

---

strcpy() function:

It copies the second string argument to the first string argument.

Syntax: strcpy(s2**,** s1**);**//copies string s1 to string s2

```
#include<stdio.h>
#include<string.h>
intmain()
{
   char s1[50];
   char s2[50];
strcpy(s1,“StudyTonight”);//copies “studytonight” to string s1
strcpy(s2, s1);//copies string s1 to string s2
printf(“%s\n”, s2);
return(0);
}
```

Strrev() function:

It is used to reverse the given string expression.

Syntax: strrev(s1);

Program:

```
#include<stdio.h>
intmain()
{
   char s1[50];
printf("Enter your string: ");
gets(s1);
printf("\nYour reverse string is: %s",strrev(s1));
return(0);
}
```

Strupr() function: It is used to convert string to uppercase

```
#include<stdio.h>
#include <string.h>
int main(){
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nUpper String is: %s",strupr(str));
 return 0;
}
```

Strlow() function: The strlwr(string) function returns string characters in lowercase

```
 #include<stdio.h>
 #include <string.h>
 int main()
 {
   char str[20];
   printf("Enter string: ");
   gets(str);//reads string from console
   printf("String is: %s",str);
   printf("\nLower String is: %s",strlwr(str));
  return 0;
 }
```

UNIT - IV

UNIT-IV

POINTERS, STRUCTURES AND UNIONS

Syllabus

Pointers

- Features of pointers-
- Declaration of Pointers-
- arithmetic operations with pointers

Structures

- Features of Structures
- Declaring and initializing of structures
- Structure within structure
- Array of structures
- Enumerated data type

Unions

- Definition and advantage of unions
- Comparison between structure and Union

UNIT - IV

Q) What is Pointer? Write about declarations of pointers.

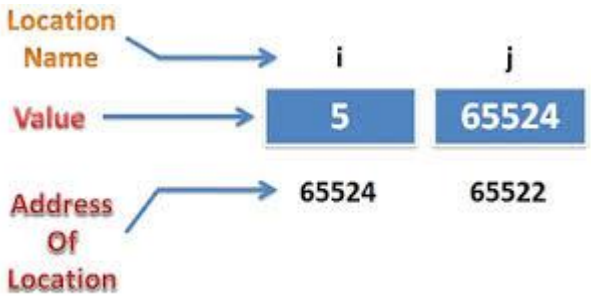
A)Pointers in C language is a variable that stores/points the address of another variable. A Pointer in C is used to allocate memory dynamically i.e. at run time. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

Pointers declaration:

Syntax : data_type *var_name;

Example : int *p; char *p;

Where, * is used to denote that "p" is pointer variable and not a normal variable.



KEY POINTS TO REMEMBER ABOUT POINTERS IN C:

- Normal variable stores the value whereas pointer variable stores the address of the variable.
- The content of the C pointer always be a whole number i.e. address.
- Always C pointer is initialized to null, i.e. int *p = null.
- The value of null pointer is 0.
- & symbol is used to get the address of the variable.
- symbol is used to get the value of the variable that the pointer is pointing to.
- If a pointer in C is assigned to NULL, it means it is pointing to nothing.
- Two pointers can be subtracted to know how many elements are available between these two pointers.
- But, Pointer addition, multiplication, division are not allowed.
- The size of any pointer is 2 byte (for 16 bit compiler).

UNIT - IV

SAMPLE PROGRAM:

```
#include <stdio.h>
int main()
{
  int *ptr, q;
  q = 50;
                /* address of q is assigned to ptr */
  ptr = &q;
               /* display q's value using ptr variable */

 printf("%d", *ptr);

 return 0;
}
```

Q3 Explain C - Pointer arithmetic

- A pointer in c is an address, which is a numeric value. Therefore, you can perform arithmetic operations on a pointer just as you can on a numeric value. There are four arithmetic operators that can be used on pointers: ++, --, +, and -
- To understand pointer arithmetic, let us consider that ptr is an integer pointer which points to the address 1000. Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer

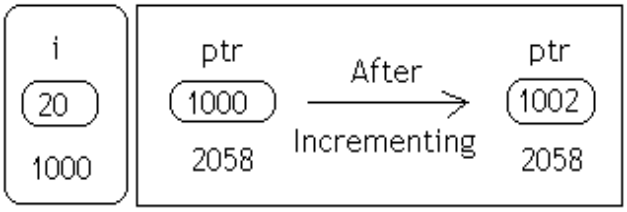
  ptr ++

- After the above operation, the ptr will point to the location 1002 because each time ptr is incremented, it will point to the next integer location which is 2 bytes next to the current location.
- If ptr points to a character whose address is 1000, then the above operation will point to the location 1001 because the next character will be available at 1001
- Incrementing Pointer is generally used in array because we have contiguous memory in array and we know the contents of next memory location.

UNIT - IV

- Incrementing Pointer Variable Depends Upon data type of the Pointer variable
- new value = current address + i * size_of(data type)
- Three way of writing increment pointer
  - Address + 1 = Address
  - Address++ = Address
  - ++Address = Address

Pictorial Representation :



Data Type	Older Address stored in pointer	Next Address stored in pointer after incrementing (ptr++)
int	**1000**	**1002**
float	**1000**	**1004**
char	**1000**	**1001**

UNIT - IV

Increment Integer Pointer

```
#include<stdio.h>

int main(){

int *ptr=(int *)1000;

ptr=ptr+1;
printf("New Value of ptr : %u",ptr);

return 0;
}
```

Output :

```
New Value of ptr : 1002
```

Q3 What is a structure?

A) C Structure is a collection of different data types which are grouped together and each element in a C structure is called member.

- If you want to access structure members in C, structure variable should be declared.
- Many structure variables can be declared for same structure and memory will be allocated for each separately.
- It is a best practice to initialize a structure to null while declaring, if we don't assign any values to structure members.

DIFFERENCE BETWEEN C VARIABLE, C ARRAY AND C STRUCTURE:

- A normal C variable can hold only one data of one data type at a time.
- An array can hold group of data of same data type.
- A structure can hold group of data of different data types and Data types can be int, char, float, double and long double etc.

Variable	Arrays	Structures
		Syntax : struct

UNIT - IV

Syntax: datatype variable1; Ex: int a; a = 20;	**Syntax:** datatype arrayname[size]; int a[3]; **Ex:** a[0] = 10; a[1] = 20; a[2] = 30;	{ datatype variable1; datatype variable2; } Ex: struct student { int a; char b[10]; }

Q4) What is a structure? How to declare a C structure? How to initialize a C structure? How to access the members of a C structure?

A) C Structure is a collection of different data types which are grouped together and each element in a C structure is called member

Creating a structure:

We use **struct** keyword to create a **structure in C**. The struct keyword is a short form of **structured data type.**

Here struct_name can be anything of your choice. Members data type can be same or different. Once we have declared the structure we can use the struct name as a data type like int, float etc.

Syntax:

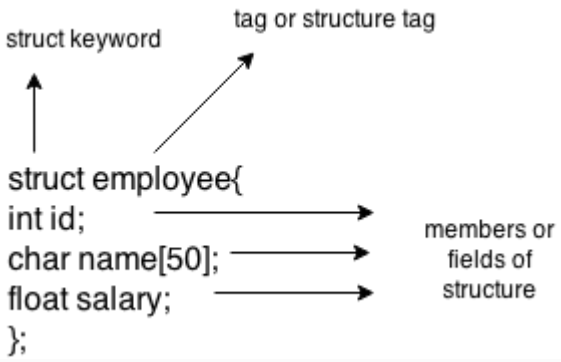
```
struct tag_name
{
data type var_name1;
data type var_name2;
data type var_name3;
};
```

UNIT - IV

Example:

```
struct employee
{
 int id;
char name[50];
float salary;
};
```

Here, **struct** is the keyword, **employee** is the tag name of structure; id, name and salary are the members or fields of the structure.



Declaring structure using normal variable:

We can declare variable for the structure, so that we can access the member of structure easily. There are two ways to declare structure variable:

1. By struct keyword within main() function
2. By declaring variable at the time of defining structure.

1st way:

Syntax: **struct structname varname;**

ex: struct employee e1,e2;

or

2nd way :

```
struct struct_name {
  DataType member1_name;
  DataType member2_name;
```

UNIT - IV

```
  DataType member3_name;
  ...
} var_name;
```

Ex:

```
 struct employee
{
int id;
char name[50];
 float salary;
}e1,e2;
```

Initializing structure using normal variable:

There are two ways to do this.
1) Using Dot(.) operator

```
var_name . memebername = value;

ex: employee.id = 100;
      emply0yee.name="Mani";
     employee.salary=9900;
```

2) All members assigned in one statement

```
struct struct_name var_name ={value for memeber1, value for memeber2
...so on for all the members}

ex: struct employee e1 = {100, "Mani", 9900};
```

Accessing structure members using normal variable:

Structure members are accessed by " . " (member or dot operator)

Syntax:

```
var_name.member1_name;
var_name.member2_name;
```

UNIT - IV

ex:
e1.id;
e1.name;
e1.salary;

Program :

```
#include<stdio.h>
#include <string.h>
struct employee
{   int id;
    char name[50];
}e1;  //declaring e1 variable for structure
int main( )
{
   //store first employee information
   e1.id=101;
   e1.name=”sonu”;
  e1.salary=9900;
   //printing first employee information
   printf( "employee 1 id : %d\n", e1.id);
   printf( "employee 1 name : %s\n", e1.name);
 printf( "employee 1 salary: %s\n", e1.salary);
return 0;
}
```

USES OF STRUCTURES IN C:

1. C Structures can be used to store huge data. Structures act as a database.
2. C Structures can be used to send data to the printer.
3. C Structures can interact with keyboard and mouse to store the data.
4. C Structures can be used in drawing and floppy formatting.
5. C Structures can be used to clear output screen contents.
6. C Structures can be used to check computer’s memory size etc.

Q5) What is an array? Explain about arrays of structures?

A)Structure is used to store the information of One particular object but if we need to store such 100 objects then Array of Structure is used.

UNIT - IV

Example :

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

Explanation :

1. Here Book structure is used to Store the information of one Book.
2. In case if we need to store the Information of 100 books then Array of Structure is used.
3. b1[0] stores the Information of 1st Book , b1[1] stores the information of 2nd Book and So on We can store the information of 100 books.

book[2] is shown Below

	Name	Pages	Price
Book[0]			
Book[1]			

Accessing Pages field of Second Book :

```
Book[1].pages
```

Program

```
#include<stdio.h>
#include <string.h>
struct student
{
        int rollno;
        char name[10];
};
```

UNIT - IV

```
int main(){
int i;
struct student st[5];
        printf("Enter Records of 5 students");
for(i=0;i<5;i++)
{
        printf("\nEnter Rollno:");
        scanf("%d",&st[i].rollno);
        printf("\nEnter Name:");
        scanf("%s",&st[i].name);
}
printf("\nStudent Information List:");
for(i=0;i<5;i++)
{
        printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
}
return 0;
}
```

Q 6 Explain Structure Within structure

1. C define a variable of structure type as a member of other structure type.
2. The syntax to define the structure within structure is

```
struct <struct_name>{
  <data_type> <variable_name>;
  struct <struct_name>
            { <data_type> <variable_name>;
            ……..}<struct_variable>;
  <data_type> <variable_name>;
};
```

UNIT - IV

Example of Structure within Structure

The structure of Employee is declared as

```
 struct employee
 {
        int  emp_id;
        char name[20];
        float salary;
        int dept_no;
        struct date
        {
                int day;
                int month;
                int year;
        }doj;
}e1
```

Accessing Structures within Structures

For Example:-

```
e1.doj.day;
e1.doj.month;
e1.doj.year;
```

Pointers and Structures

The pointer variable to structure variable is declared as

```
struct employee *emp;
```

Access the Pointer in Structures

Access using arrow operator(->) instead of period operator(.).

For Example:

```
emp->name;  emp->emp_id;
```

Q7) What is a union in C language? explain creation, declaration,initialization of union with example.

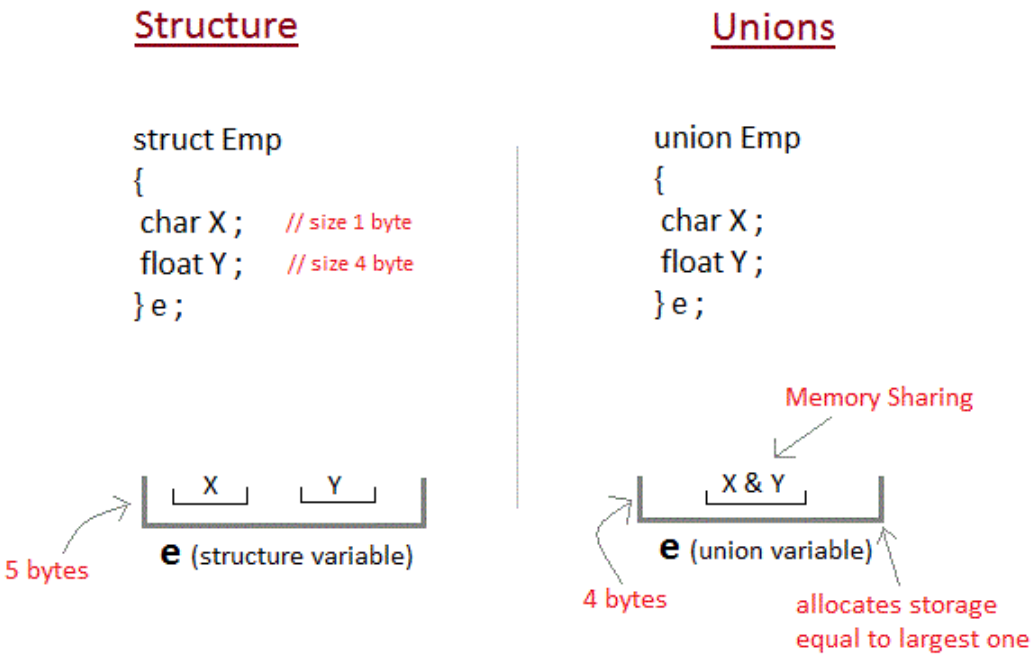
A) C Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called member.

- Union and structure in C are same in concepts, except allocating memory for their members.
- Structure allocates storage space for all its members separately.

UNIT - IV

- Whereas, Union allocates one common storage space for all its members
- We can access only one member of union at a time. We can't access all member values at the same time in union. But, structure can access all member values at the same time. This is because, Union allocates one common storage space for all its members. Where as Structure allocates storage space for all its members separately.
- Many union variables can be created in a program and memory will be allocated for each union variable separately.

Unions are conceptually similar to **structures**. The syntax to declare/define a union is also similar to that of a structure. The only differences is in terms of storage. In **structure** each member has its own storage location, whereas all members of **union** uses a single shared memory location which is equal to the size of its largest data member.



This implies that although a **union** may contain many members of different types, **it cannot handle all the members at the same time**. A **union** is declared using the union keyword.

UNION

Creation of Union variable

UNIT - IV

Syntax:
union tag_name
{
data type var_name1;
data type var_name2;
data type var_name3;
};

Example:
union student
{
int mark;
char name[10];
float average;
};

Declaring union variable:
Syntax: unionunionname variablename;
ex: union student report;

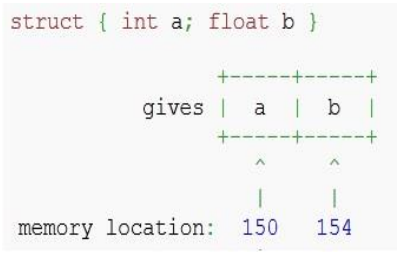
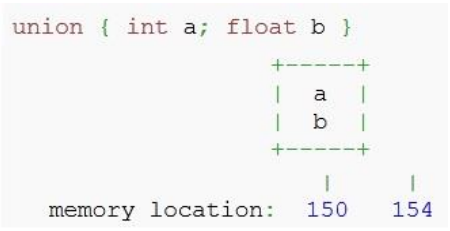
Initializing union variable:
Syntax: union unionname unionvariable ={value1,value2,value3……};
ex: union student report = {100, "Mani", 99.5};

Accessing union variable:
report.mark;
report.name;
report.average;

Q8)Write the difference between structure and union .

	Structure	Union
Memory Allocation	Members of structure do not share memory. So A structure need separate memory space for all its members i.e. all the members have unique storage.	A union shares the memory space among its members so no need to allocate memory to all the members. Shared memory space is allocated i.e. equivalent to the size of member having largest memory.

UNIT - IV

Member Access	Members of structure can be accessed individually at any time.	At a time, only one member of union can be accessed.
Keyword	To define Structure, '**struct**' keyword is used.	To define Union, '**union**' keyword is used.
Initialization	**All** members of structure can be initialized.	**Only** the **first** member of Union can be initialized.
Size	Size of the structure is ≥ to the sum of the each member's size.	Size of union is equivalent to the size of the member having **largest** size.
Syntax	struct struct_name { structure ele 1; structure ele 2; ————- ————- structure ele n; }struct_variable_name;	union union_name { union ele 1; union ele 2; ————- ————- union ele n; }union_variable_name;
Ex:	Example structure declaration: struct s_tag { int ival; float fval; char *cptr; }s;	Example union declaration: union u_tag { int ival; float fval; char *cptr; }u;
Change in Value	Change in the value of one member can not affect the other in structure.	Change in the value of one member can affect the value of other member.
Storage in Memory		

UNIT - IV

Q9) What is enumerated data type? Explain with example.

A)Enumerated data type

Enumerated data type is a user defined data type. It grabs integer vales from a list. Integer values are actually replaced by alphabetic or descriptive names. It increases ease of read of the program. These detailed names are known as enumerators or enumerator constants. Together they form a list which is apparently called enumerator list. Format of enumerated data type is as shown below:

```
enum tag_of_name
{
  member_1
  member_2
  ...
};
```

Figure Format of enumerated data type

Explanation: Here,

enum is the keyword

tag_of_name is the identifier which represents enumerated data type

enum tag_of_name represents derived data type when combined together

member_1, member_2,... represents constants of type integer. These integer constants are represented by their detailed names. These are known as enumerators.

Member_1, member_2 are not variables. They are constants. Compiler itself assigns value 0 to first member or member_1, value 1 to member_2 and subsequent members will be assigned the values in increasing order sequentially.

Let us consider an example of enumerated data type so that it can be better understood:

```
enum employee_depart
 {
  it, operations, management, accounts, stores
```

UNIT - IV

};

Explanation: Here,

enum is the keyword

employee_depart is the name of tag or name of enumerated data type

it, operations, management, accounts, stores are enumerators

Compiler will assign vale 1 to it, operations will be assigned value 2 and so on.

Since we know the syntax of defining enumerated data type next step is to declare variables for this data type.

Syntax : **enum tag_of_name variables;**

Let us consider the above example of enumerators and using that the variables will be declared as :

enum employee_depart emp, em;

Explanation: Here,

enum employee_depart is the derived data type

emp, em are the variables of enumerated data type

We can explicitly assign values to these enumerators also. Let us consider an example:

```
enum months
{
 january, february, march, april, may, june, july, august, september, october, november, december
};
enum months month1, month2
```

Explanation: Here,

enum months is the derived data type

january, february, march,..., december will assigned values from 0, 1, 2, ..., 11 respectively.

month1, month2 are the variables of enumerated data type

We can assign values to these members. Let us consider the example

```
enum months
```

UNIT - IV

```
{
  january=4, february, march, april=0, may, june, july
}month1;
```

Explanation: Here,

january=4 will assign value 4 to january then february will be assigned value 5 followed by march with value 6. Now april is assigned value 0 then may will be assigned 1 followed by june with value 2 and then july to value 3 respectively.

```
#include< stdio.h>
void main()
{
int i;
enum month {JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,DEC};
clrscr();
for(i=JAN;i<=DEC;i++)
    printf("\n%d",i);
}
```

Output :

```
01234567891011
```

UNIT - V

UNIT V

OBJECT ORIENTED CONCEPTS USING C++

UNIT - V

SYLLABUS

Object oriented Programming:

- **Introduction to Object Oriented Programming**
- **Structure of C++**
- **Simple Program of C++**
- **Storage Classes**
- **Similarities and Differences between C & C++**
- **Data Member and Data Functions**
- **Object Oriented Concepts**
    - **Class**
    - **Object**
    - **Inheritance**
    - **Polymorphism**
    - **Encapsulation**
    - **Abstraction**

UNIT - V

Introduction to C++:

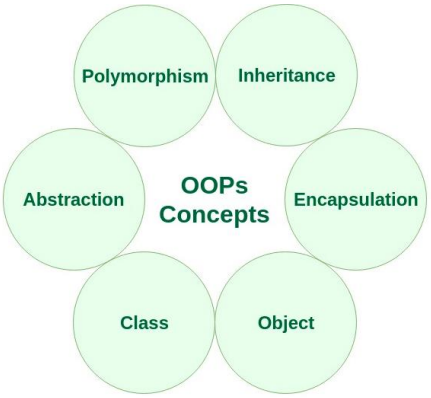
C++, as we all know is an extension to C language and was developed by **Bjarne stroustrup** at bell labs. C++ is an intermediate level language(middle level language), as it comprises a confirmation of both high level and low level language features. C++ is a statically typed, free form, multiparadigm, compiled general-purpose language.

C++ is an **Object Oriented Programming language** but is not purely Object Oriented. Its features like Friend and Virtual, violate some of the very important OOPS features, rendering this language unworthy of being called completely Object Oriented. Its a middle level language.

1Q) Briefly explain Concepts of Object-Oriented programming (oops).

A) Object-oriented programming – As the name suggests uses **objects** in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

. **Characteristics of an Object Oriented Programming language**



(1)Data Abstraction:

Data abstraction refers to the process of providing only essential information to the outside word and hiding their background details. Since classes use the concept of data abstraction, they are known as **Abstract Data Types (ADT).**

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop

UNIT - V

the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

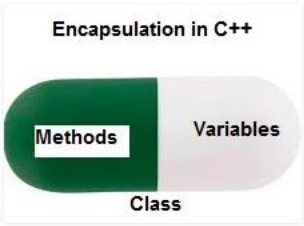
Now if we talk in terms of C++ Programming, C++ classes provides great level of data abstraction. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data ie. State without actually knowing how class has been implemented internally.

(2) Data Encapsulation:

The process of binding data and its associated function into a single unit is known as data encapsulation.

Thus the data can be accessed only by the function that is bind with it. It can not be accessed outside the world.

Thus Data Encapsulation provides data hiding facility.



Encapsulation in C++

Example(just for understanding)

(Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name “sales section”.)

(3) Inheritance:

The process of deriving a new class called the derived class from already existing class is known as inheritance.

It provides the concept of reusability. I.e. we can add new features to the existing class without modifying it.

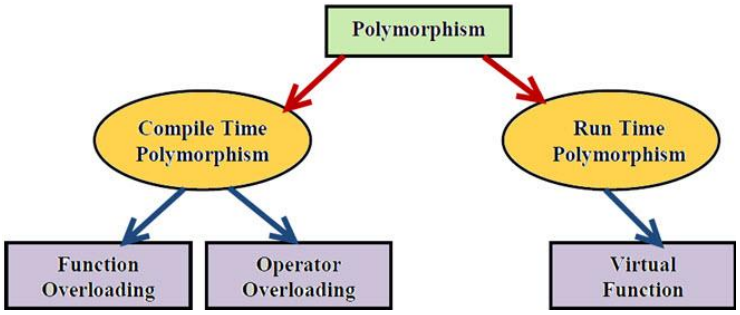
UNIT - V

There are various types of inheritance:

(1) Single
(2) Multiple
(3) Multilevel
(4) Hybrid
(5) Hierarchical

(4) Polymorphism:

The process of using same thing for different purpose is known as polymorphism.
It simply means one name multiple forms.
Two types of polymorphism:
(A)Compile time Polymorphism: Function overloading and operator overloading is example of compile time polymorphism
(B)Run time Polymorphism: Virtual function is an example of runtime polymorphism.



(5) Class: he classes are the most important feature of C++ that leads to Object Oriented programming. Class is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating instance of that class.

The variables inside class definition are called as data members and the functions are called member functions.

For example : Class of birds, all birds can fly and they all have wings and beaks. So here flying is a behavior and wings and beaks are part of their characteristics. And there are many different birds in this class with different names but they all posses this behavior and characteristics.

Similarly, class is just a blue print, which declares and defines characteristics and behavior, namely data members and member functions respectively. And all objects of this class will share these characteristics and behavior.

UNIT - V

1. Class name must start with an uppercase letter(Although this is not mandatory). If class name is made of more than one word, then first letter of each word must be in uppercase. *Example*, class Study, class StudyTonight etc
2. Classes contain, data members and member functions, and the access of these data members and variable depends on the access specifiers (discussed in next section).
3. Class's member functions can be defined inside the class definition or outside the class definition.
4. Class in C++ are similar to structures in C, the only difference being, class defaults to private access control, where as structure defaults to public.
5. All the features of OOPS, revolve around classes in C++. Inheritance, Encapsulation, Abstraction etc.
6. Objects of class holds separate copies of data members. We can create as many objects of a class as we need.
7. Classes do posses more characteristics, like we can create abstract classes, immutable classes, all this we will study later.

Objects

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated..

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects. Once a class has been declared ,we can create variables of that type using the class name.

Each object has different data variables. Objects are initialised using special class functions called **Constructors**. And whenever the object is out of its scope, another special class member function called **Destructor**is called, to release the memory reserved by the object. C++ doesn't have Automatic Garbage Collector like in JAVA, in C++ Destructor performs this task.

2 Q) Write some applications and features of C++

A) Benefits of C++ over C Language

The major difference being OOPS concept, C++ is an object oriented language whereas C language is a procedural language. Apart form this there are many other features of C++ which gives this language an upper hand on C laguage.

UNIT - V

Following features of C++ makes it a stronger language than C,

1. There is Stronger Type Checking in C++.
2. All the OOPS features in C++ like Abstraction, Encapsulation, Inheritance etc makes it more worthy and useful for programmers.
3. C++ supports and allows user defined operators (i.e Operator Overloading) and function overloading is also supported in it.
4. Exception Handling is there in C++.
5. The Concept of Virtual functions and also Constructors and Destructors for Objects.
6. Inline Functions in C++ instead of Macros in C language. Inline functions make complete function body act like Macro, safely.
7. Variables can be declared anywhere in the program in C++, but must be declared before they are used.
8. Software complexity can be easily managed.

Applications of C++ /Uses of C++:

1. Games:

C++ overrides the complexities of 3D games, optimizes resource management and facilitates multiplayer with networking. The language is extremely fast, allows procedural programming for CPU intensive functions and provides greater control over hardware, because of which it has been widely used in development of gaming engines. For instance, the science fiction game Doom 3 is cited as an example of a game that used C++ well and the Unreal Engine, a suite of game development tools, is written in C++.

2. Graphic User Interface (GUI) based applications:

Many highly used applications, such as Image Ready, Adobe Premier, Photoshop and Illustrator, are scripted in C++.

3. Web Browsers:

With the introduction of specialized languages such as PHP and Java, the adoption of C++ is limited for scripting of websites and web applications. However, where speed and reliability are required, C++ is still preferred. For instance, a part of Google's back-end is coded in C++, and the rendering engine of a few open source projects, such as web browser Mozilla Firefox and email client Mozilla Thunderbird, are also scripted in the programming language.

4. Advance Computations and Graphics:

C++ provides the means for building applications requiring real-time physical simulations, high-performance image processing, and mobile sensor applications. Maya 3D software, used for integrated 3D modeling, visual effects and animation, is coded in C++.

UNIT - V

5. Database Software:

C++ and C have been used for scripting MySQL, one of the most popular database management software. The software forms the backbone of a variety of database-based enterprises, such as Google, Wikipedia, Yahoo and YouTube etc.

6. Operating Systems:

C++ forms an integral part of many of the prevalent operating systems including Apple's OS X and various versions of Microsoft Windows, and the erstwhile Symbian mobile OS.

7. Enterprise Software:

C++ finds a purpose in banking and trading enterprise applications, such as those deployed by Bloomberg and Reuters. It is also used in development of advanced software, such as flight simulators and radar processing.

8. Medical and Engineering Applications:

Many advanced medical equipments, such as MRI machines, use C++ language for scripting their software. It is also part of engineering applications, such as high-end CAD/CAM systems.

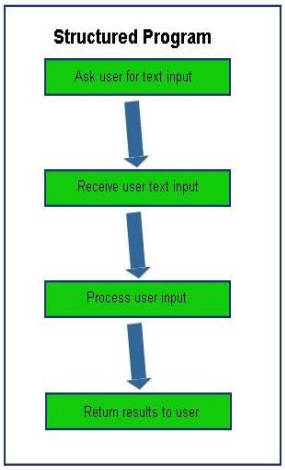
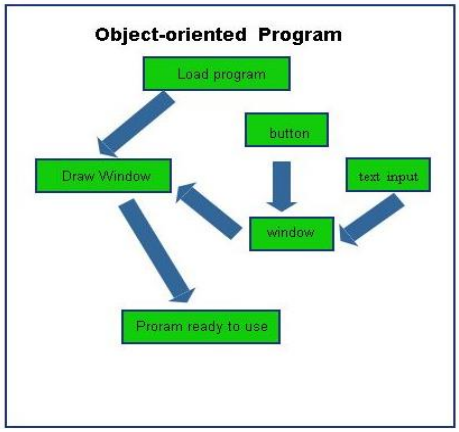
9. Compilers:

A host of compilers including Apple C++, Bloodshed Dev-C++, Clang C++ and MINGW make use of C++ language.

3 Q) Write the differences between Structured vs. Object Oriented Programming.

Structured Programming	Object Oriented Programming
Structured Programming is designed which focuses on **process**/ logical structure and then data required for that process.	Object Oriented Programming is designed which focuses on **data**.
Structured programming follows **top-down approach**.	Object oriented programming follows **bottom-up approach**.
Structured Programming is also known as **Modular Programming** and a	Object Oriented Programming supports **inheritance, encapsulation, abstraction**, **polymorphism**, etc.

UNIT - V

subset of **procedural programming language**.	
In Structured Programming, Programs are divided into small self contained **functions**.	In Object Oriented Programming, Programs are divided into small entities called **objects**.
Structured Programming is **less** secure as there is no way of **data hiding**	Object Oriented Programming is more secure as having data hiding feature.
Structured Programming can solve **moderately** complex programs.	Object Oriented Programming can solve any **complex**programs.
Structured Programming provides **less reusability**, more function dependency	Object Oriented Programming provides more reusability, less function **dependency**.
Less abstraction and less flexibility.	More abstraction and more **flexibility**.
Less data security	More data security
Flexibility is less	Flexibility is more
	

UNIT - V

4Q) Explain history of C++.

A) Background History :

- C++ Development started in 1979.
- During the creation of Ph.D. thesis, Bjarne Stroustrup worked with language called Simula.
- Simula is basically useful for the simulation work.
- Simula was first language to support object-oriented programming paradigm
- Bjarne Stroustrup identified that this OOP features can be included in the software development.
- After that Bjarne Stroustrup started working on the C language and added more extra OOP features to the classic C.
- He added features in such a fashion that the basic flavour of C remains unaffected.
- C++ includes some add-on features such as classes, basic inheritance, in-lining, default function arguments, and strong type checking

Basic History of C++

- During 1970 Dennis Ritchie created C Programming language.
- In the early 1980's, also at Bell Laboratories, another programming language was created which was based upon the C language.
- C++ is also called as C with classes
- Stroustrup states that the purpose of C++ is to make writing good programs easier and more pleasant for the individual programmer.
- C++ programming language is extension to C Language.
- In C we have already used increment operator (++). Therefore we called C++ as "Incremented C" means Extension to C.

Versions of C++ Language

There are several versions of C++ Programming Language –

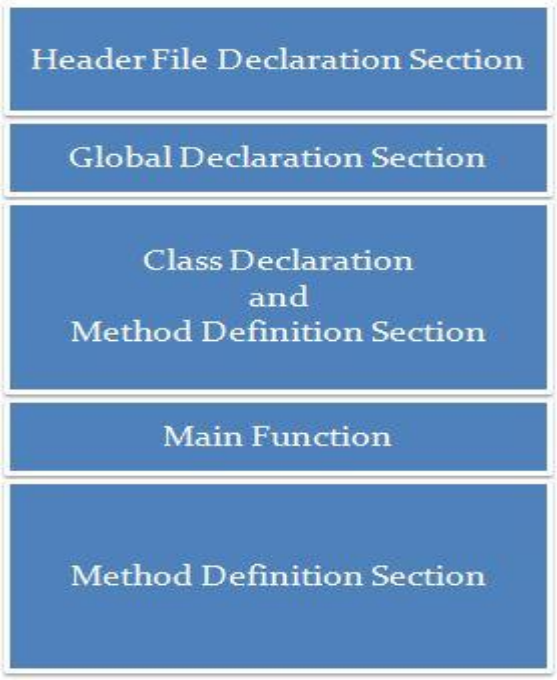
- Visual C++
- Borland C++
- Turbo C++
- Standardize C++ [ANSI C++]

UNIT - V

5Q) Explain the structure of C++ program.

C++ Programming language is most popular language after C Programming language. C++ is first Object oriented programming language.

We have summarize structure of C++ Program in the following Picture –



Section 1 : Header File Declaration Section

- Header files used in the program are listed here.
- Header File provides **Prototype declaration** for different library functions.
- We can also include **user define header file**.
- Basically all preprocessor directives are written in this section.

Section 2 : Global Declaration Section

- Global Variables are declared here.
- Global Declaration may include –
  - Declaring Structure
  - Declaring Class
  - Declaring Variable

UNIT - V

Section 3 : Class Declaration Section

- Actually this section can be considered as sub section for the global declaration section.
- Class declaration and all methods of that class are defined here.

Section 4 : Main Function

- Each and every C++ program always starts with main function.
- This is entry point for all the function. Each and every method is called indirectly through main.
- We can create class objects in the main.
- Operating system call this function automatically.

Section 5 : Method Definition Section

This is optional section . Generally this method was used in C Programming.

First program

```
#include <iostream.h>
using namespace std;
// main() is where program execution begins.
int main()
 {
  cout << "Hello World"; // prints Hello World
  return 0;
}
```

- The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header **<iostream>** is needed.
- The line **using namespace std;** tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++.
- The next line '**// main() is where program execution begins.**' is a single-line comment available in C++. Single-line comments begin with // and stop at the end of the line.
- The line **int main()** is the main function where program execution begins.
- The next line **cout << "This is my first C++ program.";** causes the message "This is my first C++ program" to be displayed on the screen.
- The next line **return 0;** terminates main( )function and causes it to return the value 0 to the calling process.

UNIT - V

6Q) What is the difference between C and C++?

A) The major difference between C and C++ is that C is a procedural programming language and does not support classes and objects, while C++ is a combination of both procedural and object oriented programming language; therefore C++ can be called a hybrid language. The following table presents differences between C and C++ in detail.

Difference between C and C++	
C	**C++**
C was developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs.	C++ was developed by Bjarne Stroustrup in 1979 with C++'s predecessor "C with Classes".
When compared to C++, C is a subset of C++.	C++ is a superset of C. C++ can run most of C code while C cannot run C++ code.
C supports procedural programming paradigm for code development.	C++ supports both procedural and object oriented programming paradigms; therefore C++ is also called a hybrid language.
C does not support object oriented programming; therefore it has no support for polymorphism, encapsulation, and inheritance.	Being an object oriented programming language C++ supports polymorphism, encapsulation, and inheritance.
In C (because it is a procedural programming language), data and functions are separate and free entities.	In C++ (when it is used as object oriented programming language), data and functions are encapsulated together in form of an object. For creating objects class provides a blueprint of structure of the object.
In C, data are free entities and can be manipulated by outside code. This is because C does not support information hiding.	In C++, Encapsulation hides the data to ensure that data structures and operators are used as intended.
C, being a procedural programming, it is a function driven language.	While, C++, being an object oriented programming, it is an object driven language.

UNIT - V

C does not support function and operator overloading.	C++ supports both function and operator overloading.
C does not allow functions to be defined inside structures.	In C++, functions can be used inside a structure.
C does not have namespace feature.	C++ uses NAMESPACE which avoid name collisions. A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries. All identifiers at namespace scope are visible to one another without qualification. Identifiers outside the namespace can access the members by using the fully qualified name for each identifier.
C uses functions for input/output. For example scanf and printf.	C++ uses objects for input output. For example cin and cout.
C does not support reference variables.	C++ supports reference variables.
C has no support for virtual and friend functions.	C++ supports virtual and friend functions.
C provides malloc() and calloc()functions for dynamic memory allocation, and free() for memory de-allocation.	C++ provides new operator for memory allocation and delete operator for memory de-allocation.
C does not provide direct support for error handling (also called exception handling)	C++ provides support for exception handling. Exceptions are used for "hard" errors that make the code incorrect.

UNIT - V

7Q)Explain Data Members and Member Functions in C++ programming

Data Members:

- The variables declared inside the class by using any fundamental data types (like int, char, float etc) or derived data type (like class, structure, pointer etc are known as data members.

Member functions:

- The functions declared inside the class are known as member functions.
- Member functions are methods or functions that are defined inside of objects.
- Generally used to manipulate data members and other object data.

There are three types of data members/member functions in C++:

1. Private members
2. Public members
3. Protected members

1) Private members

The members which are declared in private section of the class (using private access modifier) are known as private members. Private members can also be accessible within the same class in which they are declared.

2) Public members

The members which are declared in public section of the class (using public access modifier) are known as public members. Public members can access within the class and outside of the class by using the object name of the class in which they are declared.

3)Protected:

Protected access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any subclass (derived class) of that class.

Consider the example:

UNIT - V

```
class Test
{
        private:
                int a;
                float b;
                char *name;

                void getA() { a=10; }
                ...;

        public:
                int count;
                void getB() { b=20; }

                ...;
};
```

Here, a, b, and name are the private data members and count is a public data member. While, getA() is a private member function and getB() is public member functions.

private members are directly accessible within the member functions and member functions are accessible within in main() function (outside of the class) by using period (dot) operator like object_name.member_name;

Accessing Protected Data Members

- Protected data members, can be accessed directly using dot (.) operator inside the subclass of the current class
- Protected data members can be accessed in the same way as public data members from friend functions or classes and from derived classes.

8 Q) Explain Storage Classes in C++ with Examples

Storage Classes are used to describe the features of a variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program. To specify the storage class for a variable, the following syntax is to be followed:

Syntax:

```
storage_class var_data_type var_name;
```

UNIT - V

C++ uses 5 storage classes, namely:

1. auto
2. register
3. extern
4. static
5. mutable

C++ Storage Class

Storage Class	Keyword	Lifetime	Visibility	Initial Value
Automatic	auto	Function Block	Local	Garbage
External	extem	Whole Program	Global	Zero
Static	static	Whole Program	Local	Zero
Register	register	Function Block	Local	Garbage
Mutable	mutable	Class	Local	Garbage

Below is the detailed explanation of each storage class:

1. **auto:** The auto keyword provides type inference capabilities, using which automatic deduction of the data type of an expression in a programming language can be done. This consumes less time having to write out things the compiler already knows. As all the types are deduced in compiler phase only, the time for compilation increases slightly but it does not affect the run time of the program.
   Example:
   ```
   auto int x;
   ```

2. **extern:** an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere. It can be accessed within any function/block. Also, a normal global variable can be made extern as well by placing the 'extern' keyword before its declaration/definition in any function/block.
   The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program.
   Example:
   ```
   extern int x;
   ```

UNIT - V

3. **static**: This storage class is used to declare static variables which are popularly used while writing programs in C++ language. Static variables have a property of preserving their value even after they are out of their scope! Hence, static variables preserve the value of their last use in their scope. So we can say that they are initialized only once and exist until the termination of the program. Thus, no new memory is allocated because they are not re-declared. Their scope is local to the function to which they were defined. Global static variables can be accessed anywhere in the program. By default, they are assigned the value 0 by the compiler.

   Example:

   ```
   static int count = 0;
   ```

4. **register:** This storage class declares register variables which have the same functionality as that of the auto variables. The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free register is available. This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program. If a free register is not available, these are then stored in the memory only. An important and interesting point to be noted here is that we cannot obtain the address of a register variable using pointers.

   Example:

   ```
   register char b = 'G';
   ```

5. mutable: Sometimes there is a requirement to modify one or more data members of class/struct through const function even though you don't want the function to update other members of class/struct. This task can be easily performed by using the mutable keyword. The keyword mutable is mainly used to allow a particular data member of const object to be modified. When we declare a function as const, this pointer passed to function becomes const. Adding mutable to a variable allows a const pointer to change members.

   Example:

   ```
   mutable int y;
   ```