D.N.R. COLLEGE (A)

BHIMAVARAM

PROBLEM SOLVING USING C

STUDY MATERIAL

DEPARTMENT OF COMPUTER SCIENCE

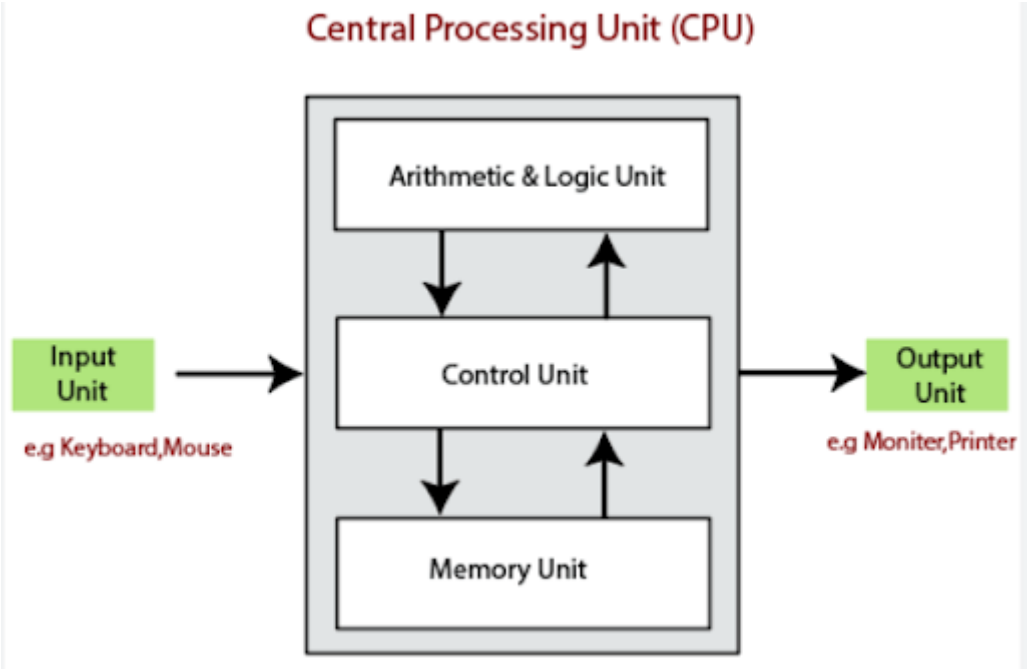
UNIT-I

INTRODUCTION TO COMPUTER AND PROGRAMMING

Computer: Computer is an electronic device which can process data transformed into information. That is useful to people.

Block Diagram of a Computer: The computer has mainly three parts.

1. Input Unit
2. C.P.U(Central Processing Unit)
3. Output Unit



1. Input Unit : The standard input device is keyboard some of the input devices are mouse, scanner, touch screen, OMR(Optical Mark Recognition), OBR(Optical Barcode Recognition).

2. Central Processing Unit : It has three types

I. Memory Unit
II. Arithmetic and Logical Unit(A. L. U)
III. Control Unit

Memory Unit: It is known has internal memory or primary memory. It has two types

RAM (Random Access Memory): It stores programs and instructions and data temporarily. Until the computer switch is turned off. Ram is volatile memory.

ROM (Read Only Memory): Rom is an non-volatile memory .The data that is stored in Rom is permanent.

Arithmetic and Logical Unit (A. L. U): It performs the actual arithmetic and logical unit processing. The result can be stored in the memory or it can be retained in arithmetic logical unit for further calculations.

Control Unit : It controls all other units in the system its main functions are

- To control transfer of information between various units in the system.
- To initiate appropriate functions by arithmetic logical unit.

3. Output Unit : It communicates the results of processing to the users in various forms. The standard output unit is monitor, V. D. U (Visual Display Unit).

CONCEPT OF HARDWARE AND SOFTWARE

Hardware – Any physical device or equipment used in or with a computer system (anything you can see and touch).

External hardware

- External hardware devices (peripherals) – any hardware device that is located outside the computer.
- Input device – a piece of hardware device which is used to enter information to a computer for processing.

Examples: keyboard, mouse, track pad (or touchpad), touch screen, joystick, microphone, light pen, webcam, speech input, etc.

- Output device – a piece of hardware device that receives information from a computer.

Examples: monitor, printer, scanner, speaker, display screen (tablet, smart phone …), projector, head phone, etc.



Internal hardware

- Internal hardware devices (or internal hardware components) – any piece of hardware device that is located inside the computer.
- Examples: CPU, hard disk drive, ROM, RAM, etc.

Software – a set of instructions or programs that tells a computer what to do or how to perform a specific task (computer software runs on hardware).

Types of Software's :

1. System Software
2. Application Software
3. Commercial Software
4. Open Source Software
5. Public Domain Software
6. Freeware Software

1. System Software: This is a collection of programs to control and operate the operations of computer hardware. This software's are written in low level language. System software interface between user and system hardware.

E g:-Operating System (OS),Compilers.

Features:-

1. Close to system
2. Difficult to design and to understand
3. Fast in speed.
4. Written in low level language.
5. Less interactive.

2. Application Software: Application software protects are designed to satisfy a particular need of particular environment. Application software is to perform various applications on the computer app.

Features:-

1. Close to user
2. Slow in speed
3. Easy to understand
4. Written in high level language
5. Easy to design
6. More interactive

3. Commercial Software: Commercial software is any software or program that is designed and developed for licensing or sale to end users.

- Commercial software is easy to use and easier information into existing system. These are mainly used in business because update services are available.

E.g.:-ATM, Customer services.

Features:-

1. Easy to use
2. Easy to implement.

4. Open Source Software: Open source software is computer software. The source code is available with a license. The copy right holder provides the rights to study change and distribute the software to any one for any purpose

E.g.:-Firefox, Open office.

5. Domain Software: Public domain software that has been placed in public domain. There is ownership such as copy right patent.

E.g.:-Flash, Video game player.

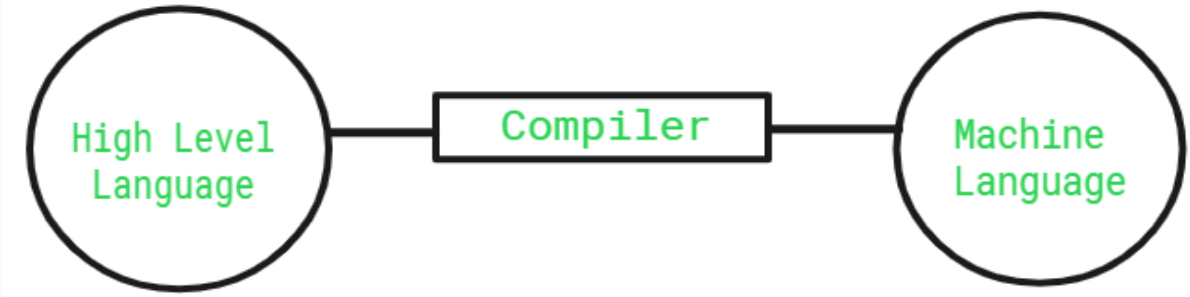
6. Freeware Software: Freeware software is domain downloadable and free of charge. It is a free for personal use. It is freeware does not contain any license.

E.g.:-Adobe reader, Skype.

COMPILER AND INTERPRETER

Compiler: A compiler translates code from a high-level programming language (like Python, JavaScript or Go) into machine code before the program runs.

- The Compiler is a translator which takes input i.e., High-Level Language, and produces an output of low-level language i.e. machine or assembly language. The work of a Compiler is to transform the codes written in the programming language into machine code (format of 0s and 1s) so that computers can understand.
- A compiler is more intelligent than an assembler it checks all kinds of limits, ranges, errors, etc.
- But its program run time is more and occupies a larger part of memory. It has a slow speed because a compiler goes through the entire program and then translates the entire program into machine codes.



Advantages of Compiler

- Compiled code runs faster in comparison to Interpreted code.
- Compilers help in improving the security of Applications.
- As Compilers give Debugging tools, which help in fixing errors easily.

Disadvantages of Compiler

- The compiler can catch only syntax errors and some semantic errors.
- Compilation can take more time in the case of bulky code.

Interpreter: An interpreter translates code written in a high-level programming language into machine code line-by-line as the code runs.

- It translates only one statement of the program at a time.
- Interpreters, more often than not are smaller than compilers.
- The simple role of an interpreter is to translate the material into a target language. An Interpreter works line by line on a code. It also converts high-level language to machine language.

Advantages of Interpreter

- Programs written in an Interpreted language are easier to debug.
- Interpreters allow the management of memory automatically, which reduces memory error risks.
- Interpreted Language is more flexible than a Compiled language.

Disadvantages of Interpreter

- The interpreter can run only the corresponding Interpreted program.
- Interpreted code runs slower in comparison to Compiled code.

Compiler	Interpreter
The compiler displays all issues after compilation.	The interpreter displays issues for specific lines.
The compiler is based on the translation linking-loading paradigm	The Interpreter is based on the Interpretation Method.
compiler requires the entire programme	The interpreter only needs one line of code.
A compiler transforms high-level programming language code into machine code before a program's execution	In contrast, an interpreter transforms each high-level programme statement into machine code separately.
Computed code runs faster	Interpreted code runs slower than computed.

Algorithm

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

(OR)

Algorithm is a step by step process to solve a particular problem.

Properties of Algorithm:-

The properties of algorithm as follows

1. The algorithm have steps are precisely stated (defined).

2. The algorithm must have unambiguous instructions(clear instructions).

3. The algorithm should not have any uncertainty about execution of next instruction.

4. It must be finite and cannot been open ended.

5. The algorithms must be terminating after finite number of steps.

6. The algorithm should be universal leads to a unique solution of the problem.

**** Write An Algorithm for Addition of Two Numbers.**

Step-1: Start

Step-2: Read a, b values.

Step-3: Add a, b and store result in sum(sum=a + b). Step-4: Display sum.

Step-5: Stop

**** Find The Biggest of Three Numbers.**

Step-1: Start

Step-2: Read the three numbers to be compared, as A, B and C.

Step-3: Check if A is greater than B.

Step-4: If true, then check if A is greater than C.

Step-5: If true, print 'A' as the biggest number.

Step-6: If false, print 'C' as the biggest number.

Step-7: If false, then check if B is greater than C.

Step-8: If true, print 'B' as the biggest number.

Step-9: If false, print 'C' as the biggest number.

Step-10: End

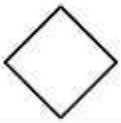
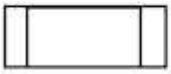
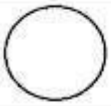
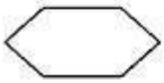
Flow chart

"A flowchart is a graphical representation of algorithm".

The program in which different types of instructions are drawn in the form of different shapes of boxes and the logical flow is indicated by interconnecting arrows. The main objective of creating flowchart is to help the programmer in understanding the logic of programmer and to trap any kind of logical errors present in algorithm.

Flow Chart Symbols:-

Flow chart use different symbols to represent different operations to the program. A flow chart is drawn according to define rules using standard flowchart symbols prescribed by "ANSI"-(American National Standard Institute). The standard symbols that are frequently used in flow chart are

Symbol		Purpose
	Oval	Start or Stop
	Parallelogram	Input or Output
	Rectangle	Process
	Diamond	Decision
	Arrow or Line	Flow Direction
	Double sided Rectangle	Sub program/ function
	Circle	Connector
	Hexagon	Iteration

Different Symbols used in Flowchart

Flow Chart Symbols:-

Flow lines are represented by arrow lines. That are used to connect symbols these lines indicate the sequence of steps and the flow of operations.

Terminator:-Terminator is used to represent by rectangle with rounded ends. These symbol is used to indicate the beginning (start), the termination (end/stop) in the program logic.

Input (or) Output:-Input/output is represented by the parallelogram these symbol is represents an input taken from the user (or) the output that is displayed to user.

Processing:- The processing is represented by rectangle this symbol is used for representing arithmetic and data movement instructions. It denotes the logical process of moving data from one memory location to another location.

Decision:-This symbol is represented by diamond. It denotes a decision to be made. This symbol has one entry and exist paths. The path chosen depends on whether the answer to a question is yes (or) no.

Connector:-This symbol is represented by circle. It is used to join different flow lines.

Advantages of Flow Charts:

The following are the advantages of flow chart

Makes Logic Clear: It is easy follow a graphical representation of the task. The symbols are connected in such a way. That they make the system visible.

Communication: Since the flow chart is a graphical representation of a problem solving logic. It is enhanced meaning of communicating the logic of a system.

Effective Analysis: A flow chart helps the problem analysis in an effective way.

Proper Testing and Debugging: Flow chart helps to identify and correct the errors in the program. It also helps in testing process.

Appropriate Document: Flow chart acts as a high quality program documentation tool.

Disadvantages of Flowchart:

The following are the disadvantages of a flow chart

Complex: For very large programming consisting of thousands of statements. The flow chart consists of many papers (or) pages making them different to flow.

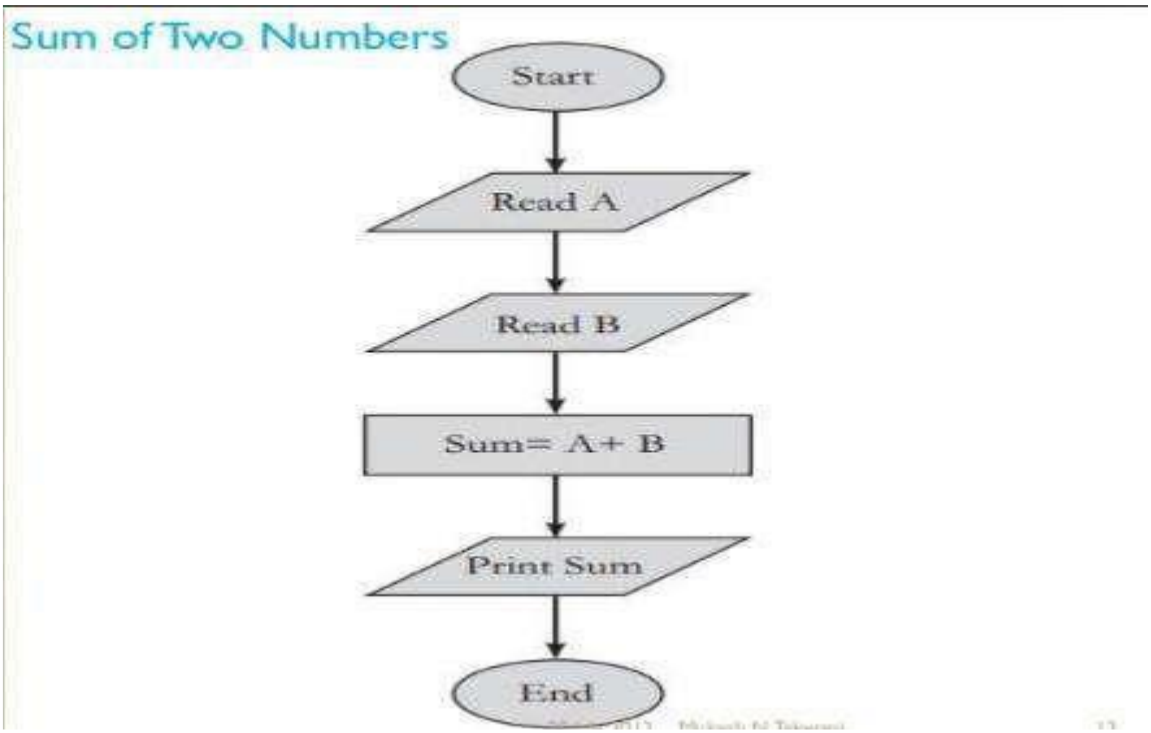
Costly: Flow charts are feasible for sort and straight forward problem solving logic flow chart because a costly flow chart, in huge application**.**

Difficult to Modify: Any modification to flow chart needs to redraw the flowchart. Considering the entire logic again due to its symbolic nature redrawing complex flow chart is a difficult task.

No Update: Programs are generally updated regularly but the corresponding flow charts are not updated regularly.

Examples:-

Draw a flow chart to find sum of two numbers



Program: Program is a set of instructions to solve a particular problem.

Programming:-

- Process of creating programs.
- A programming language is a computer language used to write instructions for computer in the well- defined format.
- The set of instructions is called is called a program and the process of creating programs is called programming.

Types of Programming Language: Programming languages are classified into major languages.

1. Machine Language (Or) Binary Language(First Generation Language)

2. Assembly Language(Second Generation Language)

3. High Level Language(Third Generation Language)

4. Very High Level Language(Fourth Generation Language)

1. Machine Language (or) Binary Language:-

- Machine level language is the lowest level programming understand by computers
- To perform various input and output operations
- The set of instructions which are directly understood by the C.P.U of computer is called machine language.
- The programming language which contains machine code is called machine language.
- All the information in the computer is handling using integrated circuits, semiconductors.
- The machine language uses two binary digits 0 and 1 to handle input and outputs. So it is also known as binary language.
- Machine language different from machine to machine because the internal structure of every computer different from one computer to another.
- Machine language can be easily used by the computer, but it is difficult to read and understand by the user.
- The machine language program runs fastly because no translations are require for the C.P.U.

2. Assembly Language(Second Generation Language):-

- Assembly language was the next high level programming language which is categorized as the second generation language
- Assembly language is easy to understand compare to machine language. It uses English words it performs specific operations for example “add” is used for addition, “sub” is used for subtraction etc.
- Assembly language is machine depended language
- Assembler is used to translate assembly language instructions in to machine language and reverse also
- Assembly language statements are return in one per line where each statements contains operations

- Programming in assembly language requires extensive knowledge of computer design.

3. High Level Language (or) Third Generation Language:-

- High level programming languages includes: FORTRAN, COBOL, PASCAL, BASIC, C, C++, JAVA which enables the programs to develop the software applications.
- The high level language use syntax which is very easy to understand.
- Programs written in high level language (or) shorter in length.
- Translates like compilers & interpreters are used to translate programs written in high level languages into machine languages and reverse also
- High level languages are machine independent
- High level languages are easy to learn because they use common English words as, they (or) their keyword
- It is easy to modify and maintain the programs certain in high level language.

4. Very High Level Language:-

- Fourth generation languages are the easiest programming languages available today. They are very easy to learn because these syntax and grammar is very easy to learn.
- Programming in this languages does not require any programming experience in other language.
- Fourth generation languages are machine independent.
- Fourth generation languages are also known as very high level languages.
- Most fourth generation languages are used to occur data bases
- Sql(structured query language) is the best example fourth generation language. It is used to create and modify information in dbms (database management system).

History of C:

'C' is a general purpose programming language. It is a procedure – oriented, Structured programming language.

The structured programming language allows you divided into small modules by using functions. It is a middle level language that means it has good high-level language programming skills and it also has low level programming features.

Dennis Ritchie developed C language in the year 1972 at AT&T Bell laboratories in U.S.A. The C-language was developed from the language B.C.P.L (Basic combined

programming language). The C-language is well suited for developing system software and application software

Features of C-Language:

C-Language is very powerful and popular because of its features. The Main features are:

Portability: C-Language programs are highly portable. Portability means a c-program written in one environment can be executed in another environment. For example you write a program in DOS environment you can run in windows environment.

Structured Programming Language: C-language is a structured programming language the structured programming basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as functions.

Extendibility: C-language has an important facility called extendibility. It means you can write your own file or functions and include in another programs in other words a user can write No. of functions, sub- programs according to the requirement.

Reliability: A 'C' compiler gives and accurate results it has a facility of warning which guides for better and efficient programming.

Middle Level Language: 'C' language is also called middle level language. Because it has both types of features i.e. high-level languages as well as low-level languages.

Powerful: C is provides variety of data types, functions, conditional statements and looping statements.

Uses of C Language:

- 'C' is very simple language i.e., used by software professionals
- The uses of 'c' language are:-
- C-language is mainly used for system programming.
- It is widely accepted by professionals
- For portability and convenience is sometimes used as an intermediate language for implementing other languages.
- 'C' language is widely used to implement user applications.
- For creating compiles of different languages. Compiler is converts into source code to machine code.
- Unix kernel is completely developed in C- language.

Keywords:

C-languages have some reserved words which cannot be used as variables the reserved words are called keywords.

There are mainly 40keywords among which **32 keywords** are used by many 'C' compilers these keywords are called standard keywords whereas the remaining keywords are called optional keywords.

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

Optional Keywords :

ado	Fortran	sum	huge
entry	Near	far	pascal

Identifiers: Identifier is a name that is given to variables, functions and arrays.

- Rules for constructing Identifiers:
- Identifiers must be from the character set.
- The first character of an identifier should be an alphabet. It should not be a digit or special character.
- Identifiers should not be a keyword.
- Special characters are not accepted in the identifiers except '-' (under score)
- The length of an identifier should not be exceeding eight characters.

Character Set: Character set means that the characters and symbols that a C-Program can understand and accept these are grouped to form the commands, expressions, words, statements and other tokens for C- language. There are mainly four categories.

1. Letters (or) Alphabet: In the character set, character or alphabet are represented by (A-Z) or (a-z).

E.g : A B C…. (or) a b c……

2. Digits : In the character set digit are represented by (0-9)
E.g : 0 1 2 3 4 5 6 7 8 9

3. Special Characters : There are total 30 special characters used in the C-programming. Special characters are used for C- statements like to create an arithmetic statement.

Characters	Meaning	Characters	Meaning	Characters	Meaning
,	Comma	{	Left brace	+	Plus symbol
.	Period	}	Right brace	-	Hypen sign
:	Colon	<	Left angle	*	Asterisk sign
?	Question mark	>	Right angle	#	Hash symbol
‘	Single quote	=	Equal to sign	%	Percentage sign
“	Double quote	!	Exclamation mask	^	Caret symbol
(	Left parenthesis	|	Pipe symbol	&	Ampersand sign
)	Right parenthesis	/	Forward slash	@	At the rate
[	Left bracket	\	Backward slash	_	Underscore
]	Right bracket	~	Tide symbol	;	Semicolon

4. Empty Space Character : White spaces has blank space ,new line return, horizontal tab spaces etc.,

Variables : Variable is an identifier .It is used for storing value .The variable can be changed during the execution of the program .Variable refers to an address of memory where the data is stored.

For example

1. int a;
2. int a, b, c;

Types of Variables: There are 5 types of variables which are as follows:

1. Local variables
2. Global variables
3. Static variables
4. Automatic variables
5. External variables

1. Local Variables

Variables that are declared inside the functions are called local variable. Local variables must be declared before use. Only local functions can change the value of variables.

Example

```
int main( )
{
int m =10; //local variable
}
```

2. Global Variables

Variables that are declared outside the functions are called global variables. Any functions can change the value of variables.

Example

```
int n = 6; //global variable
int main()
{
int m =10; //local variable
}
```

3. Static Variables

variables that are declared with **the static keyword** are called static variables.

```
int main()
{
int m =10; //local variable
static n = 6; //static variable
}
```

4. Automatic Variables

all the variables that are declared inside the functions are default considered as automatic variables. Automatic variables can be declared using the auto keyword.

```
int main()
{
int m =10; //local variable (Automatic variable)
auto n = 6; //automatic variable
}
```

5. External Variables

External variables are declared using the extern keyword. The variables with the extern keyword can be used in multiple C source files.

extern m =10; //external variable

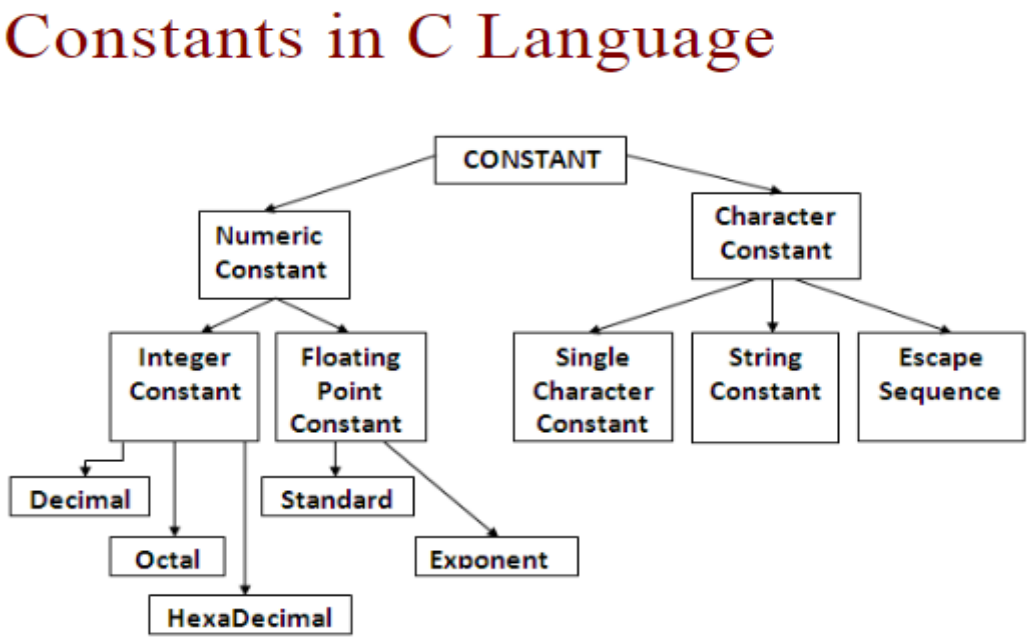
Rules for Naming a Variable in C

We give a variable a meaningful name when we create it. Here are the rules that we must follow when naming it:

1. A variable name must only contain alphabets, digits, and underscore.
2. A variable name must start with an alphabet or an underscore only. It cannot start with a digit.
3. No whitespace is allowed within the variable name.
4. A variable name must not be any reserved word or keyword.
5. The C language treats lowercase and uppercase very differently, as it is case sensitive. Usually, we keep the name of the variable in the lower case.

Constants: The value was not changed during the execution of the program is called constants. Mainly they are two types of constants. They are

1. Numeric constants.
2. Character constants.



1) Numeric Constants: These have numeric data with or without decimal points having positive or negative sign. These are further sub divided into two categories. There are

- Integer Constant
- Real or Float Constant

Integer Constant: Integer constant has integer data without any decimal points are with any positive or negative sign. These are further sub divided into three types. Those are

i.Decimal Integer Constant

ii.Octal-Integer Constant

iii.Hexa - Decimal Integer Constant

i) Decimal Integer Constant: These have no decimal points it is a combination of 0 to 9 digits. These have either positive or negative sign.

E.g.: 8, 7, 6, 5, 4…etc.,

ii) Octal - Integer Constant: These consist of combination of numbers from 0to7 with positive or negative sign. It has leading with 'O' (Upper case or lower case). 'O' means octal

E.g.: O37, O-35.

iii) Hexa- Decimal Integer Constant: These have Hexa decimal data leading with OX or X or H (capital or small). These have combination of 0 to 9 and A to F (capital or small). These letters represents the numbers 10 to15.

E.g.:23A,OXB2.

2) Real or Float Constant: Some constants which have decimal point value with in it is having any positive or negative sign.

Eg: 22.34, 2.4 E 38 that means 2.4 X 1038.

Real constants are further divided into two categories.

I.) With Exponent Part

II.) Without Exponent Part.

i.) Without Exponent Part: Without exponent 'E' and having a decimal part mantissa.

E.g.: 30.6, -30.6

ii.)With Exponent Part: It is also called a scientific representation. Here 'C' has base value of 10 It computes the power.

E.g.: 3.5 X 105 = 3.5 e5.

3) Character Constants: Character constants have either a single character or a group of characters or a character with back slash (\) used for special purpose. These are further divided into three types.

a. Single Character Constant.

b. String Character Constant.

c. Back Slash Character Constant.

a) Single Character Constant: These have a single character with in single quotes. So, These are called single character constant.

E.g.: 'a', 'G', 'A',etc.

b) String Character Constant: A string is a combination of character or group of Characters, a string constant or a string is enclosed with in double. So it is called String constant.

E.g. : "DNRCOLLEGE","JOHN",…etc.,

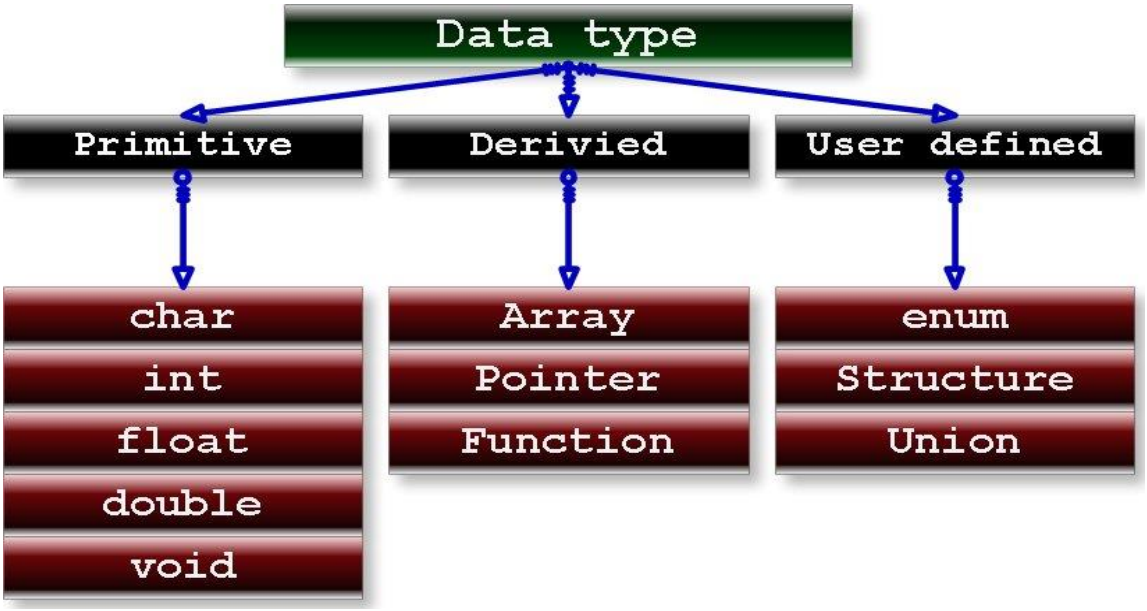
c) Back Slash: These are used for special purpose in 'C' language. These are used in output statement like print (). Puts (), another name of Back slash character constant is escaping sequences.

The Escape Sequences are:

Constant	Meaning
\b	Backspace
\n	Newline
\t	Tab space
\v	Vertical tab
\f	Move one page to next
\o	Null character
\r	returns

Basic Data Types in "C":

A Datatype is a set of values along with a set of rules for allowed operations. 'C' supports several data types of data each of which is stored differently in the computer's memory mainly data types are divided into three types.



1. Primary Data Type: 'C' supports mainly four primary data types.

- Character Data Type
- Integer Data Type
- Float Data Type
- Double Data Type.

Character Data Type:

- The character data type accepts single character only.
- Characters are either signed or unsigned. But mostly characters are used an unsigned type.
- The size of the character data type is 1 byte in the memory.
- The range of unsigned character is 0to 255.
- The range of signed are character is –128 to +127.
- char is the keyword of the character data type.

Syntax: char list of variables;
E.g. char ch1, ch2, ch3;

Integer Data Type:

- An integer type accepts integer values only.
- It does not contains any real or float values.
- The range of an integer variable is -32, 768 to +327,67.
- int is the keyword for integer data type.
- In generally 2 bytes of memory is required to store an integer value.

Syntax: int list of variables;
E. g: int a, b, c;

Float Data Type:

- The float data type accepts real values it can contains any floating point values.
- The range of the floating variable is 3.4E – 38 to 3.4E + 38.
- float is the keyword.
- In generally 4 bytes of memory is required to store an float value with 6 digits of precision.

Syntax: float list of variable;
Eg: float f1, f2, f3;

Double Data Type :

- The double data type accepts large floating value.
- The range of the double variable is 1.7E –308 to 1.7E + 308.
- Double is the key word for double data type.
- In generally 8 bytes of memory is required to store double value.

Syntax: double list of variables;

E.g. double d, e, f; [22]

Void Datatype : void is an empty data type that has no value. The void keyword specifies that the function does not return a value.

2. Derived Data Types: Derived data types are derived from the primary data types. The derived data types may be used for representing a single or multiple values. These are called secondary data type. The derived data types are arrays, pointers, functions, etc.

3. User Defined Data Types: The data types are defined by the user is called user defined data types. The user defined data types are structures, unions etc.

Enumerated Data Types : It allows the user to define a variable or an identifier, which is used for representation of existing data types. In other words, it provides us a way to define our own data type and also can define the value for a variable or an identifier stores into the main memory.

Syntax : enum identifier { v1,v2,v3…….,vn};
Here enum is the reserve word and v1,v2,v3…,vn all are the values which is also called enumeration constants.
Eg : enum month{jan,feb,mar,….,dec};

Typedef : This is used to represent the existing data type .i.e. by using this the new type can be used in place of the old type anywhere in a C program. Also we can create the typedef variables for improving the readability of the program.

Syntax : typedef data-type identifier;

Here data-type may be int, float ,double and char. Identifier gives us the information of new name given to the data type.
Note that typedef cannot create a new type.
Eg: typedef int pay;

Operators: Operators are used for compute a formula or compare two variable values or create logical relationship between two operands or low-level programming we can say operators are used for processing. Operators are divided into:

1. Arithmetic Operators.
2. Relational Operators.
3. Logical Operators.
4. Assignment Operators.
5. Conditional Operators.
6. Bit Wise Operators.
7. Increment / Decrement Operators.
8. Comma Operator.
9. Equality Operators.
10. Size of Operators.

1. Arithmetic Operators: ‘C’ provides all the basic Arithmetic operators in an Arithmetic expression like x + y x and y are the operands. ‘+’ is the operator. ‘C’ used the precedence rules to decide which operator is used first these are listed in the following table.

Operator	Descriptions	Example
+	Addition	A + B
-	Subtraction	A - B
*	Multiplication	A * B
/	Division	A / B
%	Modulus	A % B

2. Relational Operators: The comparison can be done with the help of relational operators. An expression such as containing a relational operator is termed as a relational expression. The value of relational expression is either 1 or 0.It is‘1’ if the specified relation is true. And ‘0’ if the relation is false.

Operator	Description	Example
<	Less than	A < B
<=	Less than or equal to	A< = B
>	Greater than	A > B
>=	Greater than or equal to	A > = B
==	Equal to	A = = B
!=	Not equal to	A ! = B

3. Logical Operators: A statement contains more than one relational operators they must be separated by a logical operator an expression which combines two or more relational expressions is termed as a logical expression compound relational expression.

Operator	Description	Example
&&(Logical AND)	If both conditions are true. The result will be true. Otherwise false	(A>B)&&(A<C)
‖(Logical OR)	If atleast one condition is true. The result will be true. Otherwise false.	(A>B)‖(A<C)
!(Logical NOT)	If condition is true the result will be false. Ifcondition is false the result will be true.	!(A>B)

4. Assignment Operators: Assignment operators are used for assign the result of an expression to a variable the equal to sign (‘=’) is the assignment operator.

Operator	Description	Example
=	Assign the value to Operate to the left	A=B
+=	Adds the operant and assigns The result to left operant.	A+=B
-=	Subtract the right operant From the left operant and stores The result in the left operant.	A-+B

5. Conditional Operators: In 'c' Conditional operator '? :' Constructs conditional expression of the form

Syntax: Exp 1 ? Exp 2 : Exp 3 ;

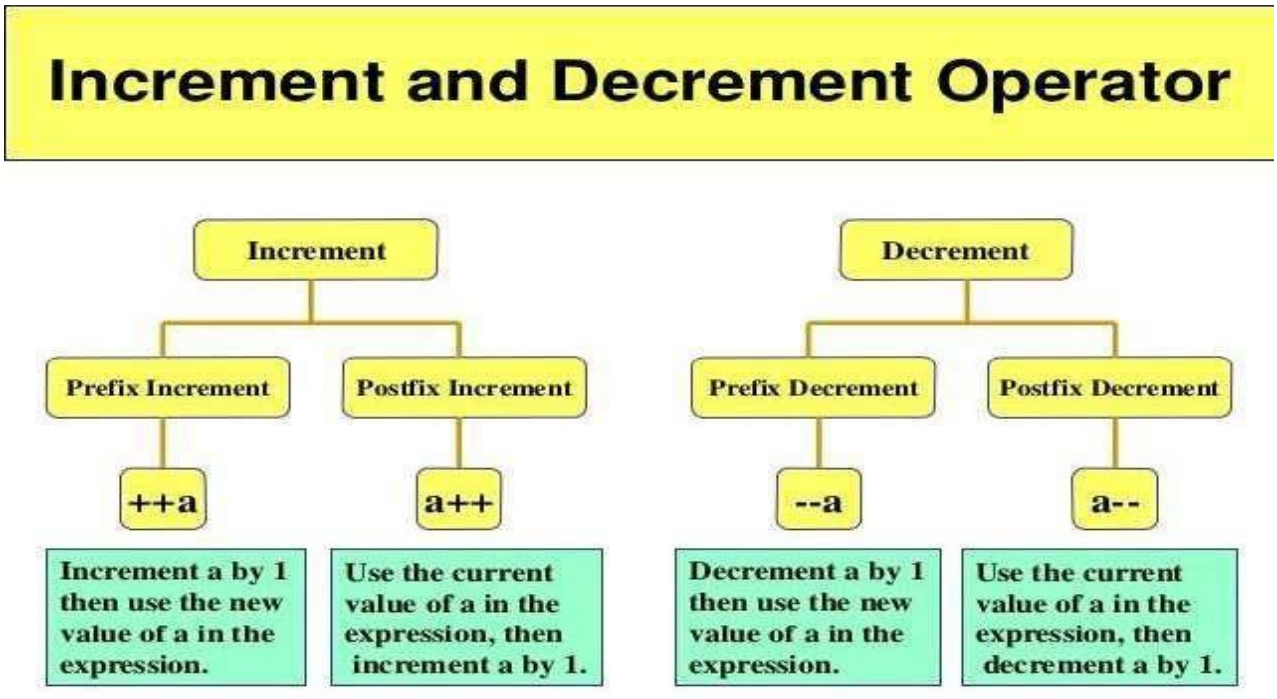
Example : (A>B)?(A+B):(A-B);

Where Exp1, Exp2, Exp3 are expression. The operator '?:' works as follows Exp1 is evaluated first. If it is true then the expression Exp2 is evaluated. If the Exp1, is false Exp3 is evaluated.

6. Bitwise Operators :A special type of operators known as bitwise operators for manipulation of data in Bit level. These operators are used for testing the Bits are shifting them right or left.

Operator	Description	Example
&(and)	Bitwise and	A&B
!(or)	Bitwise or	A|B
^(Exclusive or)	Bitwise exclusive	A^B
~	Bitwise not	~A
<<	Bitwise left shift	A<<1
>>	Bit wise right shift	A>>2

7. Increment / Decrement Operators : In some cases it is necessary to modify the value of the variable by adding one to the variable or –1 to the variable until the loop terminates the 'c' provides the mechanism increment or decrement operators to accomplish this.



Increment Operators : In 'C' the operator '+ +' is used as Increment operator it adds 1 to the variable we can add 1 to the variable in one of the two types.

1. Pre Incrementing.
2. Post Incrementing.

1)Pre-Incrementing : This operator first increments the value of the variable and then execution proceeding.

Syntax:'++'variable name;

E.g.:'++'a;'++'a is equivalent to a=1+a;

2)Post-Incrementing: This operator continuous with the execution before adding 1 to the variable and then increments the variable by 1

Syntax : variable name'++";

Ex : a'++';a'++' is equivalent to a=a+1.

Decrement Operator : In 'C' the operator'--'is used as decrement operator. It subtracts one from the variable we can subtract one from the variable in one of two ways.

1. Pre– Decrementing
2. Post–Decrementing

1**)Pre-Decrementing :** This operator first decreases the value of the variable and then execution proceeds.

Syntax:'--'variable name;

Ex: '--'a;'--'a is equivalent to a= -1+a.

2)Post–Decrementing : This operator continuous with the execution before subtracting one from the variable and then decrements the variable by 1.

Syntax: Variable name'- -';

Ex: a'--';a'--' is equivalent to a=a-1.

8. Comma Operator: The comma operator is used to separate more than one variable invariable declaration. The comma operator is also used to separate two or more expressions.

Ex : int a,b,c;

Ex : int x=2,y=5;

9. Equality Operators: C language supports two kinds of equality operators to compare their operands for equality or inequality. They are '=='equal to x==y, '!=' not equal x!=y.

10. Size of Operators: The size of operator is a unary operator used to calculate the size of data type this operator can be applied to all data types this operator is used to determine the amount of memory space that the variable/expression/data type will take.
Ex: int a =10;

unsigned int result;

result = size of (a);

result = 2

Which is the space required to store the variable 'a 'in memory.

Structure of the 'C' Program.

The 'C' Program structure contains the following sections.

Documentation Section.
Header File Section
Definition Section
Global Declaration Section
main( ) Function **{** **Local Declaration Part** **Execution Part** **}**
User Defined Section

Documentation Section: In the documentation section we can give the comments. Here comment statements are non-executable statements. Comment can be divided into two types there are

1. Single line comments are represented by"//".
2. Multi line comments are represented by"/*…….
……..*/.

Header File Section: This section provides instructions to the compiler to link functions from the Library each header file by default contains with the extension of 'h' the file should be included by using # include.
E.g : #include<stdio.h>

The<stdio.h>is a file it is included all the definitions of input, output functions.

Definition Section: We can define a variable with its value in the definition section.
Syntax: # define variable name value;

E.g.

1. # define a=10;

2. # define name**“ “;**

Global Declaration Section : Some variables are used in more than one function such variables are called global variables and that variables are declared in the global declaration section that is outside of the main() function.

Main( ) Function : Every ‘C’ program must contain main ( ) with empty parenthesis after main is necessary. The function main is the starting point of every ‘C’ program. The program execution starts with the opening braces ({) and ends with the closing braces (}) between these braces the programmer should give the program statements. The main has two parts. They are

Declaration Part : The declaration part declared the entire variables that are used in executable part the initialization of variables are also done in this part.

Execution Part : This part has reading, writing and processing statements having input or output functions, formulas, conditional statements, looping statement and function calling statements.

User Defined Section : In this section function definitions are defined by the user. These functions are generally defined after the main function or before the main function. This section is optional.

E.g.:-Write the First ‘ C ’ Program

```
#include<stdio.h>
void main( )
{
clrscr( );
printf(“My First Program in C”);
 return 0;
getch( );
}
```

Output:-My First Program in C

#include<stdio.h>

- It is the first statement in our code. All pre-process commands starts with # symbol(hash).

- The #include statements tells the compiler to include the standard library (input/output) or header file <stdio.h>in the program.

main()function

- The main ( ) function after all the statements in the program have been written. The last statements in the program have been written. The last statements in the program return will an integral value to the operating system.
- The two{ }curlybracesareusedtogroupofalltherelatedstatementsofmain()function.

printf("My First Program in C");

The printf( ) function is defined in stdio.h file and is used to print text on the screen. The message to be displayed on the screen to the enclosed with in double quotes (" ") and put inside brackets (or) parenthesis. The backslash is an escape sequence and represents a new line character.

Example:-

Sequence	Meaning
\n	New Line
\t	Tab Space
\b	Back Space
\v	Vertical Tab
//	Back Slash

return 0:

This is are turn command that is used to write the value "0" to the operating system give an indication. That there are no errors during the execution of program.

Input/Output Functions

The Input/ Output functions are used for reading the data from the keyboard and displaying the result on the screen are the two main tasks of any program to perform these tasks 'C' has no. of Inputs & Output functions. When a program is needs data it takes data through the input functions and send the result through the output function.

Input Functions : Input functions are

1. scanf( ) function
2. getch( )function
3. getchar( )function
4. getche( )function
5. gets( )function

1)scanf( ): The scanf function is used to read the data from the keyboard. You can store the given value into the variable through the scanf ( );

Syntax : scanf("Controlstring",&v1,&v2,&v3, ----------&vn");

Here v1, v2,---- vn are the variables and "&" is used to store the given value into the variables. The control string has some format strings.

Some Format Strings are:

Format string	Meaning
%c	To print a single character
%d	To print the integer value
%ld	To print the long integer value
%f	To print the floating value.

2)getchar( ): This function is used for reading a single character from the keyboard. The getchar ( )can be assigned a character into the character type variable.

Syntax: ch=getchar();

Where 'ch' is the variable of character type.

3)getch( ) : The getch( ) is used to get a single character from the keyboard it will not display the character on the screen and it will store the given character on the buffer it is used at the end of the program to terminate the output screen.

Syntax: getch();

4)getche( ): The getche( ) is used to get a single character from the keyboard it will display the character on the screen and the character will be stored on buffer it is used at the end of the program to terminate the output screen.

Syntax: getche();

5)gets ( ): The purpose of the gets function is to read the string it can read a string until you press enter key from the keyboard it will mark null character ('\0') at the end of the string.

Syntax: gets(ch);

Where 'ch' is the string variable.

Output Functions: Output functions are

1. printf( )
2. putchar( )
3. puts( )

printf(): The printf( ) is used to display a text message or a value stored in the variable. It requires conversion symbol and the variable name to print the data.

Syntax:printf("controlstrings",v1,v2, ----------vn");

(OR)

printf ("Message line or text line"); Where v1,v2,vn are the variable.

The Control Strings Uses Some printf()

Format Strings Some Format Strings are:

Format string	Meaning
%c	To print a single character
%d	To print the integer value
%ld	To print the long integer value
%f	To print the floating value.

putchar ( ): The putchar ( ) is a single character output function. It can display a single character on the screen at a time.

Syntax: putchar(ch);

Where'ch'is the variable of character datatype in which a single character data is stored.

puts(): The purpose of puts( ) is to print or display a string inputed by the gets().

Syntax: puts(s); (OR)

puts("Text line");

Where's' is the string variable it will display the string stored in 's'.

UNIT-II

CONTROL STATEMENTS

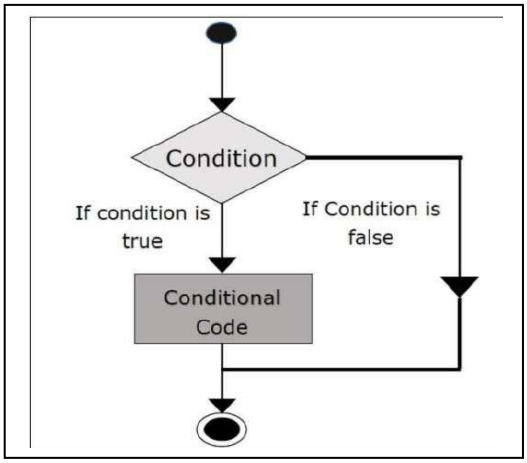
DECISION MAKING STATEMENTS: Decision-Making Statements and are used to evaluate one or more conditions and make the decision whether to execute a set of statements or not. These decision-making statements in programming languages decide the direction of the flow of program execution.

They provide conditions using boolean expressions that are evaluated to a true or false boolean value, they are sometimes referred to as conditional statements. A particular piece of code will run if the condition is true; if the state is false, the block will not run.The statement includes

a) if statement
b) if- else statement
c) nested if-else statement
d) if-else-if statement
e) switch case statement

a) if statement: Only one statement occurs in "if". It is having only one block.

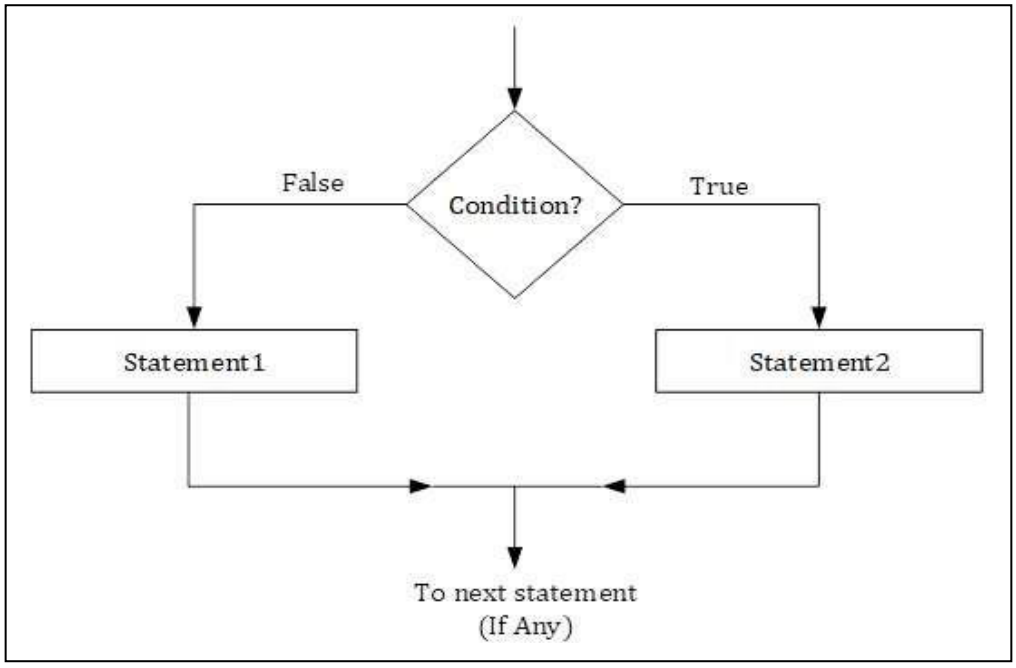
Syntax:- if (condition)
{
True Statement
}
Statement x;



First the condition will be checked. If the condition is true than the true statement block will be executed and after execution of this block, statement x will be executed. If the condition is false then only statement X will be executed.

b) if-else statement: This statements also has a single condition with two different blocks (i) is true block and other one is false block.

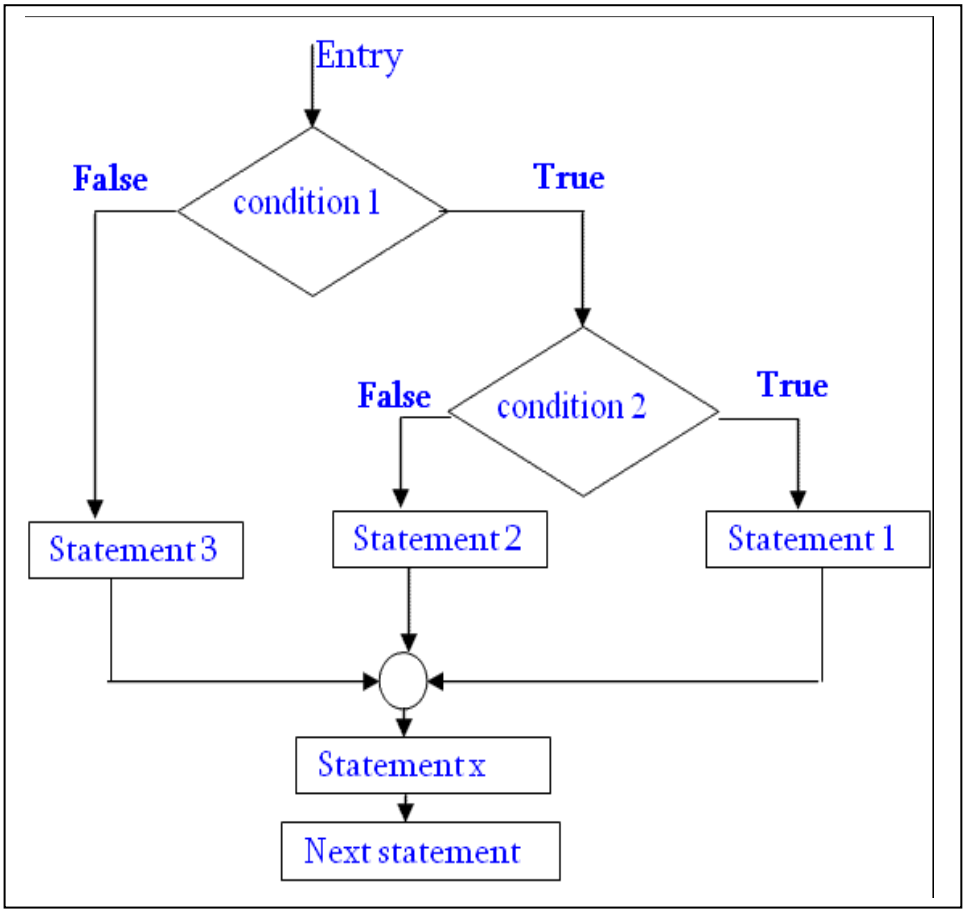
Syntax:- if (condition)
{
True statement;
}
else
{
False statement;
}
Statement x;



First the condition will be checked. If the condition is true then true statement block will be executed. Then control goes to statement x. If the condition is false then the false statement block will be executed then control goes to statement x.

c) nested if-else statement: When an if statement occurs within another if statement is called nested if else statement.

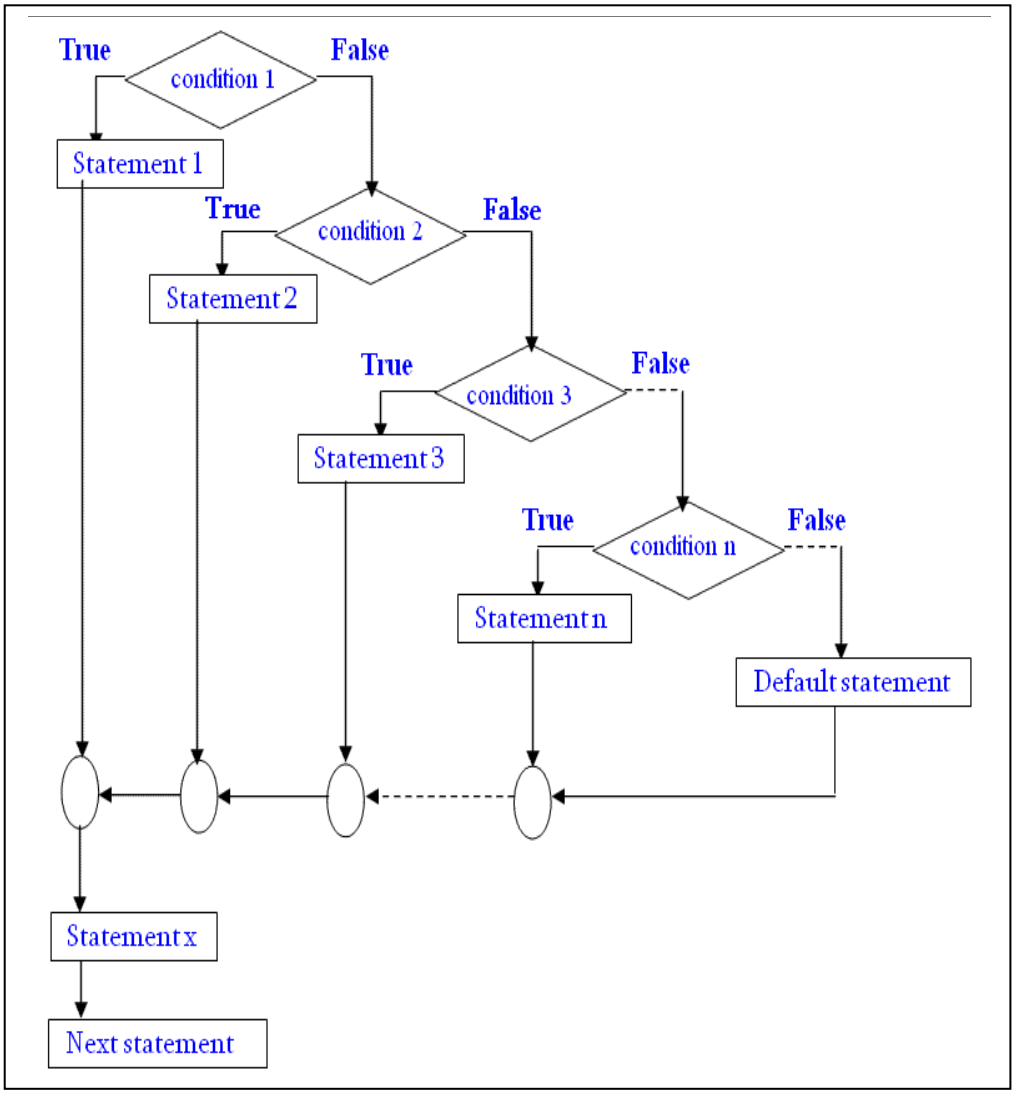
Syntax:-if(condition)
{
if(condition 2)
{
Statement 1;
}
else
{
Statement 2;
}
}
else
{
Statement 3;
}
Statement x;



Condition one will be checked if it is true condition two will be checked. If condition 2 is true then the statement one will be executed. If condition 2 is false the statement two will be executed.

d) else-if ladder statement: No. of conditions arise in a sequence, then we can use ladder if statement to solve the problem in the simple manner.

Syntax:- if(condition)
{
Statement 1;
}
elseif(condition 2)
{
Statement 2;
}
elseif(condition 3)
{
Statement 3;
}
elseif(condition)
{
Statement n;
}
else
{
default statement;
}
Statement x;



In this statement 1st condition will be checked, if it is true the statement 1 will be executed, if the condition 1 is false the condition 2 will be checked. If it is true statement 2 will be executed. Otherwise further next condition will be checked and this process will be continue till the end of the condition.

e) switch case statement: The switch case statement is a multi way branch statement. The tests whether expression matches one of the constant values. This switch case statement requires only one arguments which is checked with numbers of cases options.

Syntax:-

switch (expression)

{

case value 1: Statement 1;

break;

case value 2: Statement 2;

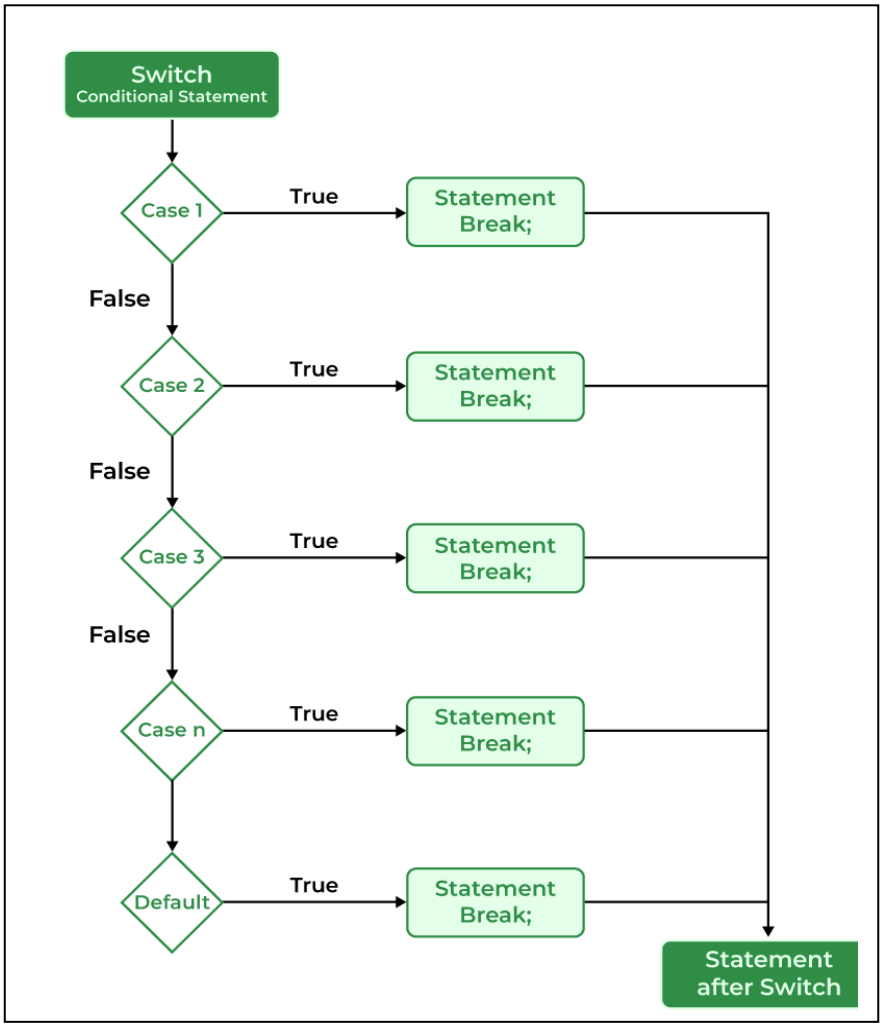
break;

case value n: Statement n;

break;

default: statement;

}



Write a program to find a day of week.

#include<stdio.h>

#include<conio.h>

void main( )

{

int day;

clrscr( );

printf("\n Enter day number:");

scanf("%d",&day);

switch(day)

{

case 1:

printf("\n sunday");

break;

```
case 2:
printf("\n monday");
 break;
case 3:
printf("\n tuesday");
 break;
case 4:
printf("\n wednesday");
break;
case 5:
printf("\n thursday");
break;
case 6:
printf("\n friday");
 break;
case 7:
printf("\n saturday");
break;
default:
printf("Invalid day number");
}
getch( );
}
```

Output:1

Enter day number: 1

Sunday

Output:2

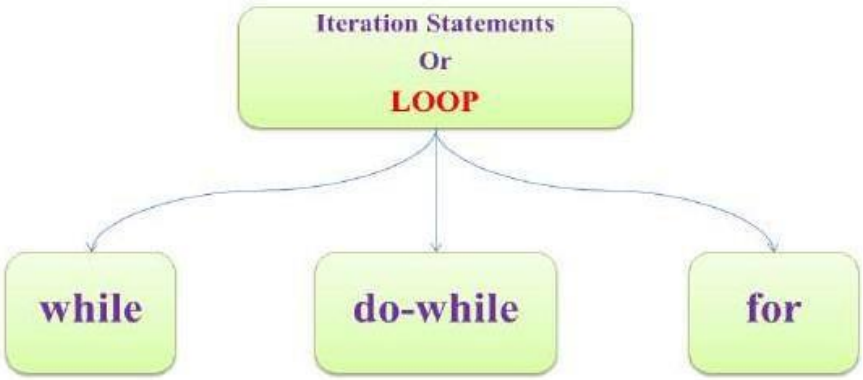
Enter day number: 8

Invalid day number

Iterative Statements (Looping Statements): Iterative statements are used to repeat their execution of list of statements depends on the value of integer expression.

(or)

A single statement (or) group of statements will be executed again and again in a program such type of processing is called loop.



"C" supports three types of iterative statements also known as looping statements. They are

- while-loop
- do-while loop
- for loop

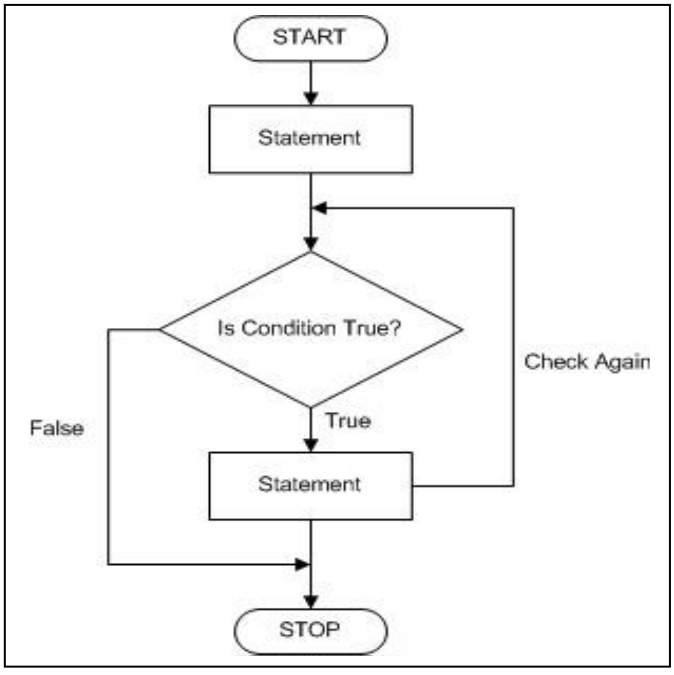
while loop: In while loop one (or) more statements are repeatedly executed. Until a particular condition is true. while loop is an entry control loop.

Syntax:-while (condition)

{

Block of statements;

}



In the while loop first the condition is tested. If the condition is true. Then only the body of statement will be executed. It will be executed again and again till condition becomes false. Otherwise if the condition is false. The control will be jump to false statement.

do-while loop: The do-while loop is similar to while loop. The only one difference in a do-while loop .The test condition is tested at the end of the loop. That means it is clear the body of loop gets executed atleast once. do-while loop is an exit control loop.

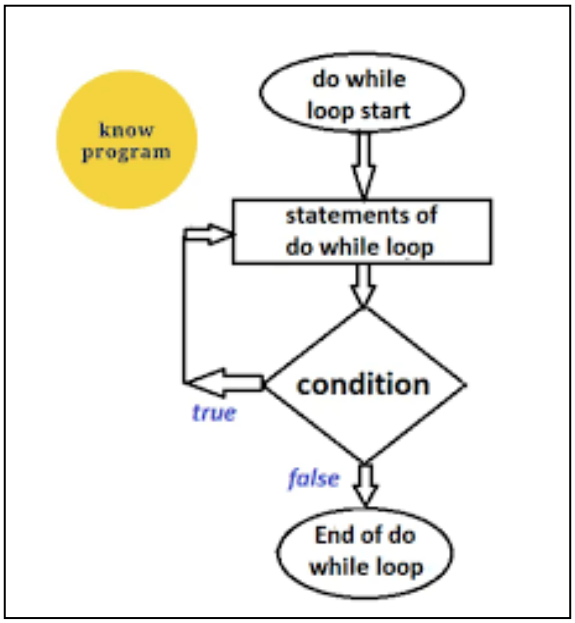
Syntax:-do

{

Block of statements;

}

while(condition);



for - loop: It is a looping statement which repeat again and again till it satisfy the defined condition.

It is on e step loop which initialize (or) initialization, check the condition and increment (or) decrement Step in the loop in a single statement.

Syntax:

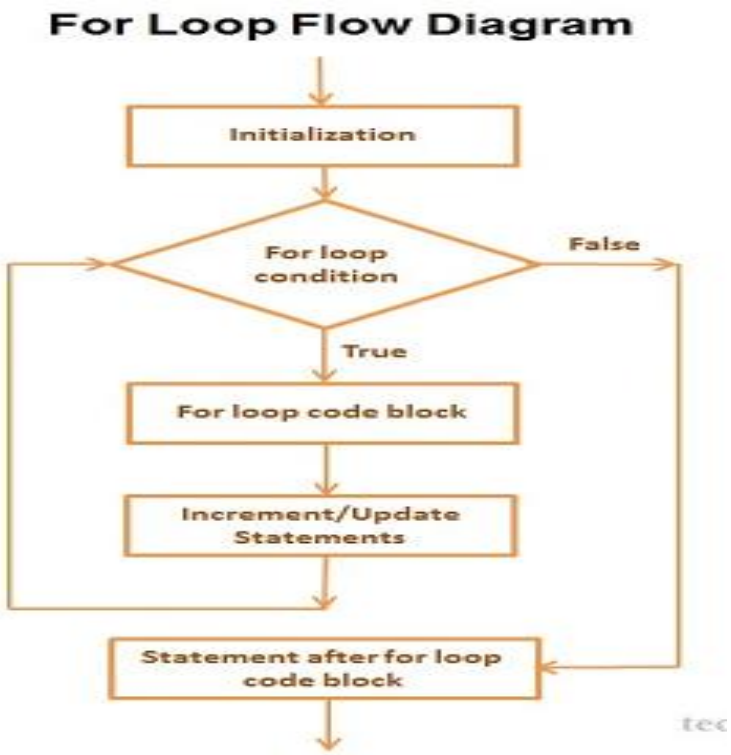
for (initialization; test condition; increment/decrement)

{

Body of loop;

}

Statement – x;



The execution for the "for loop" is as follows

1) Initialization of variable is done first

2) The value of variable is tested is tested is using test condition. If the condition is true the body of the loop is executed if the condition is false the loop is terminated.

3) When the body of loop is executed the control transfers back to the 1st statement for the last expression. That is increment/decrement.

4) After increment/decrement the value of the control again goes to the test condition. If the condition is true, the body of the loop is again executed. The process continuous until the test condition is false.

E.g: Write a C program to print 1 to N number using for loop.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
int i, n;
clrscr( );
printf ("Enter n value");
scanf ("%d", &n);
printf ("\n 1 to n numbers are:");
for (i=1; i<=n; i++)
```

```
{
printf (“\n%d”, i);
}
getch( );

}
```

Output:

Enter n value: 5

1 to n numbers are:1 2 3 4 5

Write A ‘C’ Program to Find the Fibonacci Series of A Given Number by using for-loop.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int a=0,b=1,c=1,n;
clrscr( );
printf("enter n value:");
scanf("%d",&n);
printf("\n The required fabnocci series is:");
printf("%d %d",a,b);
for(; c<=n;)
{
b=a;
a=c;
c=a+b;
printf("%d",c);
}
printf("\n");
getch( );
}
```

Output:

Enter n value: 5

The required fabnocci series is : 0 1 1 2 3 5 8

Example program for while – loop

```
#include<stdio.h>
 #include<conio.h>
void main( )
{
int i=1,n;
clrscr();
printf("Enter n value:");
scanf("%d",&b);
while(i<=n)
{
printf("\n Computer Science Department");
i++;
}
getch();
}
```

Output: Enter n value: 5
Computer Science Department
Computer Science Department
Computer Science Department
Computer Science Department
Computer Science Department

Write A 'C' Program to Find The Sum of Individual Digits of A Positive Integer.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int n,sum=0;
printf("\n Enter a Positive integer:");
scanf("%d",&n);
while(n>0)
{
sum=sum+n%10;
n=n/10;
}
printf("\n Sum of Individual Digits of A Positive Integer is %d", sum);
getch( );
```

```
}
```

Output :

Enter a Positive integer : 123
Sum of Individual Digits of A Positive Integer is :6

Write A 'C' Program to Check Whether A Number is Armstrong or Not.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
intm,n,x,sum=0;
clrscr( );
printf("Enter n value:");
scanf("%d",&n);
m=n;
while(n>0)
{
x=n%10;
sum=sum+x*x*x;
n=n/10;
}
if(m==sum)
printf("\n Given number is armstrong",m);
else
printf("\n Given number is not armstrong",m);
getch( );
}
```

Output: 1

Enter n value: 153
Given number is Armstrong

Output: 2

Enter n value:123
Given number is not Armstrong

Example program for do while – loop

```
#include <stdio.h>
int main ( )
{
 int a = 1;
 do
 {
  printf("do while loop\n");
  a++;
   }
 while( a <=5 );
 printf("End of loop");
 return 0;
}
```

Output :

```
do while loop
do while loop
do while loop
do while loop
do while loop
End of loop
```

JUMP CONTROL STATEMENTS

In C, jump statements are used to jump from one part of the code to another altering the normal flow of the program. They are used to transfer the program control to somewhere else in the program.

Types of Jump Statements in C

There are 4 types of jump statements in C:

1. break
2. continue
3. goto
4. return

1. break in C

The break statement exits or terminates the loop or switch statement based on a certain condition, without executing the remaining code.

break is a keyword in “c” statement is used to terminate the loop i.e., the control comes out from the current loop. When over the break keyword is encountered. It is also called as loop terminator.

The break statement is widely used with for loop, while loop & do-while loop.

Syntax : break;

Uses of break in C

The break statement is used in C for the following purposes:

1. To come out of the loop.
2. To come out from the nested loops.
3. To come out of the switch case.

The statements inside the loop are executed sequentially. When the break statement is encountered within the loop and the condition for the break statement becomes true, the program flow breaks out of the loop, regardless of any remaining iterations.

2. Continue in C

The continue statement in C is used to skip the remaining code after the continue statement within a loop and jump to the next iteration of the loop. When the continue statement is encountered, the loop control immediately jumps to the next iteration, by skipping the lines of code written after it within the loop body.

The continuous statement is opposite to break statement. It appears in the body of the loop. Whenever a continuous statement appears the rest of the statements in the loop are skipped. And it continuous with the next iteration of current loop.

Syntax : continue;

3. Goto Statement in C

The goto statement is used to jump to a specific point from anywhere in a function. It is used to transfer the program control to a labeled statement within the same function.

It doesn’t require any condition goto is a keyword in ‘c’ this statement process control any where in the program. i.e., control is transferred to another part of program without testing any condition. The user has to define goto statement as follows.

Syntax: goto label;

4. Return Statement in C

The return statement in C is used to terminate the execution of a function and return a value to the caller. It is commonly used to provide a result back to the calling code.

Syntax: return *expression*;

Example program on jump statements

```
#include <stdio.h>
int main( )
{
 int choice;
 printf("Select an option:\n");
 printf("1. Print 'Hello'\n");
 printf("2. Print 'World'\n");
 printf("3. Exit\n");
 choice=3;
 switch (choice)
{
 case 1:
 printf("Hello\n");
 break;
 case 2:
 printf("World\n");
 break;
 case 3:
 printf("Exiting...\n");
 goto end; // Jump to the 'end' label to exit
 default:
 printf("Invalid choice\n");
 break;
 }
 // This is where the 'end' label is defined
 end:
 printf("Program ends here.\n");
 return 0;
}
```

Output:

Select an option:

1. Print 'Hello'

2. Print 'World'

3. Exit

3

Exiting...

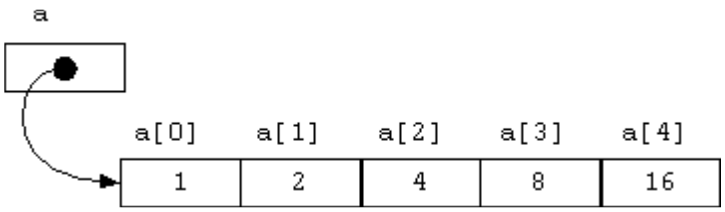
Program ends here.

UNIT-III

DERIVED DATATYPES IN C

ARRAY

Definition: Array is a collection of collection of similar data items (or) elements. These data elements have same data type. The elements of array are stored in sequence memory location and referred by index.



Arrays are used to reduce confused and length of the program. Each element of an array can be accessed with the array name followed by a subscript which includes in square brackets .The subscript provided from zero(0) for the first element ,increased by one to the next element.

Declaration of an Array: An array must be declared before its use. Declaring an array means we need to specify.
Syntax:- Data type Array name [size] ;
Example:

int a[5];

float c[10];

Datatype : What kind of data value it can store .E.g.: integer, float, char, double….etc.,

Array Name : To identify an array name.

Size: Minimum number of values that an array can store.

Initialization of Arrays: An elements of array can be initialized at the time of declaration. When an array is initialized we need to provide a value for every element in the array.
Syntax: Data type Array-name [size]={list of values};
Example: int a[5]={1,2,3,4,5};

Types of Arrays: There are three types of arrays.
1. One – Dimensional Array
2. Two – Dimensional Array

3. Multi – Dimensional Array

1) One – Dimensional Array: A list of items can be given one variable name using only one subscript and such variable is called a single subscripted variable (or) one dimensional array .

Declaration of One Dimensional Array: Like any other variable, array must be declared before they are used .

Syntax: Data type array name [size];

Example:

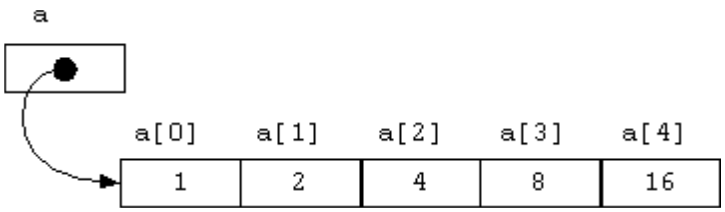
int a[5];

float c[10];

Initialization of One Dimensional Array: The array can be initialized at the time of declaration .

Syntax: Data type array name [size] = {list of values};

Example: int a[5]={1,2,4,8,16};



In above diagram, array name is “a”, array size is 5,

Array index (or) address represented by a[0],a[1],a[2],a[3],a[4].

Array values are (1,2,4,8,16).

E.g.: 1

```
#include<stdio.h>
int main ( )
{
  int a[5] = {10,20,30,40,50};
  int i;
  printf ("elements of the array are");
  for ( i=0; i<5; i++)
  printf ("%d", a[i]);
}
```

Output

Elements of the array are
10 20 30 40 50

E.g.:2

```
#include<stdio.h>
main ( )
{
  int a[5],i;
  printf ("enter 5 elements");
  for ( i=0; i<5; i++)
  scanf("%d", &a[i]);
  printf("elements of the array are");
  for (i=0; i<5; i++)
  printf("%d", a[i]);
}
```

Output

The output is as follows −
enter 5 elements 10 20 30 40 50
elements of the array are : 10 20 30 40 50

E.g.:3 Write A 'C' Program to Find Both The Largest and Smallest Number in A List of Integer Values.

```
#include<stdio.h>
int main()
{
int a[20],i,n,large,small;
printf("\n Enter the Array Size:");
scanf("%d",&n);
printf("\n Enter the Array Elements:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
large = small =a[0];
for(i=1;i<n;i++)
{
if(a[i]>large)
{
```

```
large=a[i];
}
if(a[i]<small)
{
small =a[i];
}
}
printf("\n The Smallest Element is %d",small);
printf("\n The Largest Element is %d",large);
return 0;
}
```

Output:

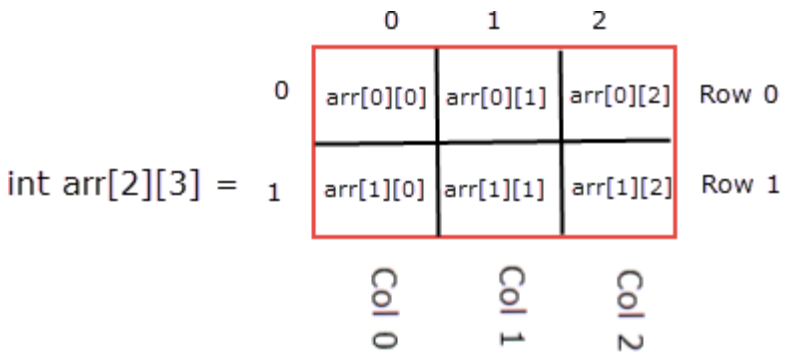
Enter the Array Size :7
Enter the Array Elements : 6 8 2 4 3 9 1
The Smallest Element is : 1
The Largest Element is : 9

2)Two-Dimensional :A two dimensional array is specified using two subscripts. Where one subscript is denotes rows and other subscript is denotes column. It is also known as double dimensional array (or) two dimensional array.

Declaration of Two-Dimensional Array:

Syntax: Data type array name [rows][column];

E.g.: int arr[2][3];



Initialization of Two-Dimensional Array: The two dimensional array can be initialized at the time of declaration.

Syntax: Data type array name[row size][column size] = {list of values};

Example: int arr[2][3]={5,7,10,2,1,3};

Here the value assigned to the array is as follows.
arr[0][0]=5 arr[1][0]=2
arr[0][1]=7 arr[1][1]=1
arr[0][2]=10 arr[1][2]=3

Write A 'C' Program on Addition of Two Matrices.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int A[20][20],B[20][20],C[20][20],i,j,m,n;
clrscr( );
printf("Enter range:");
 scanf("%d%d"&m,&n);
printf("Enter A matrix values:");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&A[i][j]);
}
}
printf("Enter B matrix values:");
 for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&B[i][j]);
}
}
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
C[i][j]=A[i][j]+B[i][j];
}
}
printf("addition of two matrices:");
for(i=0;i<m;i++)
```

```
{
for(j=0;j<n;j++)
{
printf("%3d",C[i][j]);
}
printf("\n");
}
getch();
}
```

Out put:

Enter range: 2 2
Enter A matrix values :1 1 1 1
Enter B matrix values :4 4 4 4
Addition of two matrices:5 5 5 5

STRING

Definition: A string is a group of characters. A string in 'C' defined as any group of characters enclosed between **double quotes(" ")** marks.

The string can be of any length, the end of the string is marked with the single character('\0') the null character.

The strings are actually one dimensional array of characters terminated by null character. The character arrays are declared in c in a similar manner of numeric array.

Declaration:

Syntax: char string name[size];

Strings can be initialized at the time of declaration at the same time initializations at the time of declaration the string can be initialized in any one of the following manner.

E.g: char name[10] = "Krishna";

char name[10] = {'k','r','i','s','h','n','a,};

"C" library supports a large number of string handling functions. Following are the most commonly used String Handling Functions.

String Handling Functions:

The 'C' library supports a large number of string handling functions that can be used to manipulate the string in many ways you must include the string header file in your program

String handling functions are:

1. strcat ( ) concatenation
2. strcpy( ) copy
3. strlen( ) length
4. strcmp( ) compare
5. strrev( ) reverse
6. strupr( ) upper case
7. strlwr( ) lower case

1) strcat( ) : The strcat( ) function is used to joins two strings.

Syntax: strcat(str1,str2);

str1 and str2 are two strings. When the function strcat() is executed str2 is appended to str1.

2) strcpy( ) : This function is used to copy one string into another string it accepts two strings as arguments.

Syntax: strcpy(target,source);

If first argument is generally an identifier, that represents the strings. The second argument can be a string constant. The function copies the string of source to target.

3) strlen( ) : This function counts and returns the number of characters in a string.

Syntax: variable= strlen(string);

Where variable is an integer variable, which receives the value of the length of the string the argument is a string constant. [51]

4) strcmp( ) : This function compares two strings and returns a integer value.

Syntax: strcmp(str1,str2);

It returns zero if the strings are equal, greater than zero if string 1 is bigger than string 2 and less than zero if string1 is less than string2.

5) Strrev(): The purpose of this function is two reverse a string this function takes string variable as a single argument here the first character becomes last and the last character becomes first in the string.

Syntax: strrev(string);

E.g: char str1[10]="degree";

Printf("reverse string of str1=%s",strrev(s1));

O/P:- reverse string of str1=eerged

6) Strupr(): The purpose of this function is to convert into upper case this function converts all the character of a string from lower case to upper case.
Syntax:-strupr(string);

E.g: char str1[10]=”degree”;

Printf(“str1 is converted into upper case :%s”,strupr(str1));

O/P:- str1 is converted into upper case :DEGREE

7) Strlwr(): The purpose of this function is to convert string into lower case this function converts all the character of a string from upper case to lower case.
Syntax:-strlwr(string);

E.g: char str1[10]=”DEGREE”;

Printf(“str1 is converted into lower case :%s”,strupr(str1));

O/P:- str1 is converted into lower case :degree

E.g: Write a ‘C’ Program to perform various String Operations.

```
#include<stdio.h>
#include<conio.h>
 #include<string.h>
 void main( )
{
char s1[20]="govt"; char s2[20]="college";
char s3[20]="ORGANIZATION";
clrscr( );
printf("\n Length of string s1 is:%d",strlen(s1));
strcat(s1,s2);
printf("\n Join two strings s1 and s2:%s",s1);
printf("\n String s2 convert to uppercase:%s",strupr(s2));
 printf("\n String s3 convert to lowercase:%s",strlwr(s3));
 strrev(s1);
printf("/n Reverse of string s1 is:%s",s1);
strcpy(s1,s2);
printf("\n Copy of string is:%s",s1);
getch( );
```

}

Out put:

Length of string s1 is:4

Join two strings s1 and s2:govtcollege

String s2 convert to uppercase: COLLEGE

String s3 convert to lowercase: organization

Reverse of string s1 is : tvog

Copy of string is : college

2. Example program for string reverse function in 'C'

```
#include <stdio.h>
#include <string.h>
int main( )
{
   char str[ ] = "Hello world";
   printf("The string is : %s\n", str);
   strrev(str);
   printf("The string after using function strrev( ) is : %s\n", str);
   return 0;
}
```

Output:

The string is : Hello world

The string after using function strrev( ) is : dlrow olleH

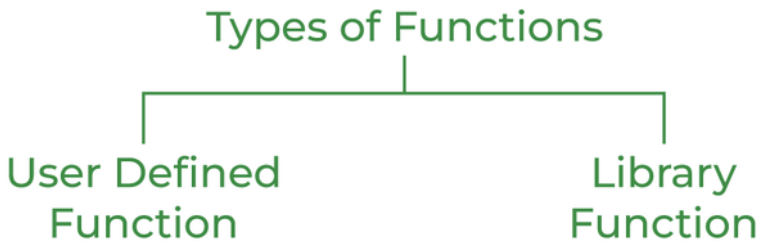
UNIT – IV

FUNCTIONS

Function: A function is a group of statements that together perform specific tasks. A large program can be split into smaller segments. So that it can be efficiently solved.

The function are categorized into two types

- Pre-defined functions (or) Library functions.
- User-defined functions



The major difference between the user defined functions and pre-defined functions are not required to be written by user. While writing a program. Many program required to a particular group of unstructured accessed repeatedly. From the different place with in the program. This repeated instructions can be placed in a single function and can be accessed whenever necessary.

Function is a group of statements that carries out to specific task. Every c program starts with atleast one function. Which is main( ). The main( ) calls another to share the work.

Library functions are pre-defined se of functions they task is limited. Eg:-printf(), scanf(), sqrt()….etc.

These functions cannot modify. User defined functions are define by user. According to our requirement are called user defined functions.

Advantages of function:

1. It provides reusability. Because function calls many times.

2. Generally large programs can be divided into smaller programs.

3. It is too easy to modify.

4. It is easy to debug and find out the errors.

Function declaration:

Function declaration consists of three components such as return type, function name and arguments being passed. The function declaration must be ends with a semi column (;).

Syntax:-

return type function name (arg1, arg2 ……arg n);

int sum(int, int); float multi();

Function definition: The function definition consists of two components. The first line and the body of the function. The function definition is same as function declaration except that is does not end with a semi column(;)

Syntax:-

return type function name (arg 1,arg 2…..arg n)

{

function body; return(expression);

}

The body of the function may contain local variables, statements, expressions and also a return type.

Function call (or) Calling function: The function call statement invokes the function. When a function is invoked the compiler jumps to the called function to execute the statements that are a part of function. Once the called function is executed, the program control passes back to the calling function.

Syntax:

function name(variable 1, variable 2….);

Example:

```
main()
{
…..
…..
abc (x,y,z); // Call Function;
……
……
}
abc(i,j,k) // Called Function;
{
```

……
……
return();
}

Function Prototype in C

The C function prototype is a statement that tells the compiler about the function's name, its return type, numbers and data types of its parameters. By using this information, the compiler cross-checks function parameters and their data type with function definition and function call.

Function prototype works like a function declaration where it is necessary where the function reference or call is present before the function definition but optional if the function definition is present before the function call in the program.

Syntax

return _ type function _ name(parameter_list);

where,

- **return_type**: It is the data type of the value that the function returns. It can be any data type int, float, void, etc. If the function does not return anything, void is used as the return type.
- **function_name**: It is the identifier of the function. Use appropriate names for the functions that specify the purpose of the function.
- **parameter_list**: It is the list of parameters that a function expects in parentheses. A parameter consists of its data type and name. If we don't want to pass any parameter, we can leave the parentheses empty.

Types of Functions:

Depending upon the arguments return value sends back to the calling functions based on this the functions are divided into four types.

1. A function without arguments and without return a value.
2. A function with arguments and with return value.
3. A function with arguments and without return value.
4. A function without arguments and with return value.

1. A function without arguments and without return value:

- ❖ In this function neither the data is passed through the calling function not the data sent back from the called function.

- ❖ There is no data transfer between calling & the called function. The function is only executed & nothing is return (or) obtain.
- ❖ These functions are independent. They read data values& print results in the same block.
- ❖ Such functions may be useful to print same messages, draw a line or split the lines etc.

Syntax:-

Calling function	Analysis	Called function

Void fname();
 	
statements;
{
fname();
statements();
}
	No arguments are passed
	
	Novaluesare passed
		void fname()
		{
		
		Statements;
		}

Example:-

```
#include<stdio.h>
 void main( )
{
void msg( ); /* calling function*/
msg();
}
void msg( ) /* called function */
{
Printf("welcome");
}
```

Output: welcome

2. function with arguments and with return value:

- ❖ In such functions the copy of actual arguments is passed to formal arguments.
- ❖ The return value is sent back to the called function.
- ❖ In such functions the data is transferred between calling and the called function i.e., communication between is made.

Syntax:-

Calling function	Analysis	Calledfunction

Void main( )
{
int fname(actual arguments list);
f name(actual arguments list);
}
	Arguments are passed.
	Values are sent back
		int fname(formal arguments list)
		{
		Statements;
		return value;
		}

Example:-

/*addition of two numbers*/

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int add(int,int);
 int a,b,c;
clrscr( );
printf("Enter two values");
scanf("%d%d", &a,&b);
c=add(a,b);
printf("sum=%d",c);
getch( );
}
int add(int x,int y)
{
int z;
 z=x+y;
return z;
```

}

Output:

Enter two values : 1 3

Sum = 4

Function with arguments and without return value:

- In such functions the arguments are passed through the calling function.
- The calling function operates the values but no result is sent back.
- Such functions are dependent on the calling function there is no gain to the main( ) function.

Syntax:-

Calling function	Analysis	Called function

void main()
{
Void fname(actual arg datatype list1);
statements;
fname(actual arg list1);
statements;
}
	Arguments are passed
	
	No values are sent back
		Void fname(formal arglist)
		{
		Statements;
		}

Example:-

```
/*write a program to given number is even or odd*/
#include<stdio.h>
#include<conio.h>
void main()
{
void evenodd(int);
 int a;
clrscr( );
printf("Enter one value:");
scanf("%d",&a);
evenodd(a);
 getch();
```

```
}
Void evenodd(intn)
{
if(n%2==0)
printf("Given number is even:");
else
printf("Given number is odd");
}
```

Output:

Enter one value:-4

Given number is even

Function without arguments and with return value:

- In such type of function no arguments are passed through the main function but the called function returns the value
- The called function is independent. It reads the value input from the keyboard or generates from the initialization and returns the value
- Here both the calling & called function are communicated with each other.

Syntax:-

Calling function	Analysis	Called function

Void main( )
{
int fname();
statements;
fname;
statements;
}
	No arguments are passed
	
	Values are sent back
		int fname()
		{
		Statements;
		return value;
		}

Example program:

```
/*Sum of three numbers*/
 #include<stdio.h>
#include<conio.h>
```

```
void main( )
{
int sum();
int s;
clrscr( );
s=sum( );
printf(“Sum=:%d”,s);
}
sum( )
{
intx,y,z;
printf(“Enter three values :”);
scanf(“%d%d%d”,&x,&y,&z);
 return(x+y+z);
}
```

Output:

```
Enter three values : 123
Sum=6
```

Nested Function: A nested function is a function defined inside the definition of another function. In the C programming language, nesting occurs when one or more functions are used within another function. In the C programming language, we cannot define a function within another function (nested function is not supported by C language).

Example :

```
#include<stdio.h>
 int my_fun( )
{
  printf("check_fun function");
  printf("\n");
 }
int main( )
{
      my_fun( );
      printf("Main Function\n");
      printf("Done");
}
```

Output :

```
check_fun function
Main Function
Done
```

Return Statement

return statement ends the execution of a function and returns the control to the function from where it was **called**. The return statement may or may not return a value depending upon the return type of the function. For example, int returns an integer value, void returns nothing, etc.

we can only **return** a **single value** from the function using the return statement and we have to declare the data_type of the return value in the function definition/declaration.

Syntax:

return return_value;

There are various ways to use return statements. A few are mentioned below:

1. Methods not returning a value

one cannot skip the return statement when the return type of the function is non-void type. The return statement can be skipped only for void types.

a) Not using a return statement in void return type function:

While using the void function, it is not necessary to use return as the void itself means nothing (an empty value).

Syntax:

```
void  func()
{
  ----
  ----
  ----
}
```

Example:

```
#include <stdio.h>
void Print( )
{
   printf("Welcome to C language");
}
int main()
{
```

```
    Print( );
    return 0;
}
```

Output: Welcome to C language

b) Using the return statement in the void return type function:

As void means empty, we don't need to return anything, but we can use the return statement inside void functions as shown below. Although, we still cannot return any value.

Syntax:

```
void func()
{
    return;
}
```

Example:

```
#include <stdio.h>
  void Print( )
{
    printf("Welcome to C language ");
    return;
}
int main( )
{
    Print( );
    return 0;
}
```

Output: Welcome to C language

2. Methods returning a value

For functions that define a non-void return type in the definition and declaration, the return statement must be immediately followed by the return value of that specified return type.

Syntax:

```
return-type func()
{
    return value;
}
```

Recursion:

- ‘C’ language supports recursive features i.e., function is called repeatedly by itself. The recursion can be directly (or) indirectly.
- The direct recursion function called itself. In indirect recursion function call another function. Then the called function calls calling function.

Example:-

Write A ‘C’ Program to Find Out Factorial of Given Number Using Recursion.

```
#include<stdio.h>
 #include<conio.h>
 void main( )
{
int fact(int);
clrscr( );
printf(“Enter n value”);
scanf(“%d”, &n);
f=fact(n);
printf(“\n The factorial of %d is %d”, n, f);
getch( );
}
int fact(int n)
{
int f;
 if(n==0)
{
return n;
else
f=n*fact(n-1);
return f;
}
```

Output:-

Enter n value: 2

The factorial of 5 is 120

Parameters passing methods with an example programs

There are two ways to pass arguments or parameters of functions
1) Call by value
2) Call by reference

1) Call by value :

Call by value in which values of variables are passed by the calling function to the called function.

Example program ;

/* Swapping of two numbers in Call by value*/

```
#include<stdio.h>
#include<conio.h>
void swap(int,int);
void main( )
{
int a,b;
clrscr( );
printf("Enter a,b values :");
scanf("%d%d",&a,&b);
printf("\n Before swapping in main a=%d \t b=%d \t",a,b);
swap(a,b);
printf("\n After swapping in main a=%d \t b=%d \t",a,b);
getch( );
}
void swap(int p,int q)
{
int temp;
temp=p;
p=q;
q=temp;
printf("\n After swapping in function p=%d \t,q=%d \t",p,q);
}
```

Output :

```
Enter a,b values : 2 3
Before swapping in main a=2 b=3
After swapping in function p=3,q=2
After swapping in main a=2,b=3
```

2) Call by reference :

Call by reference in which address of variable passed by the calling function to called function.

Example:-

/* Swapping of two numbers in Call by reference*/

```
 #include<stdio.h>
#include<conio.h>
void swap(int *,int*);
 void main( )
{
int a,b;
clrscr( );
printf("Enter a,b values :");
scanf("%d%d",&a,&b);
printf("\n Before swapping in main a=%d \t b=%d \t",a,b);
swap(&a,&b);
printf("\n After swapping in main a=%d \t b=%d \t",a,b);
getch( );
}
void swap(int *p,int *q)
{
int *temp;
*temp=*p;
*p=*q;
*q=*temp;
printf("\n After swapping in function p=%d \t, q=%d \t",*p,*q);
}
```

Output :

```
Enter a,b values : 2 3
Before swapping in main a=2 b=3
 After swapping in function p=3,q=2
 After swapping in main a=3,b=2
```

Storage Classes

- The storage class determine the part of memory when the variable would be stored.
- Each variable has a storage class which decides scope, visibility and life time of that variable.

- A variable declared inside of the main function is called “local variables”.
- A variable declared outside of any function is called “global variable”.

There are 4 types of storage classes in ‘c’

1. Automatic storage class
2. External storage class
3. Static storage class
4. Register storage class

1. Automatic storage class: Automatic variables are defined inside a function. A variable declared inside the function without use storage class, name by default is an automatic variable. Automatic variables is also called as local variable. It is declared to use “auto “keyword.

Syntax:

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

2. External storage class: The variable are declared outside of the function the external variable is also called as “global variables” this variables are used any function in the program. In case external and auto variables are declared with the same name in the program. The first priority is given to the auto variables. It is declared to use “extern” keyword.

Syntax:

```
void main( )
{
extern int a=7;
}
```

3. Static storage class: The static variable may be any internal or external type. It is depending upon where it is depending upon where it is declared. If it is declared outside of the function. It will be static variable (or) global variable. In case it will be declared inside of the function. It will be auto variable or local variable. Optional (when variable is declared as static its garbage value is removed and initialize to null value). It is declared to use “static” keyword.

Syntax:

```
void main( )
{
int x;
```

static int y;
printf("x=%d and y=%d",x,y);
}

Output:

X=1239 ↓ Garbagevalue and y=0 ↓ nullvalue

4. Register storage class: A variable is declared to use "register" keyword is called register storage class variable. When a variable is declared using register keyword. Then the value of that variable is directly stored the C.P.U register.

Syntax:

```
void main( )
{
register int;
 clrscr( );
for(i=0;i<10;i++)
printf("%d", i);
getch( );
}
```

Storage Specifier	Storage	Initial value	Scope	Life
auto	Stack	Garbage	Within block	End of block
extern	Data segment	Zero	global Multiple files	Till end of program
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block

Pointers: A pointer is a variable, which contains the address of another variable. A pointer provides an indirect way of accessing the value of a data item.

Declaring a pointer variable: A pointer is a variable should be declared before they are used.

In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the **(*) dereference operator** before its name.
The general syntax used for the declaration of pointer is as.
Syntax: Data-type *pointer-variable;

Example: int ***ptr**;

Where data type may be integer (int), real(float),character(char) or double. Also here (asteric sign) means it is pointer operator and pointer variable is any variable linked with'*' sign.

Pointer Initialization: Pointer initialization is the process where we assign some initial value to the pointer variable. We generally use the (&) address of operator to get the memory address of a variable and then store it in the pointer variable.

Example:
int var = 10;
int * ptr;
ptr = &var;

Advantages and Disadvantages of Pointers:

Advantages:

1. Pointers increase the execution speed of the C-Program and are more efficient.
2. Pointer accesses the memory elements very easily.
3. Pointer reduces the length and complexity of the program.
4. By using pointer, we can declare lesser number of variables in memory.
5. Pointers access the memory elements very easily.
6. We can pass arguments to functions by reference.

Disadvantages:

1. Pointers are slower than normal variables.
2. If used incorrectly pointer leads to bugs (errors).
3. An erroneous input leads to erroneous output.

Pointer Arithmetic

The Pointer Arithmetic refers to the legal or valid arithmetic operations that can be performed on a pointer. It is slightly different from the ones that we generally use

for mathematical calculations as only a limited set of operations can be performed on pointers. These operations include:

- Increment in a Pointer
- Decrement in a Pointer
- Addition of integer to a pointer
- Subtraction of integer to a pointer
- Subtracting two pointers of the same type
- Comparison of pointers of the same type.
- Assignment of pointers of the same type.

Pointers and Arrays: pointers and arrays are closely related. An array name acts like a pointer constant. The value of this pointer constant is the address of the first element.

Difference between ARRAY and POINTERS:

ARRAYS	POINTERS
1. An array is a collection of similar datatypes. 2. Syntax: Datatype arrayname[size]; 3. Example:-int a[10]; 4. Array can be initialized at definition. 5. They are static in nature. Once memory is allocated, it can be resized. 6. The assemble code of array is different than pointer.	1. A pointer is a variable which contains the address of another variable. 2. Syntax: Datatype *pointer variable; 3. Example: int*x; 4. Pointer can be initialized at definition. 5. Pointer is dynamic in nature the memory allocation can be resized. 6. The assemble code of pointer is different than array

Function pointers: Every function has an address function pointers are pointer variables that point to the address of a point function pointers can be declare assign values and used to access the functions.

In order to declare a pointer to a function the name of function must be enclosed between parenthesis and an asteric (*) symbol is inserted before the name.

Syntax:- the syntax of declaring a function pointer is

retune type(*function pointer name)(arg list);

Ex:-int (*func)(int a,float b);

UNIT-V

Dynamic Memory Manegement

- The process of allocating memory at runtime is known as dynamic memory allocation.
- With the static memory allocation some amount of memory is unused. This unused memory is actually empty. But it filled with garbage values
- To avoid this problem DMA technique was introduced
- DMA allows allocating memory at run time. Hence there will be no wastage of memory
- We can achieve DMA with following functions and which are defined as<alloch.h> or <stdio.h>header files

1. malloc( )
2. calloc( )
3. Realloc( )
4. free( )

1.malloc( ):

- Malloc allocates requested size of bytes and return void pointer to the first byte of the allocated space.
- Malloc( ) allocates single block of requested memory.
- Syntax:- declare a pointer
- Pointer name=(cast type*)malloc(byte size);

2. calloc( ):

- Calloc ( ) allocates multiple blocks of requested memory. Each of the same size and returns starting address of the memory to the pointer variable. This is widely used in array.
- Syntax:-declare a pointer
- Ptr=(int*)calloc(n,2);

3. realloc( ):

- This Realloc( ) is used to make changes in the memory location(i.e.)increase or decrease is already allocated by malloc( )or calloc( ).
- This realloc reallocates the memory.
- Syntax:-declare a pointer
- Pointer variable=(cast type*)realloc(pointer name, new size);

4. free():

- It is essential to clean up memory at the end of the program.
- In the memory location filled with the garbage values so it is very much required to clean up.
- Syntax:-void tree(void*back);

STRUCTURE:

Definition: A Structure is a collection of one or more variables of different data types grouped together under a single name.

A Structure is similar to a record it stores related information about variable. Structure is a basically user defined data type that can store related information.

The major difference between a structure and array is that an array contains related information of some data type, but structure is a collection of variables with different data types.

Syntax:

```
struct  structure_name
{
data type var-name1;
data type var-name2;
};
```

For example: To declare a structure for a student with related information roll no, name, fee section --,is declared as exactly in the example.

Ex:-
```
struct student
{
int  rollno;
char name[20];
 float fees;
};
```

Eg:-
```
struct student
{
int  rollno;
char name[20];
char course[20];
 float fees;
```

};
struct student stud1={01,”ravi”,”bsc”,4500};

Accessing the members of a structure: Each member of a structure can be used just like a normal variable. A structure member variable is generally accessed to using a “.” dot operator.
Syntax:- struct-var.member-name
the dot(.)operator is used to select a particular member of the structure.
Eg:- std1.rno=01;
std1.name=”rahul”;
std1.course=”bsc”;
std1.fees=4500;

Nested structure: A structure can be placed with in another structure. A structure that contains another structure as its member is called nested structure
The easier and clear way is to declare a nested structure separately and then group them in a high level
structure (from lower level to higher level) the nesting must be done from inside out.

Eg:-Write a c program to read and display student information using structure with in a structure

```
#include<stdio.h>
#include<conio.h>
int main( )
{
struct dob
{
int day;
int month;
int year;
};
struct student
{
int roll-no;
char name[30];
```

```
float fees;
struct dob date;
};
struct student std1;
clrscr( );
printf("\n enter roll-no:");
scanf("%d",&std1.roll-no);
printf("\n enter name:");
scanf("%d",&std1.name);
printf("\n enter fees:");
scanf("%f",&std1.fees);
printf("\n enter date of birth:");
scanf("%d%d%d",&std1.date.day,&std1.date.month,&stud1.date.year);
printf("\n*********student details*********");
printf("\n rollno=%d",std1.rollno);
printf("\n name=%s",std1.name);
printf("\n fees=%f,std1.fees);
printf("\n dob=%d-%d-%d", std1.date.day,std1.date.month,stud1.date.year);
getch();
return 0;
}
```

Output:

```
enter roll-no:2
enter name:"Victor"
[67]
enter fees:5000
enter dob:1-12-1998
******student details**********
roll-no=2
name=Victor
fees=5000
dob=1-12-1998
```

UNIONS

A **union** is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple purpose.

Declaring a union: The syntax for declaring a union is same as that of declaring a structure using the keyboard
struct and the union is used **union** keyword

Syntax:

union union-name

{

data type var-name;

data type var-name;

};

Ex:-write a c program to read and display student information using unions

```
#include<stdio.h>
#include<conio.h>
int main()
{
union student
{
int roll-no;
char name[30];
float fees;
char dob[30];
};
union student std1;
clrscr( );
printf("\n enter roll-no:");
scanf("%d",&std1.roll-no);
printf ("\n enter name:");
scanf ("%d",&std1.name);
printf ("\n enter fees:");
scanf ("%f",&std1.fees);
printf ("\n enter date of birth :");
```

```
scanf (“%d”,&std1.dob);
printf(“\n*********student details*********”);
printf(“\n rollno=%d”,std1.rollno);
printf(“\n name=%s”,std1.name);
printf(“\n fees=%f,std1.fees);
printf(“\n dob=%d,std1.dob);
getch();
return 0;
}
```

Output:

```
enter roll-no:1
enter name: raju
enter fees:4500
enter dob: 17-11-1998
******student details**********
roll-no=1
name=raju
fees=4500
dob=17-11-1998
```

Difference between **STRUCTURE** and **UNION.**

STRUCTURE	UNION
1. Every member has its own memory space	1. All member use the same memory space
2. Keyword **struct** is used	2. Keyword **union** is used.
3. All members maybe initialized	3. Only its first member location is possible.
4. Different Interpretations of the same memory location are not possible.	4. Different Interpretations of the same memory Locations are possible.
5. Consume memory space compare to union	5. Conservation of memory is possible
6. A structure allocates the total size of all elements in it	6. A union only allocate as much memory as its largest element requires