C Programming

(EG2101CT)

II Yr. / I Part

(DCOM/IT)

By

Arjun Chaudhary

Published in : www.arjun00.com.np
www.facebook.com/arjun00.com.np

Authors © Arjun Chaudhary
Email:- info@arjun00.com.np

Unit 1. Programming Language Fundamentals **[6 Hrs.]**

1.1. Introduction to Program and Programming Language
1.2. Types of Programming Language (Low Level and High-Level Language)
1.3. Language Translator (Assembler, Compiler and Interpreter)
1.4. Program Error, Types of Error (Syntax, Semantic, Runtime Error)
1.5. Program Design Tools (Algorithm, Flowchart)

Unit 2. Introduction to C **[8 Hrs.]**

2.1. Overview and History of C
2.2. Features, Advantages and Disadvantages of C
2.3. Structure of C Program, Compiling Process
2.4. Character set used in C, Data types, Variables. C Tokens (Keywords, Identifier, Constants, Operators), Header files, Library function
2.5. Preprocessor Directives, Escape Sequence, Comments
2.6. Input Output Operation
  2.6.1. Formatted input/output function (printf(), scanf())
  2.6.2. Unformatted input/output function (getchar(), putchar(), gets(), puts(), getc(), putc())

Unit 3. Operators and Expressions **[4 Hrs.]**

3.1. Operators, Operand, Operation, Expression
3.2. Types of Operators (Unary, Binary, Ternary, Arithmetic, Relational, Logical, Assignment, Increment/Decrement, Conditional, Bitwise, Size-of Operators)

Unit 4. Control Structure/Statement **[12 Hrs.]**

4.1. Sequential Statement
4.2. Decision/Selection/Conditional Statement
  4.2.1. if statement
  4.2.2. if…else statement
  4.2.3. if…else if…else statement
  4.2.4. Nested if…else statement
  4.2.5. Switch statement
4.3. Loop (for, while and do-while)
4.4. Jump statement (break, continue, goto statement)

Unit 5. Array and String **[8 Hrs.]**

5.1. Introduction to Array, Declaration, Initialization
5.2. Types of Arrays (1-D Array, Multi-dimensional Array)
5.3. String, Array of String
5.4. String Handling Function (strlen(), strrev(), strupr(), strlwr(), strcpy(), strcat(), strcmp())

Unit 6. Function **[6 Hrs.]**

6.1. Introduction
6.2. Function components (function declaration, function call, function definition)
6.3. Types of function (library/built-in function and user-defined function)
6.4. Category of function according to return value and arguments
6.5. Parameter passing in C (call by value and call by reference)
6.6. Recursion (recursive function)
6.7. Passing array to function
6.8. Passing string to function

Unit 7. Structure and Union **[6 Hrs.]**

7.1. Structure: definition, declaration, initialization, size of structure
7.2. Accessing member of Structure
7.3. Array of Structure
7.4. Nested Structure
7.5. Union: definition, declaration, size of union
7.6. Structure Vs. Union

Unit 8. Pointer **[4 Hrs.]**

8.1. Introduction to Pointer
8.2. Address (&) and indirection (*) operator
8.3. Pointer Arithmetic Operations
8.4. Pointer to Pointer in C
8.5. Dynamic Memory Allocation (malloc(), calloc(), realloc(), free())

Unit 9. Data files **[6 Hrs.]**

9.1. Introduction to data files
9.2. Types of files (text file, binary file)
9.3. File handling operation
9.4. Opening and closing file
9.5. Creating file
9.6. Library functions for READING from a file and WRITING to a file: (fputs, fgets, fputc, fgetc fprintf, fscanf)

Final written exam evaluation scheme			
Unit	**Title**	**Hours**	**Marks Distribution***
1	Programming Language Fundamentals	6	8
2	Introduction to C	8	11
3	Operators and Expressions	4	5
4	Control Structure/Statement	12	16
5	Array and String	8	11
6	Function	6	8
7	Structure and Union	6	8
8	Pointer	4	5
9	Data files	6	8
	Total	**60**	**80**

UNIT 1

Programming Language Fundamentals

Programming Language

- The set of sequenced instructions which command the computer to perform particular operation or a specific task is called a program.
- The process of writing a program following the grammar of the programming language is called programming.
- The programming language is used to communicate with the computer. It provides the set of symbols, characters, strings, numbers, variable, constantetc. that helps us to make the statement. A statement is used to express the task what we want to perform that's why we need programming language. Itis simply the coded language used by programmers to write instructions thata computer can understand to do what the user wants.

Types of Programming Language

There are two types of programming languages, which can be categorized into the following ways:-

- **Low level language:** This language is the most understandable language used by computer to perform its operation. It can be further categorized into two types:-
  - ✓ **Machine Language (1GL):**- It consists of string of binary numbers (i.e. 0s and 1s) and it is the only onelanguage, the processor directly understands
  - ✓ **Assembly language (2GL):**- It is machine oriented language. It uses mnemonics codes in place of 0s and1s. The program is converted into machine code by assembler.

- **High level language:**
  - ✓ Instructions of this language's is closely resemble to human language or English like words or sentences, problem oriented and mathematical notations to perform the task.
  - ✓ The high level language is easier to learn.
  - ✓ It requires less time to write and is easier to maintain the errors.
  - ✓ The high level language is converted into machine language by one of the two different languages translator programs, interpreter or compiler.
  - ✓ Examples of high level languages are BASIC, FORTRAN, ALGOL, COBOL, PL/1, PASCAL, C etc.
  - ✓ High level language are categorized: Procedural oriented language (3GL), Problem-oriented language (4GL),Natural Language (5GL).

Language Translator

Language translator is a program that translates instructions written in assembly or high level language (source code) into machine language (object code). There are mainly three types of language translator:-

- ✓ **Assembler:** Assembler is a translator that converts code of assembly language into the machine language.
- ✓ **Compiler:** A compiler is a translator which translates high level program (source code) into machine instructions (object code) at once and then it can be immediately executed anytime thereafter. This is the fastest method of translating a program.
- ✓ **Interpreter:** An interpreter is a translator which translates high level language into a machine language, one line at a time and executes line of the program after it has been translated. Since, the interpreter needs to evaluate the program letter by letter, then word byword, and then line by line, its speed is incredibly slow.

Different between Compiler and Interpreter

Compiler	Interpreter
• It translates the whole program into machine code at a time. • It traps the errors after compiling the complete program. • The compiling process is incredibly faster. • Compiler based program is difficultto code and debug. • It creates the object code. • Compiler based programminglanguage are C, C++ etc.	• It translates one line or single statement of a program into machine code at a time. • It traps the errors after translating a line of the program at a time. • The translating process is slower. • Interpreter based program is easy to code and debug. • It does not create object code. • Interpreter based programming language are Basic, Visual Basic etc.

Algorithm:

An algorithm is a step-by-step procedure for solving a given problem. It is utilized to create a solution that computers can understand and execute. A strong understanding of algorithms in C is essential for developing efficient programs and solving complexproblems.

Features of Algorithm:

1. Proper understanding of the problem
2. Use of procedures/functions to emphasize modularity
3. Choice of variable names
4. Documentation of the program

Example :-

Algorithm to computer the average of three numbers.

1. Start
2. Read Two number a and b
3. Calculate the sum of a and b and store it in sum
4. Display the value of sum
5. Stop.

Flowchart:

Flowchart is the graphical representation of an algorithm using the standard geometric symbols. It includes a set of various standard shaped boxes that are interconnected by flow lines.

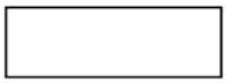
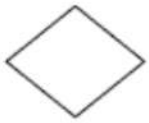
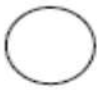
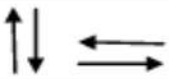
Advantages of flowchart:-

1. **Communication:** Quickly provide logic, ideas and descriptions of algorithms.
2. **Effective analysis:** Clear overview of the entire problem.
3. **Proper documentation:** Documents the steps followed in an algorithm. Ithelps us understand its logic in future.
4. **Efficient coding:** More ease with comprehensive flowchart as a guide.
5. **Easy in debugging and maintenance:** debugging and maintenance of program is easy.

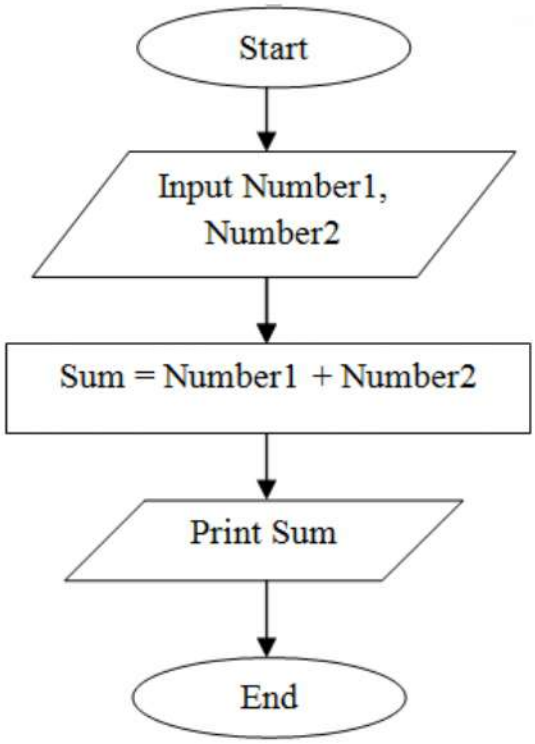
Disadvantages of flowchart:-

1. **Time-consuming:** Designing a flowchart can be a time-consuming process.
2. **Complex:** It can be difficult to draw a flowchart for large and complex programs.
3. **Difficult to modify:** It can be very difficult to modify an existing flowchart.
4. **Difficult to reproduce:** Flowcharts can be difficult to reproduce.
5. **Costly:** Some developers think that making a flowchart is costly.
6. **Big tasks:** Some big tasks can be very difficult for the user to complete.

Symbol of Flowchart

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

Example :- **Flowchart to calculate sum of two number**



Different between Syntax and Semantics

S.No.	Syntax	Semantics
1.	Syntax is the writing format or the grammatical rule of the program.	Semantics is the logic or idea of the program.
2.	Syntax defines the set of rules for writing program statement.	Semantics is the meaning attached to individual statement.
3.	Reference material like help file, books, documents are available for the syntax.	Reference material like help file, books, documents are not available for the semantics.
4.	If a program contains syntax error, it not translated into object code. It is not executed and will not produce output.	If a program contains only semantics error, it is translated into object code. it is executed and will produce output. However, output will not be accurate.
5.	It is simple, faster and cheaper to debug syntax error.	It is complex, time consuming and expensive to debug semantics error.

UNIT 2

Introduction to C

Overview and History of C

- C programming language was created in the early 1972s by Dennis Ritchie at Bell Labs of AT&T (American Telephone & Telegraph), located in the U.S.A.
- Initially, C language was developed to be used in UNIX operating system. It inherits many features of previous languages such as B and BCPL (Basic Combined Programming Language).
- Ritchie created C as a systems programming language for writing operating systems, compilers, and other low-level software.
- C was intended to be a portable and efficient language, capable of generating machine code that was close to the performance of assembly language, but with a more expressive and easier-to-use syntax.
- C was further refined and improved, and in 1978, "The C Programming Language," written by Brian Kernighan and Dennis Ritchie, was published.
- This book became the standard reference for C programmers and helped popularize the language.
- C has become one of the most widely used programming languages in the world, used to develop
- many important software systems.
- C is popular for writing high-performance code in fields such as graphics, video games, and scientific computing.
- C has influenced the design of many other programming languages, including C++, Java, and Python.

Features of C:

The C programming languages has the following features:-

1. It has small size.
2. It has extensive use of function call.
3. It is a strong structural language having powerful data definition methods.
4. It has low level (bit wise) programming available.
5. It can handle low level activities.
6. Pointer makes it very strong for memory manipulations.
7. It has high level constructors.
8. It can produce efficient programs.
9. It can be compiled on variety of computers.

Advantages and Disadvantages of C:-

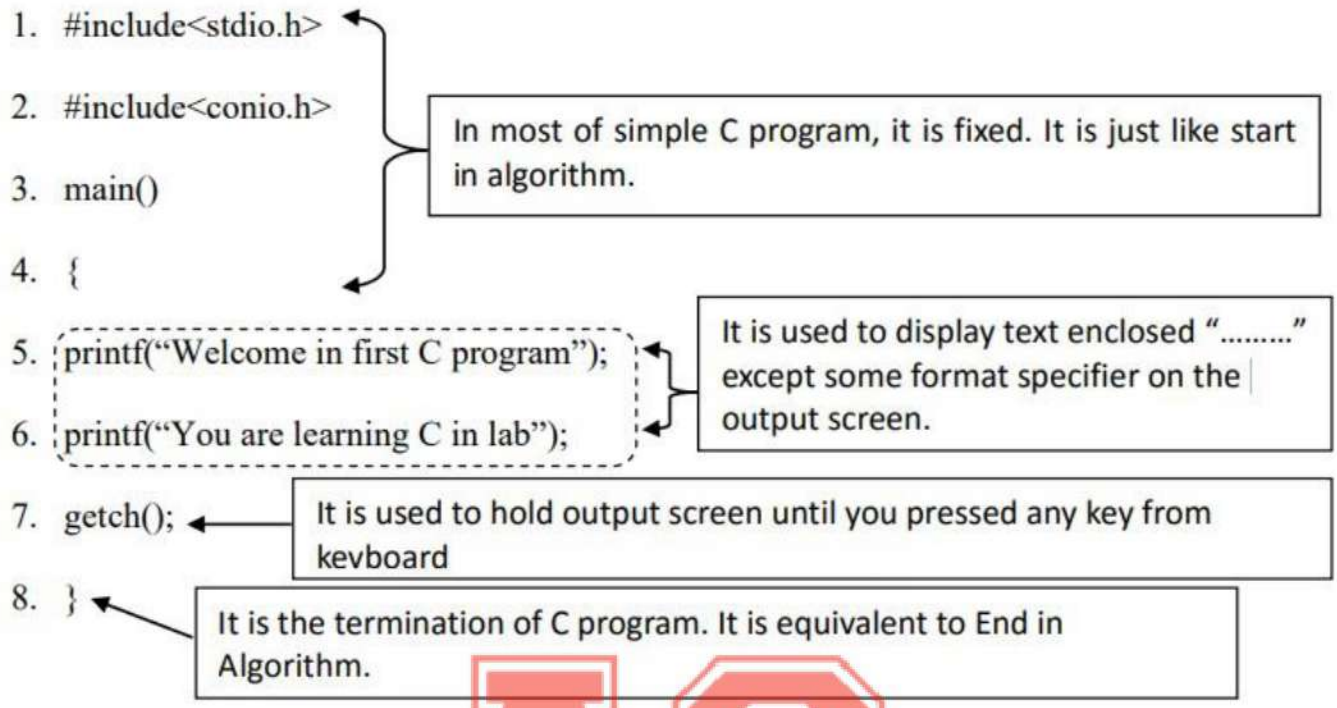
Advantages:

i. It is machine independent programming language.
ii. It is easy to learn and implement C language.
iii. It an be implemented from mobile device to mainframe computers.
iv. It is the mother of all modern programming languages, Python is fully written in C.

Disadvantages:

i. There is no runtime checking.
ii. It has poor error detection system.
iii. There is no strict type checking int data type to float variables.
iv. It does not support modern programming methodologies oriented.

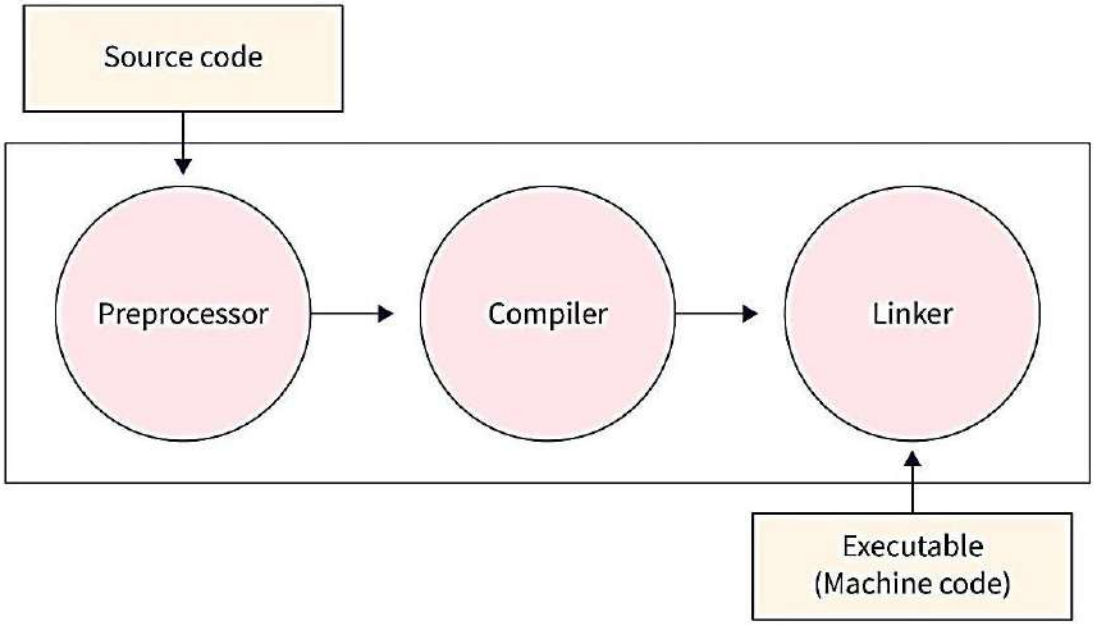
Structure of C program



Compiling process of C

The compiling process of C involves turning human-readable C code into machine-readable instructions that computers can execute. Here's a breakdown:-

- ✓ **Writing Code:** First, you write your program using a text editor or an integrated development environment (IDE).
- ✓ **Preprocessing:**- The preprocessor (part of the compiler) takes care of directives like **#include** and **#define**. It essentially includes header files and expands macros, preparing the code for compilation.
- ✓ **Compilation:** The compiler translates your C code into assembly language or machine code, which the computer can understand. It checks for syntax errors and other issues. If there are no errors, it produces an object file (.o).
- ✓ **Linking:** If your program uses functions from other files or libraries, the linker combines your object file with the necessary libraries to create an executable file. It resolves references to functions and variables.
- ✓ **Executable:** Finally, you get an executable file that contains machine code instructions. You can run this file to execute your program.



Variables in C

A variable is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

Rules for defining variables

- ✓ A variable can have alphabets, digits, and underscore.
- ✓ A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- ✓ No whitespace is allowed within the variable name.
- ✓ A variable name must not be any reserved word or keyword, e.g. int, float, etc. the format of the input and output.

Keywords in C

- A keyword is a reserved word that cannot be use it as a variable name, constant name, etc.
- There are only 32 keywords in the C language.
- Example: int, char, float, signed, void, if, do, else, struct, break, Switch, for etc.

Format specifier

- The Format specifier is a string used in the formatted input and outputfunctions.
- The format string determines
- The format string always starts with % character.

Identifiers

Identifier is name given to entities such as variables, functions, structures etc. For example, int a; float marks;

Here, a and marks are identifiers

Rules for Identifiers:

1. The first character of an identifier should be either an alphabet or an underscore
2. It should not begin with any numerical digit.
3. Commas or blank spaces cannot be specified within an identifier.
4. The length of the identifiers should not be more than 31 characters.
5. Keywords cannot be represented as an identifier.

Data Types in C

It is indicate what type of value store in variable.

-int (%d):- it accept numeric value without decimal point. 2,10,15….

-float (%f):- it accept numeric value with decimal point. 2.5, 10.6, 15.7,….

-char or string(%c or %s):- it accept character or alphanumeric values. 'a', '5a, "name", "hello"…..

%c:- for Single character.

%s:-for group of character (string)

Data Type	Size (in bytes)	Range	Format Specifiers
Char	1	-128 to 127	%c
Int	2	-32768 to 32767	%d
Float	4	3.4 x 10-38 10 3.4 x 1038	%f
Double	8	1.7 x 10-308 to 1.7 x 10308	%lf

OR

- A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

There are 4 types of data types in C, They are:

1. Basic Data types: int, char, float, double
2. Derived data Types: array, pointer, Structure, Union
3. Enumeration data types: enum
4. Void Data type: void

Header file:

A header file is a file with extension h which contains C function declarations and macro definitions to be shared between several source files. Mainly we use stdio.h and conio.h header files. They are:-

1. **stdio.h :-** stdio.h is a standard header file in C that provides functions for input and output operations. These functions include printf() for output and scanf() for input, as well as functions for file input/output operations.
2. **conio.h :-** conio.h is a non-standard header file that is specific to certain compilers, such as Turbo C and Borland C. It provides functions for console input and output operations, such as clrscr() for clearing the console screen and getch() for reading a character from the keyboard.

Preprocessor directives :

Preprocessor directives are lines of source code that are preprocessed by the preprocessor before compilation begins. They are distinguished from other lines of text by the first non-whitespace character, which is the number sign (#). The effect of each preprocessor directive is a change to the text, resulting in a transformation that does not contain the directives or comments.

Escape sequences :

- ✓ Escape sequences in C are special character sequences that begin with a backslash () and are used to represent non-printable or special characters in a string. Example:
- ✓ \n: newline character
- ✓ \t: tab character
- ✓ \b: backspace character

Comments

- ✓ Comments are used to add explanatory or descriptive text to the code, which is ignored by the compiler. There are two types of comments in C:
  - **Single-line comments**:- It begin with two forward slashes (//) and continue until the end of the line.
  - **Multi-line comments**:- It begin with /* and end with */ and can span multiple lines.

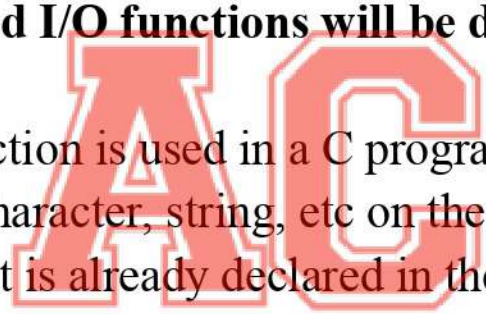
Input Output Operation -

Formatted I/O Functions :- Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user. These types of I/O functions can help to display the output to the user in different formats using the format specifiers. These I/O supports all data types like int, float, char, and many more.

Why they are called formatted I/O?

These functions are called formatted I/O functions because we can use format specifiers in these functions and hence, we can format these functions according to our needs.

The following formatted I/O functions will be discussed in this section:-



- ✓ **printf() :-** printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h (header file).

  Syntax :-

  printf("Format Specifier", var1, var2, …., varn);

- ✓ **scanf() :-** scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is declared in stdio.h(header file), that's why it is also a pre-defined function. In scanf() function we use &(address-of operator) which is used to store the variable value on the memory location of that variable.

  Syntax:-

  scanf("Format Specifier", &var1, &var2, …., &varn);

Unformatted Input/Output functions :- Unformatted I/O functions are used only for character data type or character array/string and cannot be used for any other data type. These functions are used to read single input from the user at the console and it allows to display the value at the console.

Why they are called unformatted I/O?

These functions are called unformatted I/O functions because we cannot use format specifiers in these functions and hence, cannot format these functions according to our needs.

The following unformatted I/O functions will be discussed in this section:-

- ✓ **getch() :-** getch() function reads a single character from the keyboard by the user but doesn't display that character on the console screen and immediately returned without pressing enter key. This function is declared in conio.h (header file). getch() is also used for hold the screen.

  Syntax: -

  getch(); or variable-name = getch();

- ✓ **getche()** :- getche() function reads a single character from the keyboard by the user and displays it on the console screen and immediately returns without pressing the enter key. This function is declared in conio.h (header file).

  Syntax: -

  getche(); or variable_name = getche();

- ✓ **getchar() :-** The getchar() function is used to read only a first single character from the keyboard whether multiple characters is typed by the user and this function reads one character at one time until and unless the enter key is pressed. This function is declared in stdio.h (header file)

  Syntax: -

  Variable-name = getchar();

- ✓ **putchar() :-** The putchar() function is used to display a single character at a time by passing that character directly to it or by passing a variable that has already stored a character. This function is declared in stdio.h (header file)

  Syntax:-

  putchar(variable_name);

- ✓ **gets()** :- gets() function reads a group of characters or strings from the keyboard by the user and these characters get stored in a character array. This function allows us to write space-separated texts or strings. This function is declared in stdio.h(header file).

  Syntax:-

  char str[length of string in number]; //Declare a char type variable of any length gets(str);

- ✓ **puts() :-** In C programming puts() function is used to display a group of characters or strings which is already stored in a character array. This function is declared in stdio.h(header file).

  Syntax:-

  puts(identifier_name);

- ✓ **putch() :-** putch() function is used to display a single character which is given by the user and that character prints at the current cursor location. This function is declared in conio.h(header file)

  Syntax:-

  putch(variable_name);

UNIT 3

Operators and Expressions

Operators:

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions.

Operands:

Operands in C are the values or variables that are used as input to an operator to perform an operation. Operands can be of different data types, depending on the operator being used. For example:

- 1. int x = 5; // x is an operand of the assignment operator
- 2. int y = x + 3; // x and 3 are operands of the addition operator, and y is an operandof the assignment operator

Operations:

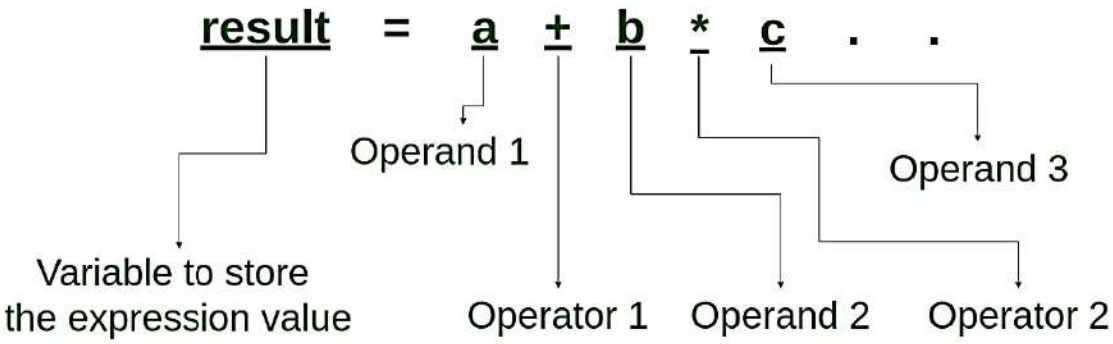
Operations in C are the actions performed by operators on operands. For example:

- 1. int x=5+3;
  the addition operator performs the operation of adding 5 and 3, and theassignment operator assigns the result (8) to x
- 2. int y=x>7?1:0;
  the ternary operator evaluates the condition x > 7, and returns either 1 or 0depending on the result.

Expression:

An expression is a combination of operators, constants and variables. An expression may consist of one or more operands, and zero or more operators to produce a value.

What is an Expression?



C language provides the following **types of operators** :-

Operators in C

	Operators	Type
Unary Operator →	++, --	Unary Operator
Binary Operator	+, -, *, /, %	Arithmetic Operator
	<, <=, >, >=, ==, !=	Rational Operator
	&&, ||, !	Logical Operator
	&, |, <<, >>, ~, ^	Bitwise Operator
	=, +=, -=, *=, /=, %=	Assignment Operator
Ternary Operator →	?:	Ternary or Conditional Operator

1. Arithmetic Operators

Operator	Description	Example
+	Adds two operands.	A+B=30
-	Subtracts second operand from the first.	A-B-10
*	Multiplies both operands.	A*B=200
/	Divides numerator by de-numerator.	B/A-2
%	Modulus Operator and remainder of after an integer division	B%A=0
++	Increment operator increases the integer value by one.	A++=11
--	Decrement operator decreases the integer value by one.	A-- = 9

2. Relational Operator

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

3. Logical Operators

Operator	Description	Example
&&	Called Logical AND operator. If both the operands arenon-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

4. Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. They are &, |, and ^.

5. Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C= A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C+ =A is equivalent to C=C+A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C-=A is equivalent to C=C-A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C* =A is equivalent to C=C*A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C/ =A is equivalent to C=C/A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C% =A is equivalent to C = C% A
<<=	Left shift AND assignment operator.	C<< =2 is same as C = C<<2
>>=	Right shift AND assignment operator.	C>>=2 is same as C = C>> 2
&=	Bitwise AND assignment operator.	C & =2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C^=2 is same as C = C^2
|=	Bitwise inclusive OR and assignment operator.	C|=2 is same as C = C|2

UNIT 4

Control Structure/Statement

The "Control Statements" enables us to specify the order in which the various instructions in a program are to be executed bye the computer. In other words the control instructions determine the "low of control" in a program. They are:

1. Sequence Statement
2. Decision Statement
3. Looping Statement

Sequence Statement :

In a sequential statement program control flows from top to bottom. Program statement are execute one after another in a sequence. It is the most simple type of control structure. It doesn't contain any conditions and looping statement.

Syntax:

```
statement_1;
statement_2;
……..
……..
statement_N;
```

For example: WAP to read two numbers and find the sum of their square.

```
#include <stdio.h>
#include <conio.h>void
main()
{
int a,b, sum=0;
printf("Enter two numbers:");
scanf("%d%d",&a,&b); sum=a*a+b*b;
printf("\n The sum of square of %d and %d is : %d", a,b,sum);
getch();
}
```

Some Problem :-

1. Calculate simple interest: si=(p*t*r)/100;
2. Calculate Area of a parallelogram: A=b*h;
3. Calculate Area of a triangle: A=1/2(b*h);
4. Calculate Area of a circle: $A=\pi r^2$; ->Hint(π=pi=3.142,r^2=r*r)
5. Calculate Volume of a cylinder: $V=\pi r^2 h$;
6. Calculate Volume of a sphere: $4/3\pi r^3$;
7. WAP to convert a temperature given in Celsius. (f=c*9/5+32;)
8. WAP to find price of n mango given the price of a dozen mango.
9. WAP to convert number of days into days and month.
10. WAP that read time in seconds and converts it in to hour, minute, and second.

#WAP to convert a temperature given in Celsius toFahrenheit?

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

void main()
{
float c,f;clrscr();
printf("Enter temperature in Celsius:");
scanf("%f",&c);f=c*9/5+32;
printf("Temperature in Fahrenheit:%f",f);
getch();
}
```

#WAP to find price of n mango given the price of adozen mango.

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

void main()
{
int pd,pn,n;clrscr();
printf("Enter price of a dozen mango:");
scanf("%d",&pd);
printf("Enter number of mango:");
scanf(("%d",&n);
pn=pd/12*n;
printf("Price of total Mango:%d",pn );
getch();
}
```

#WAP to convert number of days into days and month.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int months,days;
clrscr();
printf("Enter days:");
scanf("%d",&days);
months=days/30; days=days%30;
Printf("Months=%d Days=%d",months,days);
getch();
}
```

#WAP that read time in seconds and converts it into hour, minute, and second.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int sec,min,hr,rsec;
clrscr();
printf("Enter your time in seconds:");
scanf("%d",%sec);hr=sec/3600;
rsec=sec%3600; min=rsec/60;
sec=rsec%60;
printf("\n %d Hour, %d Minutes, %d Seconds",hr,min,sec);
getch();
}
```

Decision Statement :

In decision control structure, different sections of code are executed depending on a specific condition or the value of a variable. This allows a program to take different courses of action depending on different conditions. A decision causes a one-time jump to a different part of the program, depending on the value of an expression.

There are various decision control structures:-

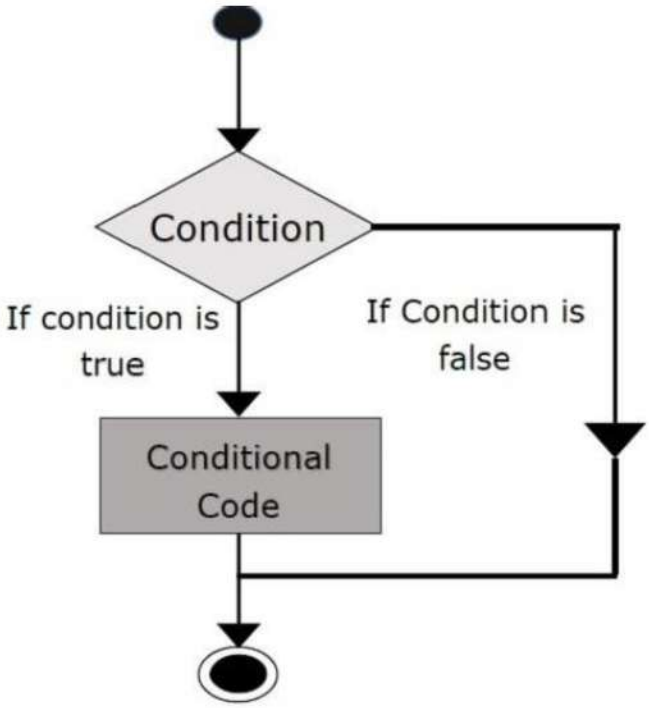
1. If Statement
2. If…else Statement
3. If…else…if…else statement
4. Switch-case statement

If statement :

It is used to execute an instruction or block of instructions only if a condition is fulfilled. If condition is true, then statement is executed otherwise the statement is ignored and the program continues on the next instruction.

Syntax:

```
If(condition)
{
 statement;
}
```



Example ;-

WAP to check whether entered number is negative:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n;
clrscr();
printf("Enter a number to be tested:");
scanf("%d",&n)
;if(n<0)
{
printf("\n The number is Negative",n);
}
getch();
}
```

#WAP to demonstrate the use of if statement.

```
#include<stdio.h>
#include<conio.
h> void main()
{
int x,y;
clrscr();
printf("Inpute an integer value for x & y:");
scanf("%d%d",&x
,&y);if(x==y)
    printf("x is equal to  y\n");
if(x>y)
   printf("x is greater than y\n");
if(x<y)
  printf("x is smaller than y\n");
getch();
}
```

#WAP that checks whether the number entered by user is exactly divisible by 5 but not by 11.

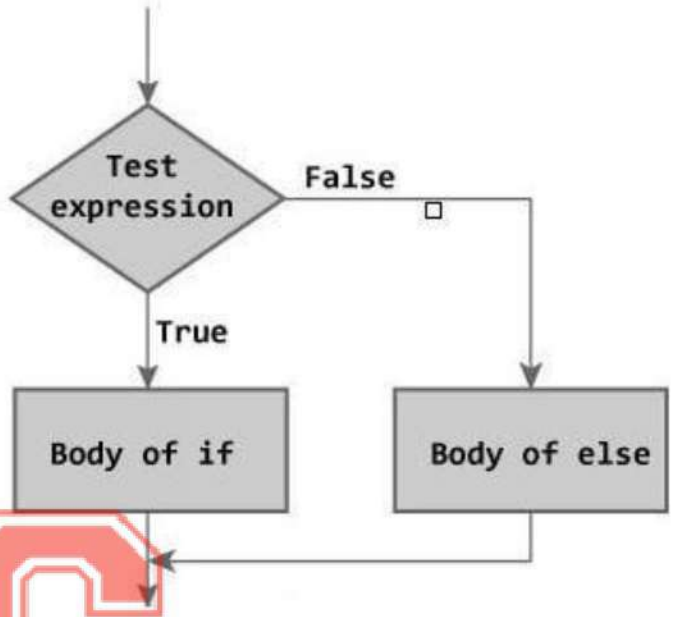
```
#include<stdio.h>
#include<conio.h>void main()
{
int n; clrscr();
printf("Enter a number");
scan("%d",&n); if(n%5==0 &&
n%11!=0)
{
printf("%d is divisible by 5 but not by
11",n);
}
getch();
}
```

If…Else Statement :

If…else: If condition is true then if portion statements are evaluated otherwise else part of the statement is evaluated. There are only two choices provided in the program statement.

Syntax:

```
If (condition)
{
block of statement;
}
Else
{
block of statement;
}
```



Example :

WAP to check whether entered number is negative or non-negative:

```
#include<stdio.h>
#include<conio.h> void
main()
{
int n; clrscr();
printf("Enter a number to be tested:");
scanf("%d",&n);
if(n<0)
{
printf("\n The number is Negative",n);
}
else
{
printf("\nThe number is Non-Negative",n);
}
getch();
}
```

#WAP to find whether a given no . Even or Odd

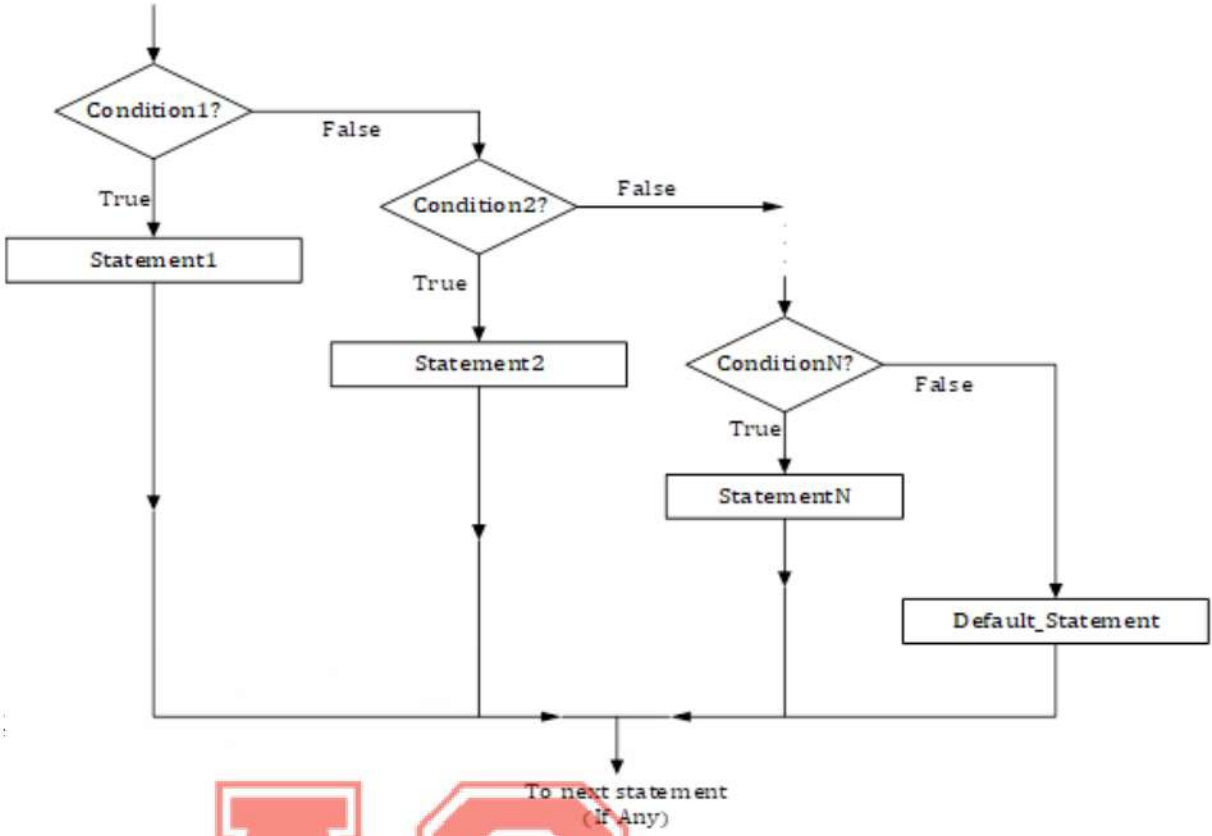
```
#include <stdio.h>
#include<conio.h>
void main() {
  int num;
printf("Enter an integer: ");
  scanf("%d",
&num); if (num %
2 == 0) {
    printf("Number is even.\t", num);
  } else {
    printf("Number is odd.\t", num);
  }
  getch();
}
```

if…else…if….else :

It is useful only when the number of choices is larger than two then this type of if…else…if…else control structure is used. It is a multi-way decision based conditional structure. This works by cascading of several comparisons. As soon as one of the conditions is true, the statement or block of statements following them is executed and no further condition is checked.

Syntax:

```
If (condition1)
  {
block of statement1;
  }
else if(condition2)
  {
block of statement2 ;
  }
  ……
  ……
else
 {
block of statement N ;
  }
```



- WAP to read 5 subject marks and calculate percentage as follows if passed.

1. Percentage >=75, Distinction with percentage.
2. Percentage >=60 and <75 First division with percentage.
3. Percentage >=45 and <60 second division with percentage.
4. Percentage >35 and <45 Pass division with percentage.
5. Otherwise Fail

- WAP to find greatest in three number.

- WAP to find the commission amount on the basis of sales amount as per thefollowing condition:

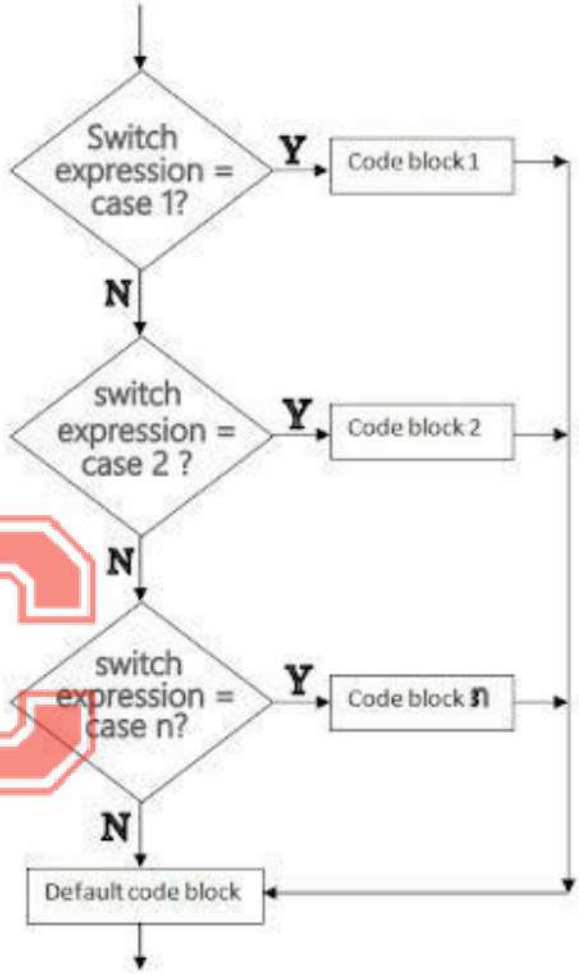
Sales amount(Rs)	Commission
0-1000	5%
1001-2000	10%
>2000	

Switch Case :

It is used to check several possible constant values for an expression. The switch statement can be used instead of multiple if…else conditional statements. It is mainly used to generate menu based programs.

Syntax:

```
switch  (expression1)
{
case 1:
statements …….
break ;case 2:
statements ……....
break ;
case N:
statements …….
break ; default:
default statement
}
```



Loop:

It is also called repetitive or iterative control structure. It is used for execution of the same section of code for more than once. A loop is a way of repeating a statement a number of times. The main advantage ofloop is that it makes the program very simple and short to produce a significantly greater result simply by repetition. C provides three looping control structure:-

1. For Loop
2. While Loop
3. Do While Loop

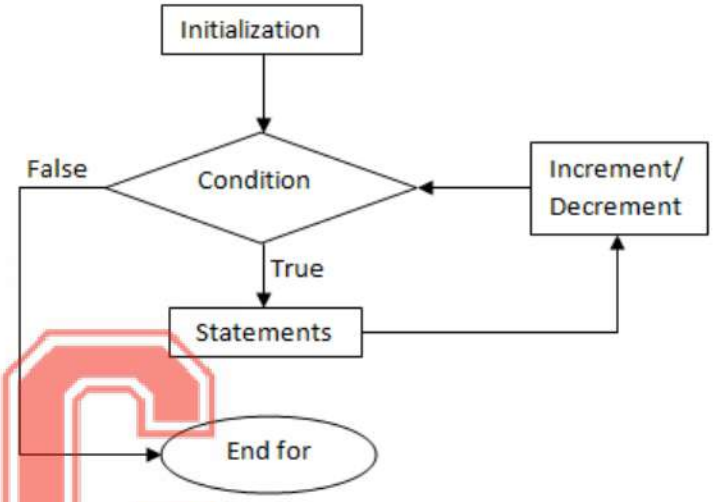
For Loop :

A for loop is a repetition control structure which allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

In for loop, a loop variable is used to control the loop. First initialize this loop variable to some value, then check whether this variable is less than or greater than counter value. If statement is true, then loop body is executed and loop variable gets updated.

Syntax :

```
for(initial; expression; inc/dec)
{
 Statement;
}
```



Example:-

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

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

Some Problem :

1. WAP to print C Programming 10 times.
2. WAP to input a number from user and check whether it is prime or not.
3. WAP to print Fibonacci series i. e 0 1 1 2 3 5 8 ……
4. WAP to print Fibonacci Series up to 100.
5. WAP to find Factorial of a given positive number.
6. WAP to print the 10 positive integers and their factorial.
7. WAP to find sum of the cubes of first ten numbers.
8. WAP to program to find the sum of first 50 natural numbers.
9. WAP input a number and display it's multiplication table.
10. WAP to print 10 terms of the following series 1 5 9 13 …

#WAP to print C Programming 10 times.

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int I; for(i=1;i<=10;i++)
{
printf("C Programming");
}
getch();
}
```

#WAP to input a number from user and check whether it is prime or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,num,r;
clrscr();
printf("enter a number:");
scanf("%d",&num); for(i=2; i<num;
i++)
{
m=num%i;
if(m==0) break;
}
if(m==0)
 printf("a number is not prime");
else
printf("a number is prime");
getch();
}
```

#WAP to print Fibonacci series i. e 0 1 1 2 3 5 8 ……

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=0,b=1,c=0,i;
printf("%d%d",a,b);
for(i=1;i<=10;i++)
{
c=a+b; printf("%d\t",c);
a=b;
b=c;
}
getch();
}
```

#WAP to print Fibonacci Series up to 100.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=1,b=1,c=0,i;
printf("%d%d",a,b);for(i=0;i<10;i++)
{
c=a+b;
if(c<100)
}
printf("%d\t",c);
}
a=b;
b=c;
}
getch();
}
```

#WAP to find Factorial of a given positive number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,n,fact=1;clrscr();
printf("enter a number");
scanf(" %d",&n);if(n<0)
{
printf("factorial of negative number is not
possible");
}
else
{
for(i=n;i>0;i--)
{
fact=fact*i;
}
printf("factorial=%d",fact);
}
getch();
}
```

#WAP to print the 10 positive integers and their factorial.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,n,fact=1;clrscr();
printf("enter a number");
scanf(" %d",&n);
for(i=1;i<=n;i++)
{
printf("\n number=%d",i);
fact=fact*i;
}
printf("factorial=%d",fact);
getch();
}
```

#WAP to find sum of the cubes of first ten numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,c,sum=0; clrscr();
for(i=1;i<=10;i++)
{
c=i*i*i;
sum=sum+c;
}
printf("sum of cube=%d",sum);
getch();
}
```

#WAP to program to find the sum of first 50 natural numbers.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,sum=0;
for(i=1;i<=50;i++)
{
sum=sum+i;
}
printf("sum=%d",sum);
return 0;
}
```

#WAP input a number and display it's multiplication table.

```
#include<stdio.h>
#include<conio.h>
int main() {
  int n, i;
  printf("Enter an integer: ");
  scanf("%d", &n);
  for (i = 1; i <= 10; i++)
  {
    printf("%d * %d = %d \n", n, i,
    n * i);
  }
  return 0;
}
```

#WAP to print 10 terms of the following series 1 5 9 13 ……

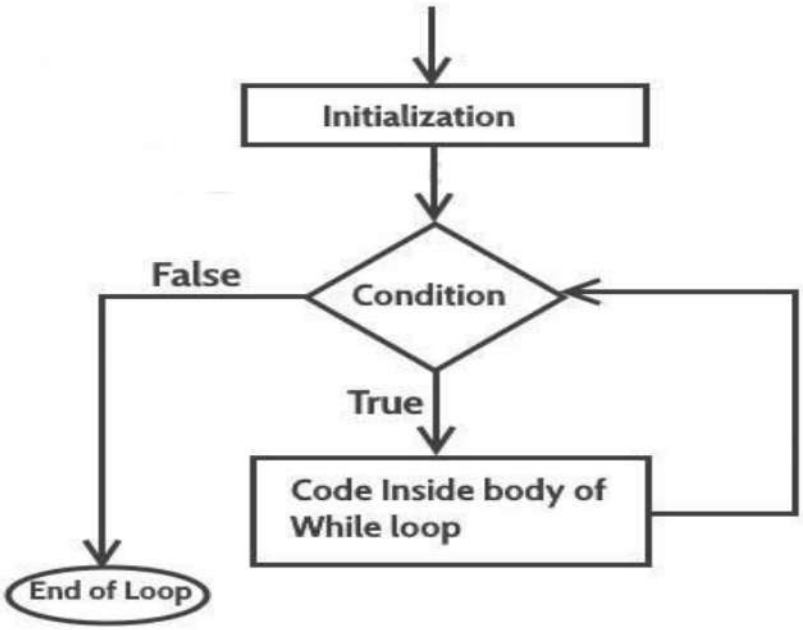
```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,n=1; for(i=1;i<=10;i++)
{
printf("%d",n);n+=4;
}
getch();
}
```

While Loop :

While loop is pre-test loop, It first test a specified condition expression and as long as the conditional expression is true, then execute the statement in body. The increment/decrement performed inside the body of the loop. It is also called entry control loop.

Syntax:

```
initialization ; while(termination
condition)
{
statements ;
…… increment/decrement;
}
```

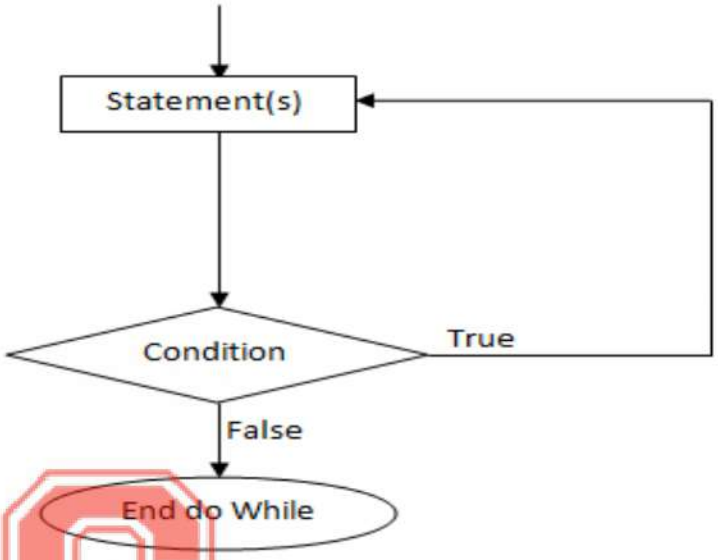


Do While Loop :

This loop is an exit control loop. It runs at least once even though the termination condition is set to false. This loop test the condition at exit point hence it is called exit control loop.

Syntax:

```
Initialization;do
  {
   statements..... ;
  increment/decrement  ;
  }
   while(termination condition);
```



Difference between for and while loop:

for loop	while loop
1.It is definite loop	1.It may be definite and may not be definite.
2.The loop expression is within a one block.	2.The loop expression is scattered throughout the program.
3.It used the keyword for.	3.It use the keyword while.
4.It contains three expressions.	4.It contain only one expression.

5.Syntax:
for(init; condition; inc/dec)
{
statements;
}
	5.Syntax:
	initial value;
	while(condition)
	{
	statements;
	increment/decrement;
	}

Difference between while and do while loop:

while loop	do while loop
1.In the while loop test condition is evaluated before the loop is executed.	1.In do-while loop the test condition is evaluated after the loop is executed.
2.It is an entry controlled loop.	2. It is an exit controlled loop.
3.It has a keyword while.	3. It has two keyword do and while.
4.Loop is not terminated with semicolon.	4. Loop is terminated with a semicolon.
5.It doesn't execute the body of loop until and unless the condition is true.	5. Body of the loop will always executed at least once since the test condition is not checked until the end of the loop.

6.Syntax:
initial value;
while(condition)
{
statements;
increment/decrement;
}
	6. Syntax:
	initial value;
	do
	{
	statements;
	increment/decrement;
	}while(condition);

Jump statement :

Jump statement is used to alter the normal flow of execution of a program. It allows the program to transfer control to a different part of the code, such as a different function or a different block of code within the same function.

There are **three types of jump statements** in C :- **goto, break,** and **continue**.

- **Goto statement :-** The goto statement is utilized to transfer control to a labeled statement in the same function. It is often considered bad practice to use goto statements as they can make the code difficult to read and understand. However, there are some events where they may be useful, such as when implementing error handling.

Syntax:

```
goto label;
.
.
.
label: statement;
```

Example:

```
#include <stdio.h>
int main() {
  int i = 1;
  loop:
printf("%d ", i);
i++;

  if (i<= 10) {
goto loop;
  }

  return 0;
}
```

- **Break statement:-** The break statement is utilized to exit a loop or switch statement before its normal termination. It is commonly utilized in loops to exit early if any specific condition is met.

Syntax:

```
while (condition) {
  if (condition) {
    break;
  }
}
```

Example:

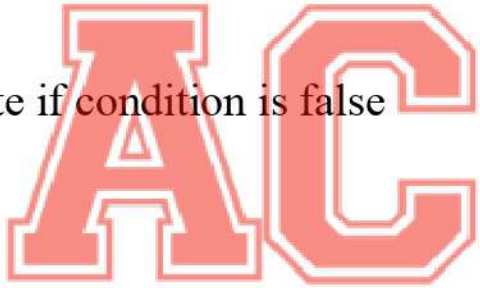
```
#include <stdio.h>

int main() {
  int i;

  for (i = 1; i<= 10; i++) {
    if (i == 5) {
      break;
    }
```

```
    printf("%d ", i);
    }

    return 0;
}
```

- **Continue statement:-** The continue statement is utilized to skip the remaining code in a loop for the current iteration and move on to the next iteration. It is commonly used in loops to skip over certain elements.

Syntax:

```
for (int i = 0; i< n; i++) {
  if (condition) {
    continue;
  }

  // Code to execute if condition is false
}
```



Example:

```
#include <stdio.h>
int main() {
  int i;

  for (i = 1; i<= 10; i++) {
    if (i % 2 == 0) {
      continue;
    }
  printf("%d ", i);
  }

  return 0;
}
```

UNIT 5

Array and String

Introduction to Array :

An array is a collection of elements of the same data type stored at contiguous memory locations. It provides a way to store multiple elements of the same typeunder a single name. Each element in the array can be accessed using an index.

You can use **array subscript** (or index) to access any element stored in array. Subscript starts with 0, which means arr[0] represents the first element in the array arr. In general arr[n-1] can be used to access nth element of an array. where n is any integer number

Declaration of array:

data_type array_name[size];

Array Declaration :

In C, we have to declare the array like any other variable before using it. We can declare an array by specifying its name, the type of its elements, and the size of its dimensions. When we declare an array in C, the compiler allocates the memory block of the specified size to the array name.

Syntax :-

data_type array_name [size];

or

data_type array_name [size1] [size2]...[sizeN;

Array Initialization :

Initialization in C is the process to assign some initial value to the variable. When the array is declared or allocated memory, the elements of the array contain some garbage value. So, we need to initialize the array to some meaningful value. There are multiple ways in which we can initialize an array in C :-

1. Array Initialization with Declaration
2. Array Initialization with Declaration without Size
3. Array Initialization after Declaration (Using Loops)

Types of Arrays :-

1. One Dimensional Array:

A list of item can be given one variable name using one subscript (or index or dimension) and such a variable is called one dimensional array.

Declaration of 1-D Array: data_type array_name[size];

2. Multidimensional Array or 2-D Array:

Multidimensional array are those having more than one dimension are such a variable are called Multidimension or 2-D Array.

Declaration of multidimensional Array: data_type

array_name[dim1][dim2]…[dimN];

Initialization of Array: storage_class data_type array_name[size]={value1, value2,…, valueN};

#WAP to read 10 integers, store it in array anddisplay them.

```
#include<stdio.h>
#include<conio.h>
main()
{
int p, a[10];
printf("\n Enter 10 numbers:");
for(p=0;p<10;p++)
{
scanf("%d",&a[p]);
}
printf("\n The numbers are:");
for (p=0;p<10;p++)
{
printf("%d\t",a[p]);
}
getch();
}
```

#WAP to input the n numbers and sort them in descendingorder.

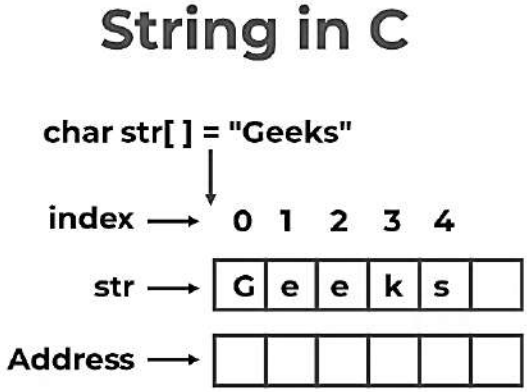
```
 #include<stdio.h>
#include<conio.h>
main()
{
  int nums[100], i, j, n, temp;
 printf("\nHow many numbers you want to
sort?:\t");
  scanf("%d", &n);
  for(i=0;i<n;i++)
  scanf("%d", &nums[i]);
  for(i=0;i<n-1;i++)
   {
    for(j=i+1;j<n;j++)
    {
if(nums[i]>nums[j])
    {
    temp=nums[i];
    nums[i]=nums[j];
    nums[j]=temp;
    }
    }
    }
    printf("\nThe numbers in
    ascending order are:\n");
    for(i=0;i<n;i++)
    printf("\t%d", nums[i]);
    getch();
    }
```

WAP to read a matrix, store it in array and display it.

```
#include<stdio.h>
#include<conio.h>
main()
{
int p,q,matrix[3][4];
printf("\n Enter a matrix of 3x4:\n\n");
for(p=0;p<3;p++)
{
for(q=0;q<4;q++)
{
scanf("%d",&matrix[p][q]);
}
}
printf("\n The elements of matrix
    are:\n\n");
for(p=0;p<3;p++)
{
for(q=0;q<4;q++)
{
printf("%d\t",matrix[p][q]);
}
printf("\n");
}
getch();
}
```

Strings

A String in C programming is a sequence of characters terminated with a null character '\0'. The C String is stored as an array of characters. The difference between a character array and a C string is that the string in C is terminated with a unique character '\0'.



String handling functions :

String handling functions are used to manipulate strings and textual data in C programming. They are defined in the header file **string.h**.

Here are some commonly used string handling functions:-

- ✓ strcpy(): Copies the value of one string to another
- ✓ strncpy(): Copies the first five characters of one string to another
- ✓ strlen(): Returns the total number of characters in one string
- ✓ strcat(): Appends one string to another
- ✓ strcmp(): Returns 0 if two strings are the same, a negative value if the first string is less than the second, and a positive value if the first string is greater than the second
- ✓ strncmp(): Compares the first four characters of two strings
- ✓ strcmpi(): Compares two strings, string1 and string2, by ignoring case
- ✓ strncat(): Concatenates a specified number of characters from the second string to the first

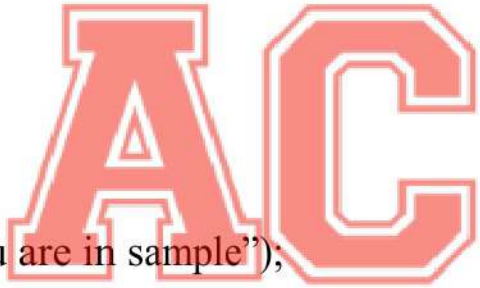
UNIT 6

Function

Introduction to Function

A function is a block of code that performs a specific task. Functions provide modularity, reusability, and maintainability to the code. They allow you to break down a program into smaller, manageable pieces, each responsible for a specific functionality.

```
#include <stdio.h>
#include<conio.h>
main ()
{
void sample( );
printf("\n You are in main");
getch();
}

void sample( )
{
printf ("\n You are in sample");
}
```



Function declaration:-

Function declaration is also known as function prototype. It inform the compiler about three thing, those are name of the function, number and type of argument received by the function and the type of value returned by the function.

While declaring the name of the argument is optional and the function prototype always terminated by the semicolon.

Function definition:

The collection of program statements that describes the specific task to be done by the function is called a function definition.

```
#include <stdio.h>
#include<conio.h>
int add(int c, int d)
{
int sum; sum = c+d;return sum;
}
main()
{
……;
}
```

Function Call :

A function can be called by specifying the function's name, followed by a list of arguments enclosed in parentheses and separated by commas.

For example, the function add() can be called with two arguments from main() function as: add (a, b);

```
#include <stdio.h>
#include <conio.h>
int add(int, int);
main()
{
int a=50,b=100, x;
x=add(a, b);
printf("%d", x);
getch();
}
int add(int c, int d)
{
int sum; sum = c+d;
return sum;
}
```

Types of function:

- Library Function (Built in function)
- User-defined Functions

Library function:

There functions are already defined in the C compilers. They are used for String handling, I/O operations, etc.

These functions are defined in the header file.

To use these functions we need to import the specific header file <stdio.h>, <conio,h>, <math.h>.

User-defined Function:

User-defined functions are the ones created by the user. It is like customizing the functions that we need in a program.

There are four types of user-define function:

- Function with no arguments but no return values.
- Functions with arguments but no return value.
- Function with arguments but return values.

Function with no arguments but no return values.

- ✓ When a function has no arguments, it does not receive any data from the calling function.
- ✓ When a function does not return a value, the calling function does not receive any data from the called function

Syntax:

```
void function_name()
{
/* body of function */
}
```

```
#include <stdio.h>
#include<conio.h>
void
add();
main()
{
add();
getch();
}
void add();
{
int a,b,sum;
printf("\nEnter two
numbers:"); scanf("%d
%d",&a,&b); sum=a+b;
printf("\nThe sum is:%d",sum);
}
```

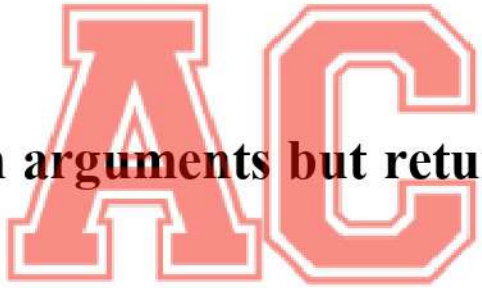
Function with arguments but no return values.

- This type of function has arguments and receives the data from the calling function.
- But after the function completes its task, it does not return any values to the calling function.

Syntax:

```
void function_name(argument_list)
{
/* body of function */
```

```
#include <stdio.h>
#include <conio.h>
void add(int, int)
main()
{
int a,b;
printf("\n Enter two numbers:");
scanf("%d %d",&a,&b);
add(a, b);
getch();
}
void add(int a, int b)
{
int sum;
sum=a + b;
printf("\n The sum is:%d", sum);
}
```



Function with arguments but return values.

- This type of function has arguments and receives the data from the calling function.
- However, after the task of the function is complete, it returns the result to the calling function via return statement.

Syntax:

```
return_type  function_name(argument_list)
{
/* body of function */
}
```

```
#include <stdio.h>
#include <conio.h>
int add(int, int);
main()
{
int a, b, x;
printf("\n Enter two
numbers:");
scanf("%d %d", &a, &b);
x=add(a, b);
printf("\n The sum is:%d", x);
getch();
}
int add(int a, int b)
{
int sum;
sum=a + b;
return sum;
}
```

Some Problem

Write functions to add, subtract, multiply and divide two numbers a and b.

Call by value and call by reference

- ✓ There are two way through which we can pass the arguments to the function such as **call** by value and **call by reference**.

1. Call by value

In the call by value copy of the actual argument is passed to the formal argument and the operation is done on formal argument.

When the function is called by 'call by value' method, it doesn't affect content of the actual argument.

Changes made to formal argument are local to block of called function so when the control back to calling function the changes made is vanish.

Example:-

```
  main()
  {
 int x,y;
 change(int,int);
printf("enter two values:\n");
scanf("%d%d",&x,&y);
 change(x ,y);
printf("value of x=%d and y=%d\n",x ,y);
 }
change(int a,int b);
{
int k;
k=a;
a=b;
b=k;
 }
```

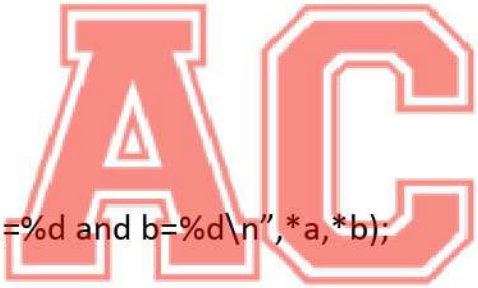
2. Call by reference

Instead of passing the value of variable, address or reference is passed and the function operate on address of the variable rather than value.

Here formal argument is alter to the actual argument, it means formal arguments calls the actual arguments.

Example:-

```
void main()
{
int a,b;
change(int *,int*);
printf("enter two values:\n");
scanf("%d%d",&a,&b);
change(&a,&b);
printf("after changing two value of a=%d and b=%d\n:"a,b);
}
change(int *a, int *b)
 {
int k;
 k=*a;
*a=*b;
 *b= k;
printf("value in this function a=%d and b=%d\n",*a,*b);
}
```



Recursive function :-

A recursive function is a function that calls itself directly or indirectly to solve a problem. It's a technique used in programming to solve problems by breaking them down into smaller, similar sub problems.

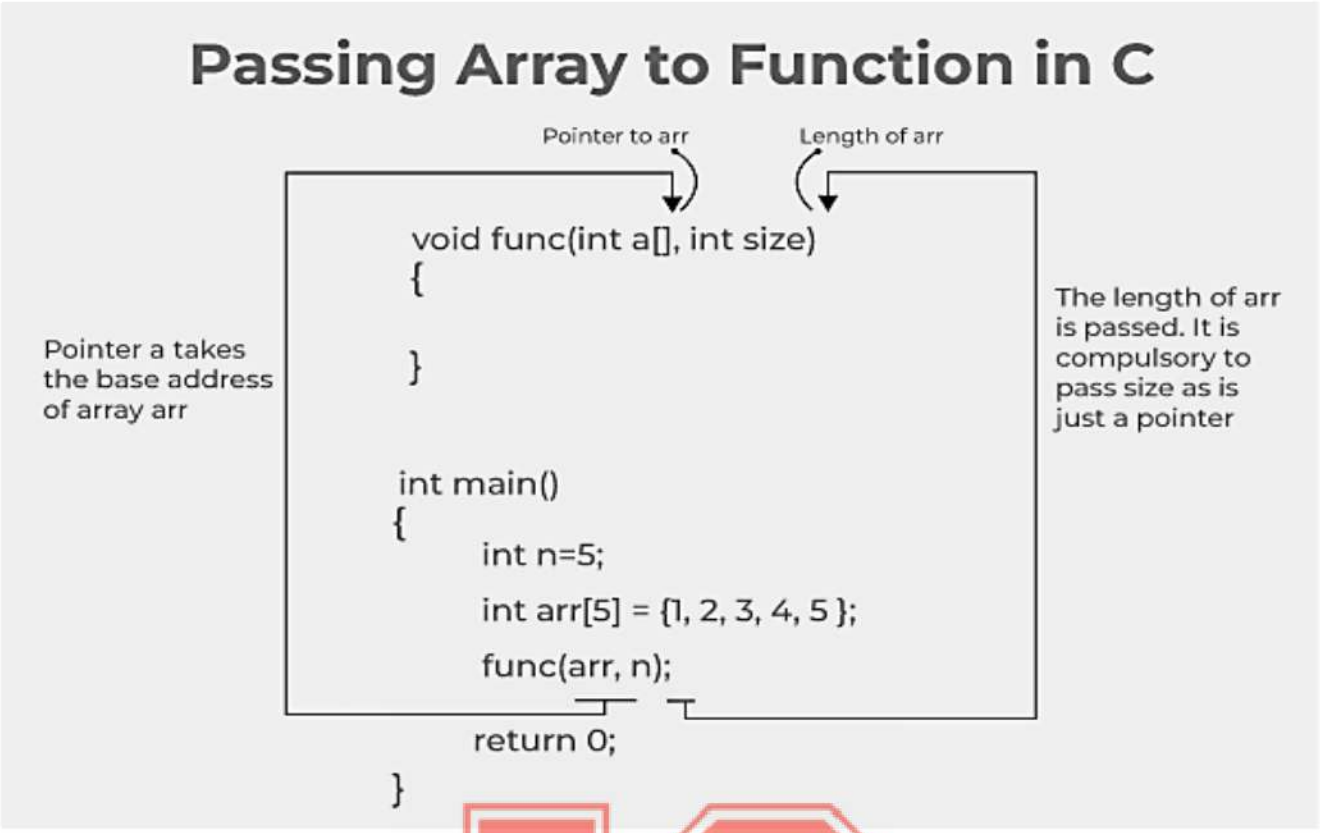
Examples of recursive functions:

Fibonacci series: The Fibonacci series is a number series created by adding the two preceding numbers in the series.

Series: The series a1=10, a2=2a1+1=21, a3=2a2+1=43, and so on.

Recursive functions

```
int recursion(n)
{
  if(n==0)
 {
    return;
  }
   return(n-1);
}
```



Passing Strings to functions

- The strings are treated as character arrays in c and therefore the rules for passing strings to function are very similar to those for passing arrays to functions.

- Basic rules are

  The string to be passed must be declared as a formal Argument of the function when it is defined

  Example:

```
Void display(char item_name[]}
{
....
}
```

UNIT 7

Structure and Union

- A **structure** is a collection of logically related data items grouped together under a single name, called structure tag. The data items enclosed within a structure are known as members. The members can be of same or different data types.

```
struct structure_name
{
data_type member1 ;
data_type member2 ;
data_type member3 ;
}
```

Examples of Structure can be as follows

```
struct employee {
        int emp_id ;
        char name [ 2 5 ] ;
        int age ;
        f l o a t salary ;
} e1 , e2 ;
```

The members of a structure do not occupy memory until they are associated with a structure variable. Above examples shows the structure variable e1 and e2 of structure employee.

Initialization of structure variable :

- ✓ Like primary variables structure variables can also be initialized when they are declared. Structure templates can be defined locally or globally. If it is local it can be used within that function. If it is global it can be used by all other functions of the program.

We can't initialize structure members while defining the structure

```
struct student
{
int age=20;
char name[20]="sona";
}s1;
```

The above is **invalid**.
A structure can be initialized as

```
struct student
{
int age,roll;
char name[20];
} struct student s1={16,101,"sona"};
 struct student s2={17,102,"rupa"};
```

If initialiser is less than no.of structure variable, automatically rest values are taken as zero.

Size of structure :

- ✓ Size of structure can be found out using sizeof() operator with structure variable name or tag name with keyword.

```
sizeof(struct student); or
sizeof(s1);
sizeof(s2);
```

Size of structure is different in different machines. So size of whole structure may not be equal to sum of size of its members.

Accessing Members of Structure:

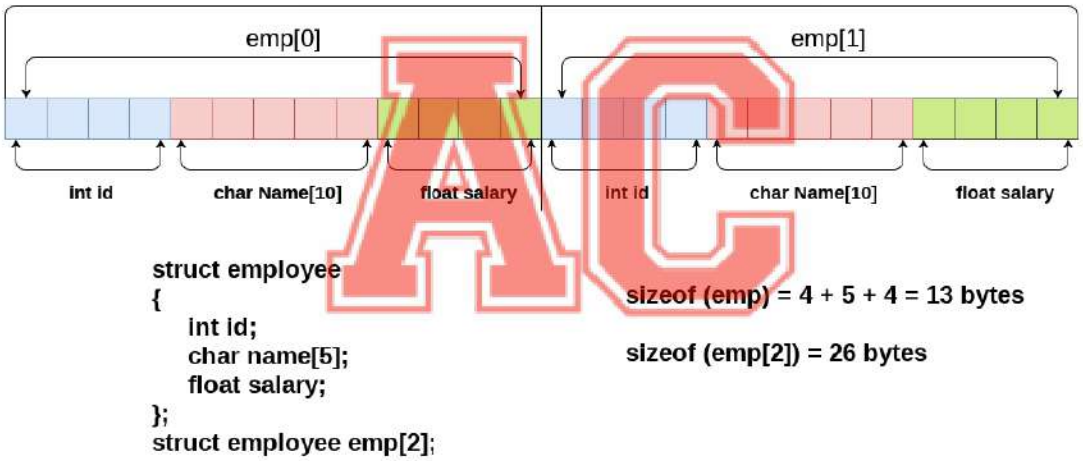
- ✓ The members of a structure can be accessed with the help of .(dot) operator. The syntax for accessing the members of the structure variable is as follows

```
struct_variable . member
struct employee e1 ;
e1 . emp id ;
e1 . name ;
e1 . salary ;
```

Array of Structures :

- ✓ An array of structres in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.

Array of structures



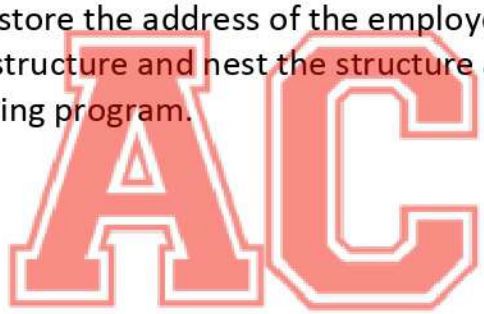
Let's see an example of an array of structures that stores information of 5 students and prints it.

- #include<stdio.h>
- #include <string.h>
- **struct** student{
- **int** rollno;
- **char** name[10];
- };
- **int** main(){
- **int** i;
- **struct** student st[5];
- printf("Enter Records of 5 students");

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

Nested Structure :

✓ C provides us the feature of nesting one structure within another structure by using which, complex data types are created. For example, we may need to store the address of an entity employee in a structure. The attribute address may also have the subparts as street number, city, state, and pin code. Hence, to store the address of the employee, we need to store the address of the employee into a separate structure and nest the structure address into the structure employee. Consider the following program.



```
#include<stdio.h>
struct address
{
char city[20];
int pin;
char phone[14];
};
struct employee
{
char name[20];
struct address add;
};
void main ()
{
struct employee emp;
printf("Enter employee information?\n");
scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin, emp.add.phone);
printf("Printing the employee information....\n");
printf("name: %s\nCity: %s\nPincode: %d\nPhone: %s",emp.name,emp.add.city,emp.add.pin,emp.add.phone)
;
}
```

Unions :

- ✓ The Union is a user-defined data type in C language that can contain elements of the different data types just like structure. But unlike structures, all the members in the C union are stored in the same memory location. Due to this, only one member can store data at the given instance.

Union Declaration :

- ✓ In this part, we only declare the template of the union, i.e., we only declare the members' names and data types along with the name of the union. No memory is allocated to the union in the declaration.

Syntax :-

```
union union_name {
datatype member1;
 datatype member2;
 ...
};
```

****Note** :- Keep in mind that we have to always end the union declaration with a semi-colon.

Union definition :

- ✓ We need to define a variable of the union type to start using union members. There are two methods using which we can define a union variable.

  - With Union Declaration
  - After Union Declaration

1. Defining Union Variable with Declaration

Syntax :-

```
union union_name {
 datatype member1;
datatype member2;
...
} var1, var2, ...;
```

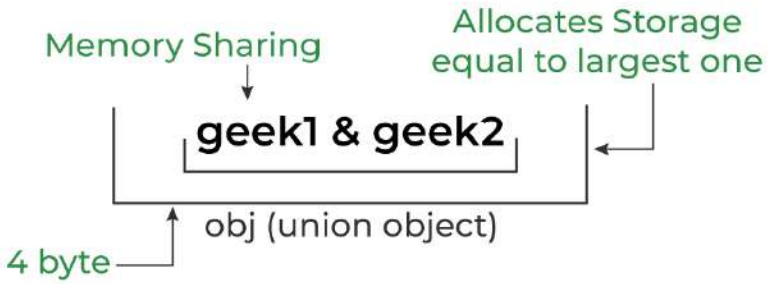
2. Defining Union Variable after Declaration

Syntax :-

```
union union_name var1, var2, var3...;
```

Size of Union :

- ✓ The size of the union will always be equal to the size of the largest member of the array. All the less-sized elements can store the data in the same space without any overflow.



Difference between Struct and Union:-

Struct	Union
The struct keyword is used to define a structure.	The union keyword is used to define union.
When the variables are declared in a structure, the compiler allocates memory to each variables member. The size of a structure is equal or greater to the sum of the sizes of each data member.	When the variable is declared in the union, the compiler allocates memory to the largest size variable member. The size of a union is equal to the size of its largest data member size.
Each variable member occupied a unique memory space.	Variables members share the memory space of the largest size variable.
Changing the value of a member will not affect other variables members.	Changing the value of one member will also affect other variables members.
Each variable member will be assessed at a time.	Only one variable member will be assessed at a time.
We can initialize multiple variables of a structure at a time.	In union, only the first data member can be initialized.
All variable members store some value at any point in the program.	Exactly only one data member stores a value at any particular instance in the program.
The structure allows initializing multiple variable members at once.	Union allows initializing only one variable member at once.
It is used to store different data type values.	It is used for storing one at a time from different data type values.
It allows accessing and retrieving any data member at a time.	It allows accessing and retrieving any one data member at a time.

UNIT 8

Pointer

Introduction To Pointer :

- ✓ A pointer is a variable that store memory address or that contains address of another variable where addresses are the location number always contains whole number. So, pointer contain always the whole number. It is called pointer because it points to a particular location in memory by storing address of that location.

Syntax:-

Data type *pointer name;

Address (&) and indirection (*) operator :-

The Address of Operator (&)

- ✓ Address (&) is an operator that provides the address of a variable. It is a unary operator that operates on only one operand. This operand can only be a variable name, we cannot use it with a constant value.

We can easily identify that variable by the variable address provided by the Address Operator (&). Address operator (&) is also called referencing operator.

Syntax :- &variable_name;

Example :-

```
#include<stdio.h>
void main()
{
 int x = 5;
 printf(" %d \n ", x );
 printf("%d \n", &x);
}
```

The Asterisk (*) or Indirection Operator

✓ Asterisk (*) is an operator that returns the value stored in a variable address. It is a unary operator that operates on only one operand. Indirection operator (*) takes the address of the variable as an argument. It is also called dereferencing operator.

Syntax :- *&variable_name;

Example :-

```
#include<stdio.h>
void main()
{
int x =5;
int y = *&x;
printf(" y = %d ", y);
}
```

Pointer Arithmetic Operations :

✓ Pointer Arithmetic is the set of valid arithmetic operations that can be performed on pointers. The pointer variables store the memory address of another variable. It doesn't store any value.

Hence, there are only a few operations that are allowed to perform on Pointers in C language. The C pointer arithmetic operations are slightly different from the ones that we generally use for mathematical calculations. These operations are:

1. Increment/Decrement of a Pointer
2. Addition of integer to a pointer
3. Subtraction of integer to a pointer
4. Subtracting two pointers of the same type
5. Comparison of pointers

1. Increment/Decrement of a Pointer

- ✓ **Increment:** It is a condition that also comes under addition. When a pointer is incremented, it actually increments by the number equal to the size of the data type for which it is a pointer.

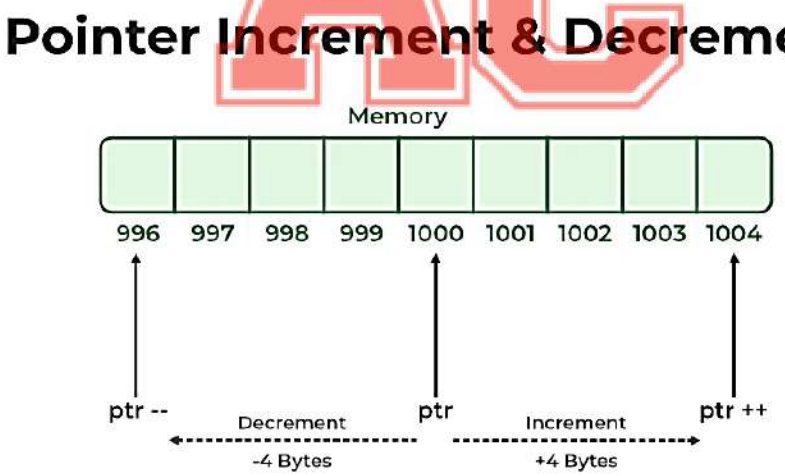
  For Example:

  If an integer pointer that stores address 1000 is incremented, then it will increment by 4(size of an int), and the new address will point to 1004. While if a float type pointer is incremented then it will increment by 4(size of a float) and the new address will be 1004.

- ✓ **Decrement**: It is a condition that also comes under subtraction. When a pointer is decremented, it actually decrements by the number equal to the size of the data type for which it is a pointer.

  For Example:

  If an integer pointer that stores address 1000 is decremented, then it will decrement by 4(size of an int), and the new address will point to 996. While if a float type pointer is decremented then it will decrement by 4(size of a float) and the new address will be 996.

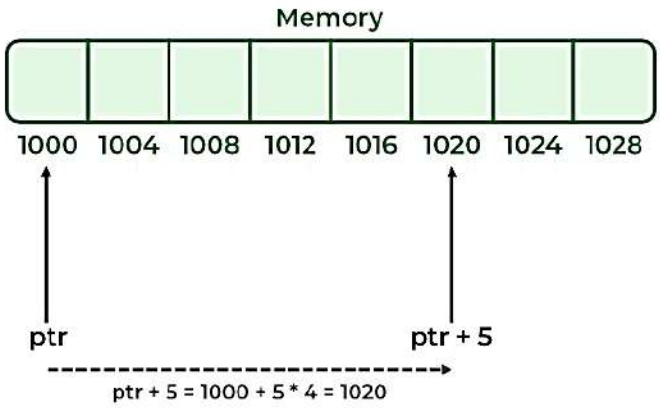


2. Addition of Integer to Pointer

- ✓ When a pointer is added with an integer value, the value is first multiplied by the size of the data type and then added to the pointer.

  For Example:- Consider the same example as above where the ptr is an integer pointer that stores 1000 as an address. If we add integer 5 to it using the expression, ptr = ptr + 5, then, the final address stored in the ptr will be ptr = 1000 + sizeof(int) * 5 = 1020.

Pointer Addition

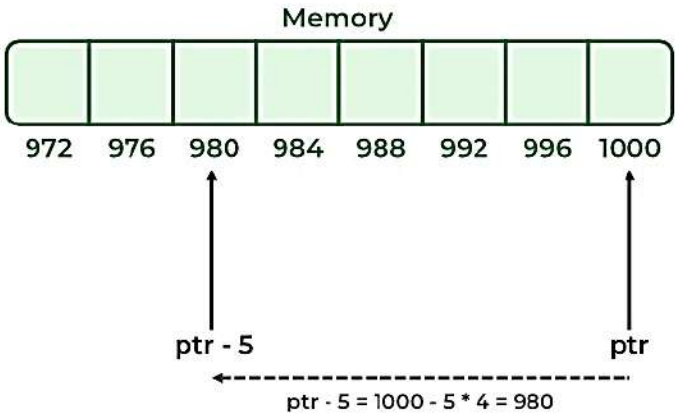


3. Subtraction of Integer to Pointer

- ✓ When a pointer is subtracted with an integer value, the value is first multiplied by the size of the data type and then subtracted from the pointer similar to addition.

  For Example:- Consider the same example as above where the ptr is an integer pointer that stores 1000 as an address. If we subtract integer 5 from it using the expression, ptr = ptr – 5, then, the final address stored in the ptr will be ptr = 1000 – sizeof(int) * 5 = 980.

Pointer Subtraction



4. Subtraction of Two Pointers

✓ The subtraction of two pointers is possible only when they have the same data type. The result is generated by calculating the difference between the addresses of the two pointers and calculating how many bits of data it is according to the pointer data type. The subtraction of two pointers gives the increments between the two pointers.

For Example:- Two integer pointers say ptr1(address:1000) and ptr2(address:1004) are subtracted. The difference between addresses is 4 bytes. Since the size of int is 4 bytes, therefore the increment between ptr1 and ptr2 is given by (4/4) = 1.

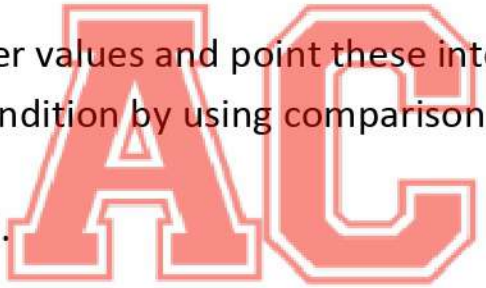
5. Comparison of Pointers

✓ We can compare the two pointers by using the comparison operators in C. We can implement this by using all operators in C >, >=, <, <=, ==, !=. It returns true for the valid condition and returns false for the unsatisfied condition.

Step 1: Initialize the integer values and point these integer values to the pointer.

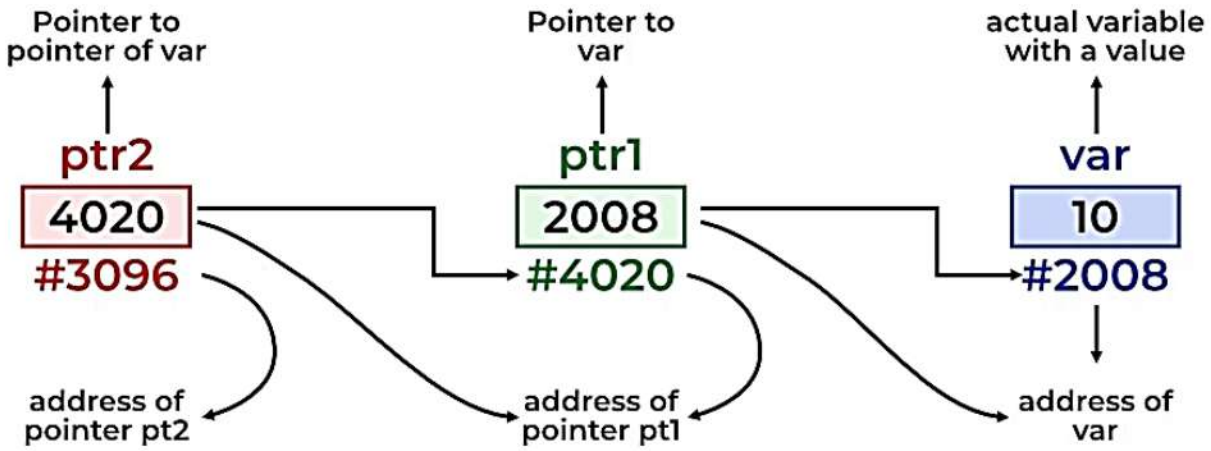
Step 2: Now, check the condition by using comparison or relational operators on pointer variables.

Step 3: Display the output.



Pointer to Pointer in C

✓ The pointer to a pointer in C is used when we want to store the address of another pointer. The first pointer is used to store the address of the variable. And the second pointer is used to store the address of the first pointer. That is why they are also known as **double-pointers**. We can use a pointer to a pointer to change the values of normal pointers or create a variable-sized 2-D array. A double pointer occupies the same amount of space in the memory stack as a normal pointer.



Dynamic Memory Allocation (malloc(), calloc(), realloc(), free())

✓ C **Dynamic Memory Allocation** can be defined as a procedure in which the size of a data structure (like Array) is changed during the runtime.

C provides some functions to achieve these tasks. There are 4 library functions provided by C defined under <stdlib.h> header file to facilitate dynamic memory allocation in C programming. They are:

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

1. malloc()

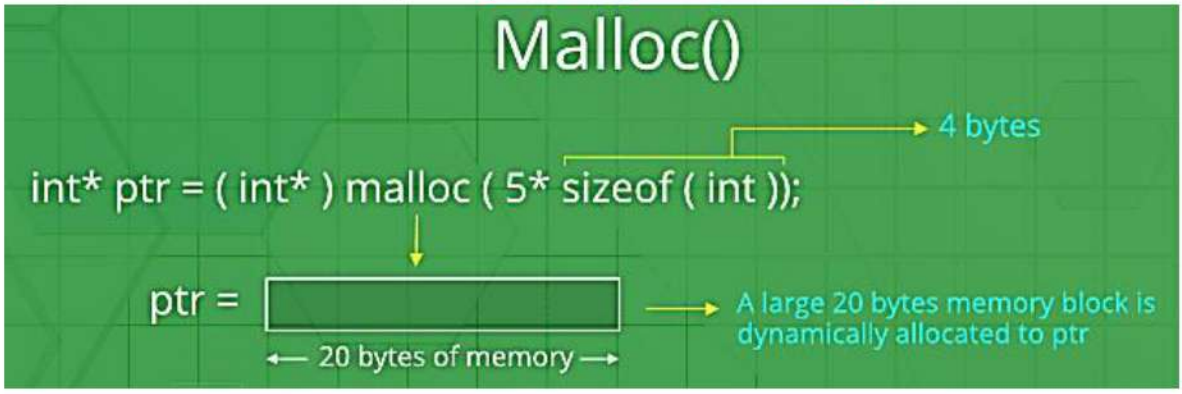
✓ The "**malloc**" or "**memory allocation**" method in C is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form. It doesn't Initialize memory at execution time so that it has initialized each block with the default garbage value initially.

Syntax :-

ptr = (cast-type*) malloc(byte-size)

For Example:- ptr = (int*) malloc(100 * sizeof(int));

Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.



2. calloc()

- ✓ "**calloc**" or "**contiguous allocation**" method in C is used to dynamically allocate the specified number of blocks of memory of the specified type. it is very much similar to malloc() but has two different points and these are:
- ✓ It initializes each block with a default value '0'.
- ✓ It has two parameters or arguments as compare to malloc().

Syntax :-

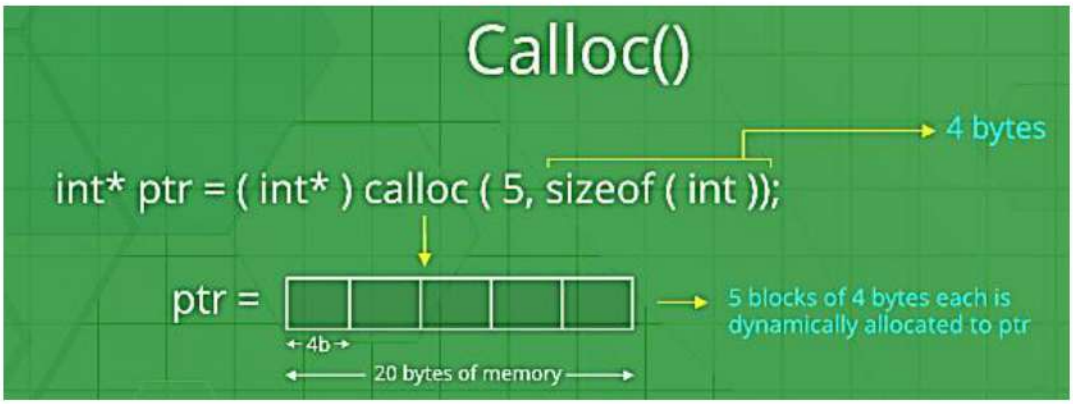
ptr = (cast-type*)calloc(n, element-size);

Here, n is the no. of elements and element-size is the size of each element.

For Example:-

ptr = (float*) calloc(25, sizeof(float));

This statement allocates contiguous space in memory for 25 elements each with the size of the float.

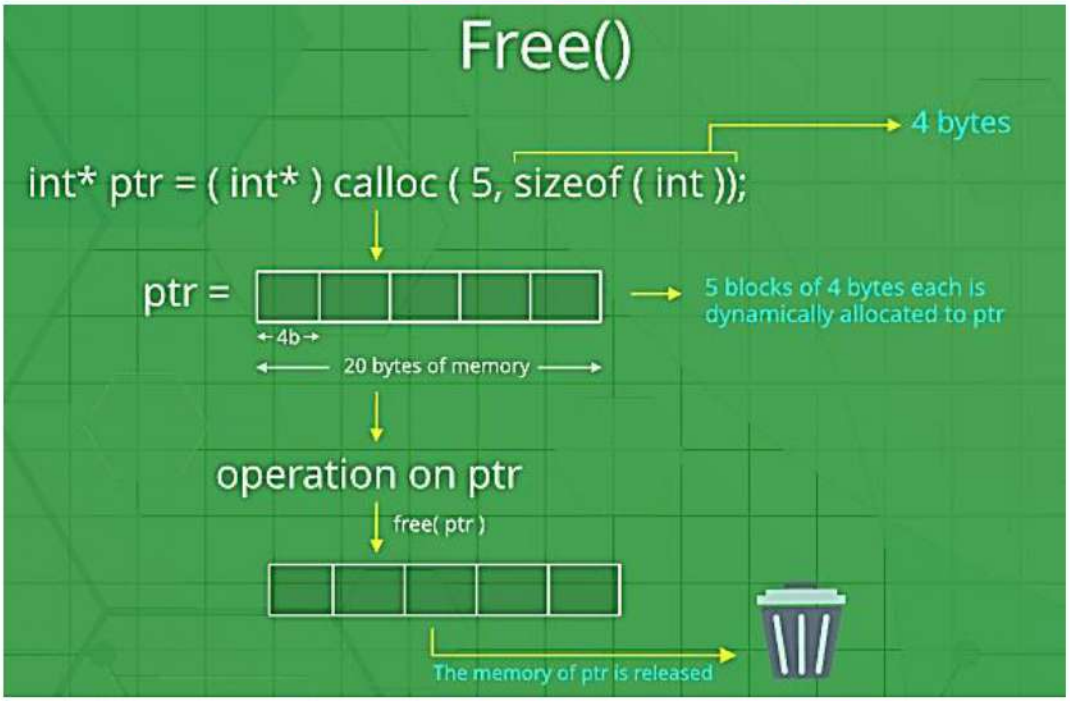


3. C free()

- ✓ "**free**" method in C is used to dynamically **de-allocate** the memory. The memory allocated using functions malloc() and calloc() is not de-allocated on their own. Hence the free() method is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

Syntax :-

free(ptr);

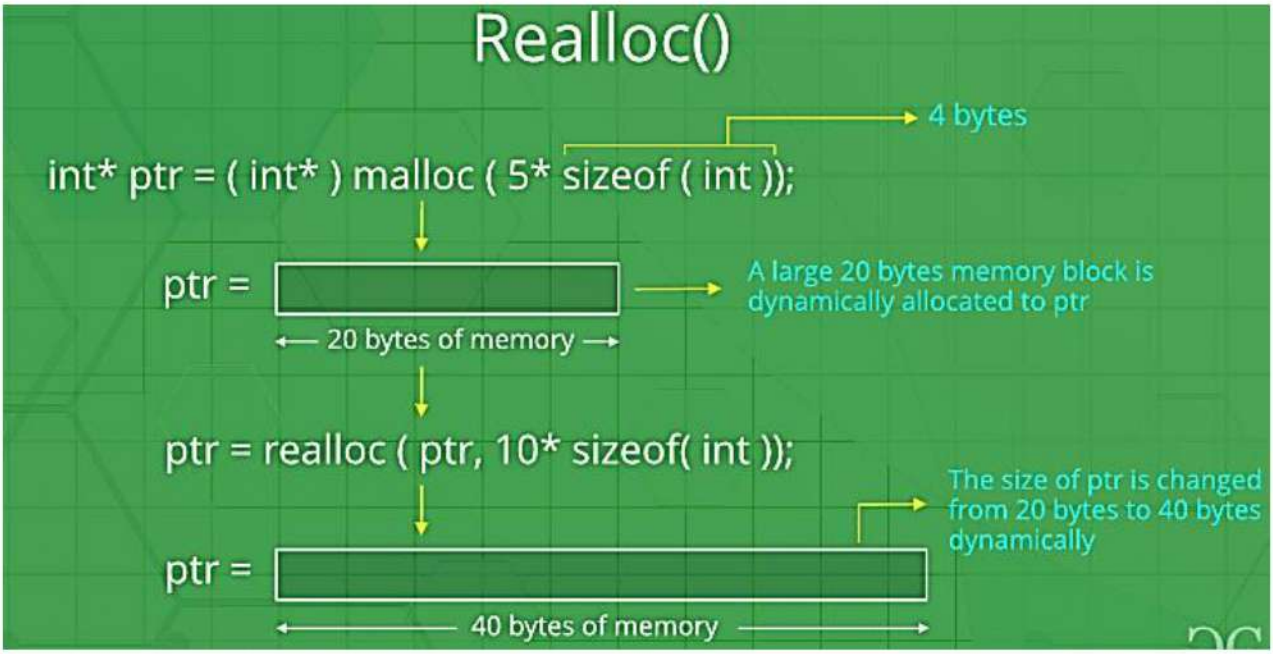


4. C realloc()

- ✓ "**realloc**" or "**re-allocation**" method in C is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to **dynamically re-allocate memory**. re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value.

Syntax :-

ptr = realloc(ptr, newSize);
where ptr is reallocated with new size 'newSize'.



UNIT 9

Data files

Introduction to data files

✓ A data files are used to store and manage information outside of the program's memory. They can be text-based or binary and are accessed using file handling functions provided by the standard library.

Types of Files in C

✓ A file can be classified into two types based on the way the file stores the data. They are as follows:

- Text Files
- Binary Files

1. Text Files

✓ A text file contains data in the **form of ASCII characters** and is generally used to store a stream of characters.

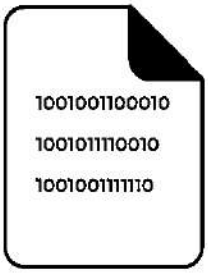
- Each line in a text file ends with a new line character ('\n').
- It can be read or written by any text editor.
- They are generally stored with **.txt file** extension.
- Text files can also be used to store the source code.

file.txt

2. Binary Files

✓ A binary file contains data in **binary form (i.e. 0's and 1's)** instead of ASCII characters. They contain data that is stored in a similar manner to how it is stored in the main memory.

- The binary files can be created only from within a program and their contents can only be read by a program.
- More secure as they are not easily readable.
- They are generally stored with **.bin file** extension.



file.bin

File handing in C

- ✓ File handing in C is the process in which we create, open, read, write, and close operations on a file. C language provides different functions such as fopen(), fwrite(), fread(), fseek(), fprintf(), etc. to perform input, output, and many different C file operations in our program.

Why do we need File Handling in C?

- ✓ So far the operations using the C program are done on a prompt/terminal which is not stored anywhere. The output is deleted when the program is closed. But in the software industry, most programs are written to store the information fetched from the program. The use of file handling is exactly what the situation calls for.
- ✓ In order to understand why file handling is important, let us look at a few features of using files:-
  - **Reusability**: The data stored in the file can be accessed, updated, and deleted anywhere and anytime providing high reusability.
  - **Portability:** Without losing any data, files can be transferred to another in the computer system. The risk of flawed coding is minimized with this feature.
  - **Efficient:** A large amount of input may be required for some programs. File handling allows you to easily access a part of a file using few instructions which saves a lot of time and reduces the chance of errors.
  - **Storage Capacity:** Files allow you to store a large amount of data without having to worry about storing everything simultaneously in a program.

File handling operation :

- ✓ C file operations refer to the different possible operations that we can perform on a file in C such as:
  - Creating a new file – fopen() with attributes as “a” or “a+” or “w” or “w+”
  - Opening an existing file – fopen()
  - Reading from file – fscanf() or fgets()
  - Writing to a file – fprintf() or fputs()
  - Moving to a specific location in a file – fseek(), rewind()
  - Closing a file – fclose()

The highlighted text mentions the C function used to perform the file operations.

File operation	Declaration & Description
fopen() - To open a file	Declaration: FILE *fopen (const char *filename, const char *mode) fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist. FILE *fp; fp=fopen ("filename", "'mode"); Where, fp – file pointer to the data type "FILE". filename – the actual file name with full path of the file. mode – refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+. Please refer below the description for these mode of operations.
fclose() - To close a file	Declaration: int fclose(FILE *fp); fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below. fclose (fp);
fgets() - To read a file	Declaration: char *fgets(char *string, int n, FILE *fp) fgets function is used to read a file line by line. In a C program, we use fgets function as below. fgets (buffer, size, fp); where, buffer – buffer to put the data in. size – size of the buffer fp – file pointer
fprintf() - To write into a file	Declaration: int fprintf(FILE *fp, const char *format, ...);fprintf() function writes string into a file pointed by fp. In a C program, we write string into a file as below. fprintf (fp, "some data"); or fprintf (fp, "text %d", variable_name);

File Pointer in C :

- ✓ A file pointer is a reference to a particular position in the opened file. It is used in file handling to perform all file operations such as read, write, close, etc. We use the FILE macro to declare the file pointer variable. The **FILE** macro is defined inside **<stdio.h>** header file.

Syntax :-

```
FILE* pointer_name;
```

File Pointer is used in almost all the file operations in C.

Opening File :

- ✓ f open( ) is a C library function is used to per an existing file or createing a new file.

Syntax :-

```
FILE* fopen(const char *file_name, const char *access_mode);
```

The modes for opening a file are as follows :-

a) r →open in read mode.

b) rt →open in both read and write mode.

c) w →open or create a text file in write mode.

d) wt →open file in both read and write mode.

e) a →open in append mode.

f) at →open in both read, write and append mode.

Closing File :

- ✓ The fclose() function is used to close the file. After successful file operations, you must always close a file to remove it from the memory.

Syntax :-

```
fclose(file_pointer);
```

where the file_pointer is the pointer to the opened file.

Creating File :

✓ The fopen() function can not only open a file but also can create a file if it does not exist already. For that, we have to use the modes that allow the creation of a file if not found such as w, w+, wb, wb+, a, a+, ab, and ab+.

Syntax :-

```
FILE *fptr;

fptr = fopen("filename.txt", "w");
```

Library functions for READING from a file and WRITING to a file :-

- ***fopen( ):***
  It is a C library function used to open an existing file or create a new file.

- ***fclose( ):***
  It is a C library function that released the memoty stream opened by fopen () function.
  Syntax:- fclose(FILE*stream);

- ***fgetc( ):***
  It is a C library function which reads a character from a file that has been opened in read mode by the fopen ( ) function.
  Syntax:- getc(FILE*stream);

- ***putc ( ):***
  It is a C library function which is used to write a character to a file. It is used to write single character to the stream as well as advance the position of the indicator.
  Syntax:- putc(int c ,FILE*stream);

- ***getw( ):***
  It is used to read integer from a file that has been opened in read mode. It is file handing function that is used to read integer value.
  Syntax:- getw(FILE*stream);

- ***putw( ):***

  It is used to write an integer to a file.

  Syntax:- int putw(int c,FILE*stream);

- ***fprintf( ):***

  It passes arguments according to specified format to the file indicated by the stream. It is implemented in file related program for writing formatted data in any file.

  Syntax:- int fprintf(FILE*stream, const char*format,..);

- ***fscanf( ):***

  It read formatted input from a file. It is implemented in file related program for reading formatted data from any file specified in the program.

  Syntax:- int fscanf(FILE*stream, const char*format,..);

- ***fgets ( ):***

  It is implemented in file related program for reading strings from any particular file. It gets the string 1 line each time.

  Syntax:- char*fgets(char*str,int n ,FILE*stream);

- ***fputs ( ):***

  It is implemented in file related program for wrting strings from any particular file.

  Syntax:- int fputs(const char*str,FILE*stream);

- ***feof( ):***

  It is used to determine in the end of the file (stream) specified has been ereached or not. It keeps on searching the end of the file (EOF) In your file program.

  Syntax:- int feof(FILE*stream);

***