C - PROGRAMMING

(For Problem Solving)

Module-1
(Introduction to Programming)

Programming Basics: Steps to solve logical and numerical problems. Representation of Algorithm, Flowchart/Pseudo code with examples, Program design and structured programming, Role of assemblers, compilers, linker, loader, interpreter in program execution.

Introduction to C: Basic structure of C program, C compilation process with different stages, standard library and header files, Syntax and Semantics. Simple Input and output with scanf and printf, formatted I/O, Introduction to stdin, stdout and stderr, variables(with data types and space requirements), datatypes, literals, operators, expressions and precedence, Expression Evaluation, Storage classes(auto, extern, static and register), type conversion, command line arguments.

Module – 2
(Flow of Control and Arrays)

Conditional, Branching and Loops: Writing and evaluation of conditionals and consequent branching with if, if-else, switch-case, ternary operator, go to, Iteration with for while and do-while loops, use of break & continue statements, common programming errors.

Array: One and two-dimensional arrays, creating, accessing and manipulating elements of arrays, solving logical problems using array.

Module – 3
(Strings, Structures and Pointers)

Strings: Introduction to strings, handling string as array of characters, basic string functions available in C(Strlen, strcpy, strstr etc), array of strings, solving logical problems using string.

Structures: Defining structures, initializing structures, unions, array of structures, nested structures.

Pointers: Idea of pointers, Defining pointers, Pointer to Pointer, Pointer to Arrays and Structures, Pointer to strings, Use of Pointers in self-referential structures, usage of self-referential structures in linked lists(no implementation), Enumeration data types.

Module – 4
(Functions and DMA)

Functions: Designing structures programs, declaring functions, signature of a function, Parameters and return type of a function, passing parameters to functions, call by values, passing arrays to functions, passing pointers to functions, idea of call by reference, Some C standard functions and libraries.

Recursion: Simple programs, such as finding Factorial, Fibonacci Series, Limitations of Recursing functions.

Dynamic Memory Allocation: Allocating and Freeing memory, Allocating memory to Arrays of different data types, Allocating memory to structures, Re-allocating the memory(modify the size of array) and freeing the memory using free function.

Module – 5
(Pre-processor and File handling in C)

Pre-Processor: Commonly used Pre-processor commands like include, define, undef, if, ifdef, ifndef.

File Handling: Text and Binary files, Creating and Reading and Writing text and binary files, Appending data to existing files, Writing and reading structures using binary files, Random access using fseek, ftell and rewind functions. File Handling in C, file types, file opening modes, file handling I/O – fprintf, fscanf, fwrite, fread, fseek, File pointers, implementing basic file operations in C.

Module-1

(Introduction to Programming)

Introduction to Programming

Program

A program is a set of instructions given to a computer to do a specific task.

Examples: A program that adds two numbers
A Programs shows today's date.

Programming Language

A programming language is a way for humans to talk to computers.
We use it to write programs.
Example: C, Java, Python.

Source Code

The instructions we write in a programming language are called source code.

Example:

```
print("Hello")
print("Welcome to C")
print("Programming World")
```

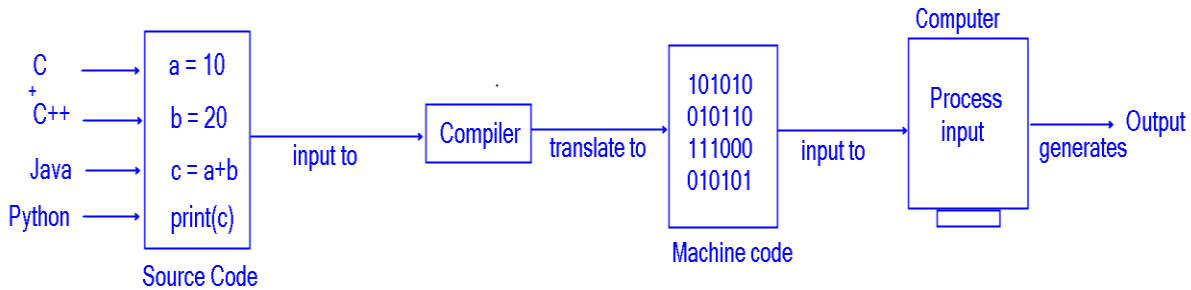
Compiler

A compiler is a special software that translates source code (human-written instructions) into machine code (computer language: 0s and 1s).
This lets the computer understand and run the program.

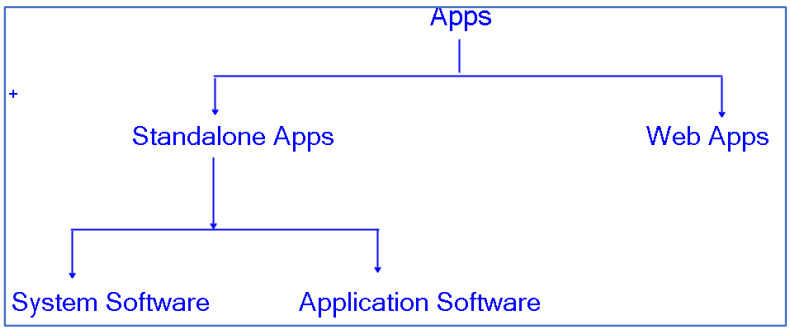
In short:

Program = Instructions.
Programming Language = The language used to write instructions.
Source Code = The written instructions.
Compiler = The translator that makes the computer understand the instructions.

The Following Diagram explains the process of communication:



Need of programming: Programming languages are mainly used to develop applications. Applications are different types like below,



Standalone Applications

- A standalone application is a program that runs on a single computer.
- No need of the Internet Connection.
- It works independently once installed.

Types of Standalone Applications

1. **System Software**
   - Helps the computer to work properly.
   - **Example:** Operating System (Windows, Linux), Device Drivers.

2. **Embedded Software**
   - Special software written for other electronic devices.
   - **Example:** Software in washing machines, ATMs, microwaves, cars.

3. **Application Software**
   - Software made for users to do specific tasks.
   - **Example:** MS Word (for documents), VLC Player (for videos), Photoshop (for editing).

Web Applications

- A web application is a program that runs on a server and we use it through a web browser (like Chrome, Edge, Firefox).
- It usually needs the internet to work.

Examples:

- Gmail (email through browser)
- Facebook, Instagram (social media)
- Online Banking apps
- Flipkart, Amazon (shopping websites)

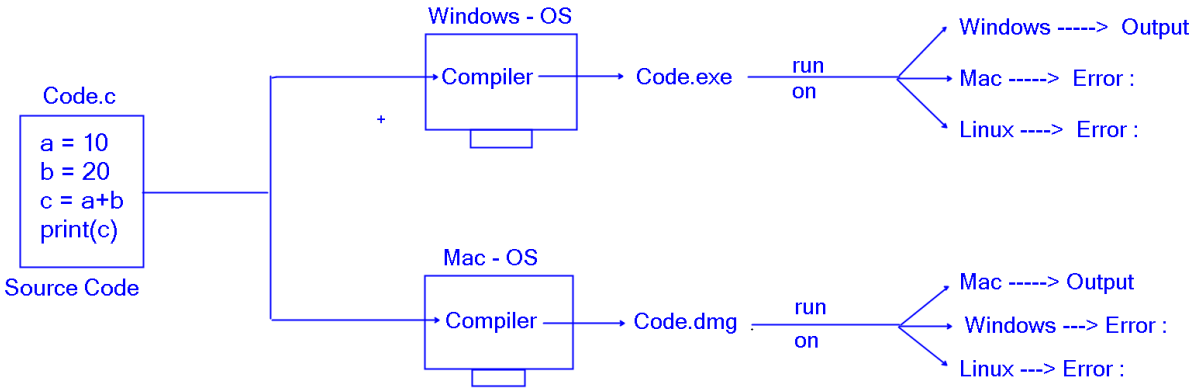
Platform Dependency:

- The program works only on the system/OS it was created for.
- If you write the program on one platform(like Windows), you cannot directly run it on another(like Linux) without changes.

Example:

- C, C++ programs : After compiling on Windows, the .exe file runs only on Windows, not on Linux.
- A Windows software cannot run on Mac unless modified.

"Write once, run only on the same platform."



Platform Independency:

- The program can run on any operating system without rewriting the code.
- You just write once, and it can run anywhere (Windows, Linux, Mac).

Example:

- **Java :** Java programs are compiled into bytecode, which runs on JVM (Java Virtual Machine).
- Python also works on multiple platforms.

"Write once, run anywhere."

Feature	Platform Dependent	Platform Independent
Works on	Only same OS	Any OS (with JVM/interpreter)
Example Languages	C, C++	Java, Python
Slogan	Write once, run there	Write once, run anywhere

Steps to Solve Logical and Numerical Problems:

When you get a problem (logical or numerical), don't jump to the answer directly. Follow these steps:

1. Understand the Problem
    a. Read carefully.
    b. Identify what is given and what is asked.
    c. **Example:** "Find the sum of first 10 numbers." : Given: 10, Asked: sum.

2. Break into Small Steps
    a. Divide the big problem into smaller steps.
    b. **Example:** First take numbers, then add one by one.

3. Choose a Method/Approach
    a. **Think how to solve:** formula, logic, or calculation.
    b. **Example:** Use formula n(n+1)/2 for sum of first n numbers.

4. Write the Steps (Algorithm)
    a. Before coding, write the step-by-step plan (like recipe).

5. Do Calculation or Coding
    a. Perform math OR write program using the algorithm.

6. Check Your Answer
    a. Verify if result is correct.
    b. Example: For 10 numbers, sum = 55.

Algorithm (Definition + Example)

- An algorithm is a step-by-step method to solve a problem.
- It is like a recipe that tells exactly what to do, in order.

Example Algorithm: Find sum of first 10 numbers

Start
Take n = 10
Initialize sum = 0
Repeat from 1 to n → Add each number to sum
Print sum
Stop

Output = 55

Practice – Writing Algorithms

Algorithm: Add Two Numbers
Problem: Find the sum of two numbers.

```
Start
Read two numbers (say a and b)
Calculate sum = a + b
Print sum
Stop
```

Algorithm: Check Number is Even or Not
Problem: Check whether a given number is even.

```
Start
Read a number n
If (n % 2 == 0) then
    Print “Even”
Else
    Print “Odd”
Stop
```

Algorithm: Find the Largest of Two Numbers
Problem: Read two numbers and find which one is bigger.

```
Start
Read two numbers (say a and b)
If a > b then
    Print “a is larger”
Else
    Print “b is larger”
Stop
```

Algorithm: Print Numbers from 1 to 10
Problem: Display numbers from 1 to 10.

```
Start
Initialize i = 1
Repeat until i <= 10
    Print i
    Increase i = i + 1
Stop
```

Flow Charts

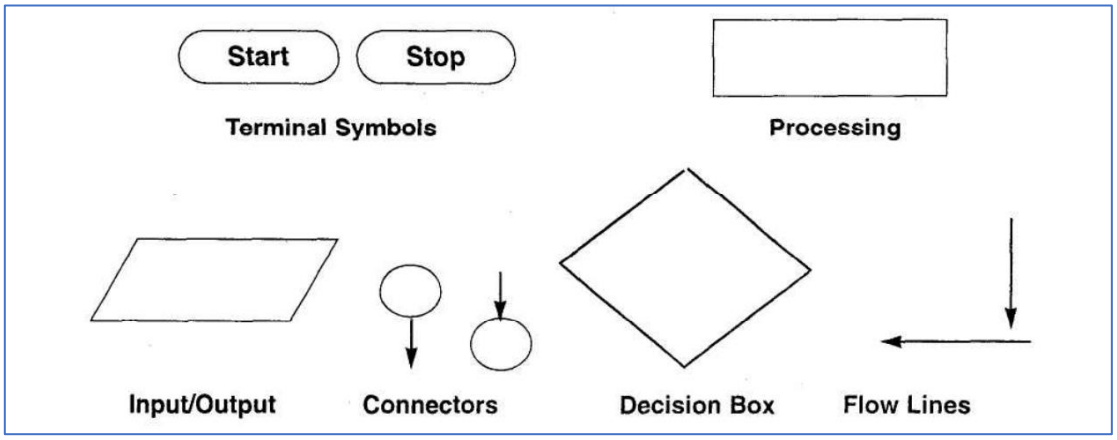
What is a Flowchart?

- A flowchart is a picture (diagram) that shows the steps of a program.
- It uses shapes and arrows to explain how the program works step by step.
- It is like a map that guides how to solve a problem.

Why Use Flowcharts?

- To understand the problem easily.
- To plan before writing code.
- To explain logic to others without writing code.

The Symbols Used in Flow Charts:

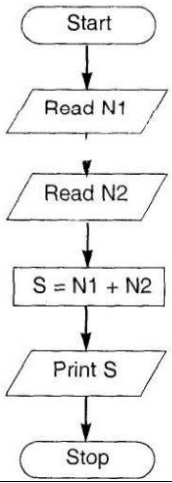


1. **Flow Lines:** Indicates the flow of logic by connecting symbols.
2. **Terminal (Start / Stop):** Represents the start and the end of a flowchart.
3. **Input / Output:** Used for input and output operation.
4. **Processing:** Used for arithmetic operations and data-manipulations.
5. **Decision Box:** Used for decision making between two or more alternatives.
6. **Connectors:** Used to join different flowline

To prepare a flowchart to add two numbers

Algorithm :

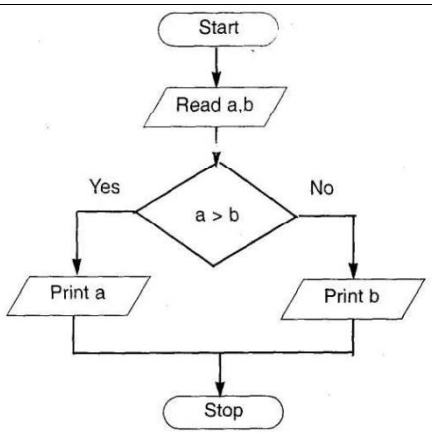
1. Start.
2. Get two numbers N1 and N2.
3. Add them.
4. Print the result.
5. Stop



To prepare a flowchart to determine the greatest of two numbers:

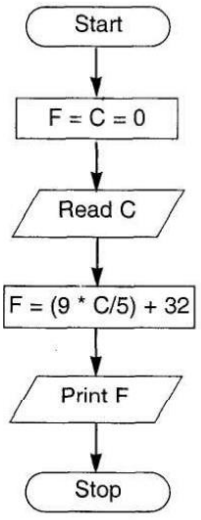
Algorithm:

1. Start
2. Get two number A and B.
3. If A > B
   then print A
   else
   print B.
4. Stop.



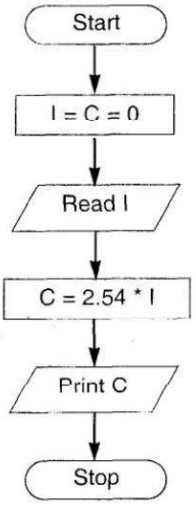
Flowchart for a program that converts temperature in degrees Celsius to degrees Fahrenheit:

1. Start.
2. Create vars F and C (for temperature in Fahrenheit and Celsius).
2. Read degrees Celsius into C.
3. Compute the degrees Fahrenheit into F.
4. Print result (F).
5. Stop.



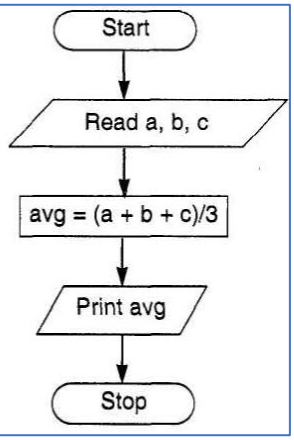
Flowchart for a program that converts inches to centimeters First let us write the steps involved in this computation technique.

1. Start.
2. Create vars C and I (for Centimetres and Inches respectively).
2. Read value of Inches into I
3. Compute the Centimetres into C.
4. Print result (C).
5. Stop.



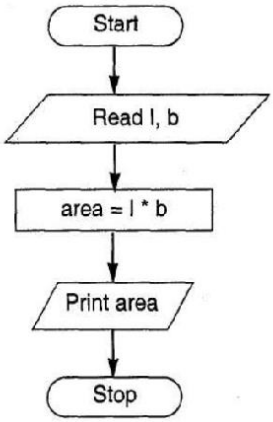
To find average of three numbers:

The steps are :
1. Start
2. Read numbers a, b, c
3. Compute the average as (a + b + c)/3
4. Print average
5. Stop.



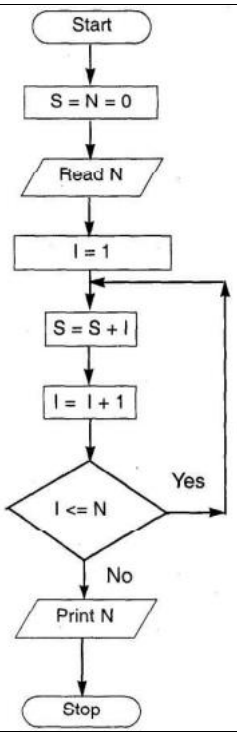
To find area of a rectangle whose length and breadth area read:

The steps are :
1. Start
2. Read length l and breadth b
3. Compute the area as l * b
4. Print area
5. Stop.



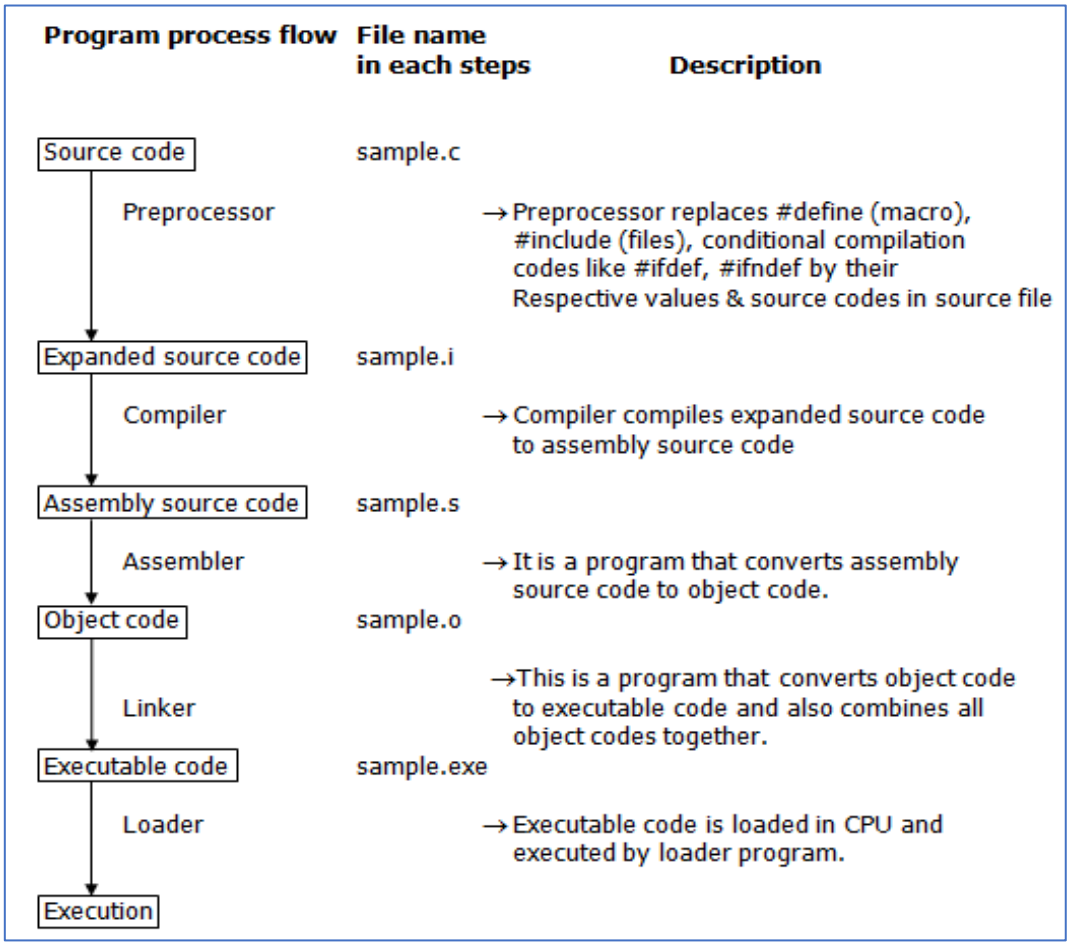
To find the sum of first N numbers:

1. Start
2. Create vars S , N, I
3. Read N
4. Set S (sum) to 0
5. Set counter (I) to 1.
6. S = S + I
7. Increment I by 1.
8. Check if I is less than or equal to N. If no, go to step 6.
9. Print S
10. Stop



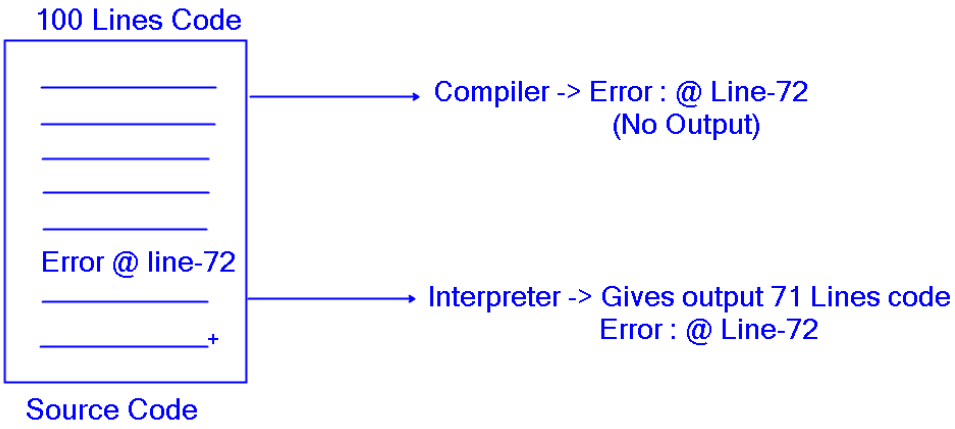
Role of Pre-processor, Assembler, Compiler, Linker and Loader:

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



Compiler v/s Interpreter:

- **Compiler:** All at once code translation (C, C++, Java etc)
- **Interpreter:** Line by line code translation and execution (Python, JavaScript).



What is Pseudo Code?

- Writing the steps of a program in English (with little programming style).
- It is not real code so Computer cannot run it.
- It is just a plan to show how the program works before writing in a programming language.

Add Two Numbers:

```
START
  INPUT a, b
  sum ← a + b
  PRINT sum
STOP
```

Check Even or Odd:

```
START
  INPUT number n
  IF n % 2 == 0 THEN
     PRINT "Even"
  ELSE
     PRINT "Odd"
  ENDIF
STOP
```

Print Numbers from 1 to 10:

```
START
  SET i = 1
  WHILE i <= 10 DO
     PRINT i
     i ← i + 1
  ENDWHILE
STOP
```

Find Largest of Two Numbers:

```
START
  INPUT a, b
  IF a > b THEN
     PRINT "a is largest"
  ELSE
     PRINT "b is largest"
  ENDIF
STOP
```

Find Factorial of a Number (n!):

```
START
  INPUT n
  fact ← 1
  FOR i = 1 TO n DO
     fact ← fact * i
  ENDFOR
  PRINT fact
STOP
```

Program Design and Structured Programming

Program Design

- It is the process of planning a program before writing the actual code.
- It answers "How will I solve the problem step by step?"

Steps in Program Design:

- Understand the problem (what is required).
- Write Algorithm (step-by-step method).
- Draw Flowchart (visual steps).
- Write Pseudo Code (program-like steps).
- Decide programming language (C, Python, Java, etc.).
- Write and test the program.

Example:

- Problem: Add two numbers.
- Algorithm → Step by step
- Flowchart → Diagram
- Pseudo Code → Program-like steps
- Finally, → Real program in C/Python.

Structured Programming:

Structured programming is a way of writing programs that makes them clear, easy to read, test, and debug. C is a structure programming language.

Main Ideas of Structured Programming:

1. Top-Down Approach – break a big problem into smaller problems.
2. Use of Control Structures:
3. Sequence → Steps in order.
4. Selection → if / else (decision making).
5. Iteration → for / while (loops).
6. Modularity – divide the program into functions or procedures.
7. No goto statements – avoid confusing jumps.

Basic Structure of a C Program:

1. **Pre-processor Directives (#include <stdio.h>):** Used to include header files before compilation.
   Example: <stdio.h> for input/output functions (printf, scanf).

2. **Main Function (int main() { … }) :** Every C program starts execution from main(). It is the entry point of the program.

3. **Variable Declarations:** Declare variables before using them.
   Example: int a, b;

4. **Input Section:** Taking values from the user using scanf().

5. **Processing Section:** Actual calculation or logic.
   Example: sum = a + b;

6. **Output Section:** Displaying result using printf().

7. **Return Statement (return 0;):** Tells the operating system that the program finished successfully.

```
#include <stdio.h>   // 1. Preprocessor Directive

// 2. Main Function
int main()
{

   // 3. Variable Declaration
   int a, b, sum;

   // 4. Input
   printf("Enter two numbers: ");
   scanf("%d %d", &a, &b);

   // 5. Processing
   sum = a + b;

   // 6. Output
   printf("Sum = %d", sum);

   return 0;  // 7. End of Program
}
```

C Standard Header Files:

- Header files are files provided by C language that contain pre-written functions.
- We include them using #include to use their functions in our program.

Common C Standard Header Files:

Header File	Purpose / Functions Provided	Example
<stdio.h>	Input/Output functions like printf(), scanf()	printf("Hello")
<conio.h>	Console I/O functions (Turbo C) like clrscr(), getch()	getch();
<stdlib.h>	General utilities: memory allocation & utility	malloc(), rand()
<math.h>	Mathematical functions like sqrt(), pow(), sin(), cos()	sqrt(25)
<string.h>	String handling functions like strcpy(), strlen(), strcmp()	

How to Use Header Files: Include at the top of the program

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

Syntax and Semantics in Programming:

Syntax:

- Syntax is the set of rules that define how we should write a program in a programming language.
- If the syntax is wrong, the program will not compile.

Example in C:

```
int a = 10;   // Correct syntax
int a 10      // Wrong syntax : Missing '=' will cause error
```

Semantics

- Semantics is the meaning of the code we write.
- Even if the syntax is correct, wrong semantics can give wrong results.

Example in C:

```
int a = 10, b = 5;
int sum = a - b;   // Syntax is correct, but if we wanted a+b, semantic is wrong
```

Key Difference

Feature	Syntax	Semantics
Meaning	Rules for writing code	Meaning of the code
Error Type	Compilation error	Logical error / wrong output
Example	Missing semicolon (;)	Wrong formula used

Input and Output in C

printf():

printf() is used to display output on the screen.

Syntax:

printf("Text or format", variables);

```
#include <stdio.h>
int main()
{
   int a = 10;
   printf("Value of a = %d", a);
   return 0;
}
```

Output: Value of a = 10

Note: %d → for integer
%f → for float
%c → for character
%s → for string

scanf():

scanf() is used to read input from the user.

Syntax:

scanf("format", &variable);

```
#include <stdio.h>
int main()
{
   int a;
   printf("Enter a number: ");
   scanf("%d", &a);  // &a → address of variable
   printf("You entered: %d", a);
   return 0;
}
```

Output: Enter a number: 25
You entered: 25

Note: Always use & before the variable in scanf except for String.

Add Two Numbers (Input + Output)

```
#include <stdio.h>
int main()
{
    int a, b, sum;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    sum = a + b;
    printf("Sum = %d", sum);
    return 0;
}
```

Output:

Enter two numbers: 5 7
Sum = 12

Reading Name and Print:

```
#include <stdio.h>
int main()
{
    char name[50];
    printf("Enter your name: ");
    scanf("%s", name);  // Read input (no spaces allowed)
    printf("Hello, %s!", name);  // Print the name
    return 0;
}
```

Output: Enter your name : Amar
Hello, Amar!

Difference Between char[] and char***

Feature	**char[] (Array of char)**	**char* (Pointer to char)**
Memory Allocation	Memory is allocated at compile time	Memory can point anywhere (string literal or dynamically allocated)
Size	Size is fixed at declaration	No fixed size; pointer can point anywhere
Example Declaration	char name[50];	char *name = "Hello";
Usage	For storing user input	For pointing to string literals or dynamic strings
Input Example	scanf("%s", name);	Cannot safely use scanf for string literal pointers

What is Formatted I/O?

- Formatted Input/Output means reading or printing data in a specific format.
- It uses format specifiers to tell the program the type of data (int, float, char, string).
- Mainly done using printf() for output and scanf() for input.

Common Format Specifiers

Specifier	Data Type	Example
%d	Integer	10
%f	Float	12.34
%c	Character	'A'
%s	String	"Ravi"
%lf	Double	45.678
%x	Hexadecimal	0x1A
%o	Octal	012

Read and Print Integer:

```
#include <stdio.h>
int main()
{
    int a;
    printf("Enter an integer: ");
    scanf("%d", &a);
    printf("You entered: %d", a);
    return 0;
}
```

Read and Print Float:

```
#include <stdio.h>
int main()
{
    float num;
    printf("Enter a float number: ");
    scanf("%f", &num);
    printf("You entered: %.2f", num); // .2f → 2 decimal places
    return 0;
}
```

Read and Print Character:

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

```
    char ch;
    printf("Enter a character: ");
    scanf(" %c", &ch); // space before %c to ignore previous newline
    printf("You entered: %c", ch);
    return 0;
}
```

Read and Print String:

```
#include <stdio.h>
int main()
{
    char name[50];
    printf("Enter your name: ");
    scanf("%s", name);  // Reads a single word
    printf("Hello, %s!", name);
    return 0;
}
```

C program to read Name, Age, Height, and Gender and print all values:

```
#include <stdio.h>
int main() {
    char name[50];
    int age;
    float height;
    char gender;

    printf("Enter your name: ");
    scanf("%s", name);  // Reads single word name

    printf("Enter your age: ");
    scanf("%d", &age);

    printf("Enter your height (in meters): ");
    scanf("%f", &height);

    printf("Enter your gender (M/F): ");
    scanf(" %c", &gender);  // Space before %c to avoid newline issue

    printf("%s, %d, %.2f, %c", name, age, height, gender);
    return 0;
}
```

Standard Streams in C

stdin (Standard Input):

- stdin is used to read input from the user.
- By default, it takes input from the keyboard.
- Used by functions like scanf(), getchar(), etc.

```
#include <stdio.h>
int main()
{
    int a;
    fscanf(stdin, "%d", &a);  // Reads integer from standard input (keyboard)
    printf("You entered: %d", a);
    return 0;
}
```

stdout (Standard Output):

- stdout is used to display output to the user.
- By default, it prints on the screen.
- Used by functions like printf(), putchar(), etc.

```
#include <stdio.h>
int main()
{
    fprintf(stdout, "Hello, World!\n");  // Prints to standard output (screen)
    return 0;
}
```

stderr (Standard Error):

- stderr is used to display error messages.
- By default, it also prints to the screen.
- Helps to separate normal output from error messages.

```
#include <stdio.h>
int main()
{
    int a = -5;
    if (a < 0) {
        fprintf(stderr, "Error: Value cannot be negative!\n");  // Print error
    }
    return 0;
}
```

Error: Value cannot be negative!

Variables in C

What is a Variable?

- A variable is a named memory location that stores a value.
- The value stored in a variable can change during program execution.

Example:

```
int age = 20;   // 'age' is a variable storing 20
age = 25;       // value changed to 25
```

Rules for Naming Variables

- Name must start with a letter or underscore _.
- Can contain letters, digits, and underscores.
- Cannot be a keyword (like int, return, etc.).
- Variable names are case-sensitive (age ≠ Age).

Valid names: age, _count, totalMarks
Invalid names: 2age, int, total-marks

Types of Variables in C

Type	Size	Value Example	Purpose
int	2 or 4 B	10, -25	Stores integers (whole numbers)
float	4 B	12.34, -5.67	Stores decimal numbers
double	8 B	12.345678	Stores bigger decimal numbers
char	1 B	'A', 'b'	Stores single characters

Example Program Using Variables:

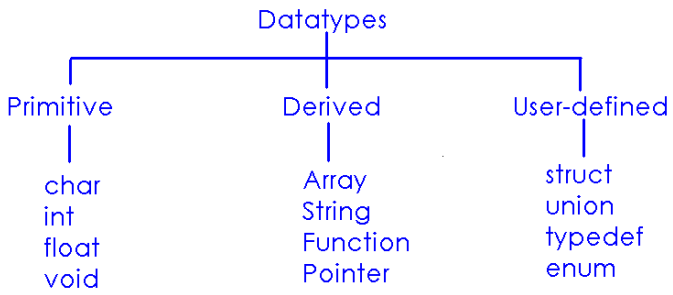
```
#include <stdio.h>
int main() {
    char name[50];
    int age;
    float height;

    printf("Enter your name: ");
    scanf("%s", name);
    printf("Enter your age: ");
    scanf("%d", &age);
    printf("Enter your height (in meters): ");
    scanf("%f", &height);
    printf("Name: %s, Age: %d, Height: %.2f", name, age, height);
    return 0;
}
```

Data Types in C

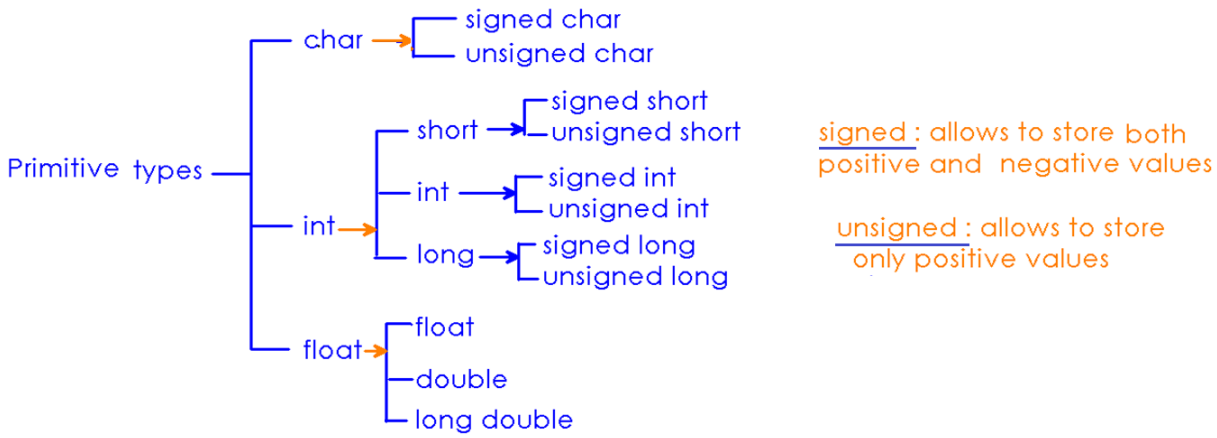
Data type:

- We need to specify the data type in variable declaration.
- Data type specifies, the type of data is allowed to store.

Data types are classified into:



Primitive types: In the above diagram primitive types are generalized. Primitive types sub classified as follows



Following table describes the data type with size, format specifier and limits:

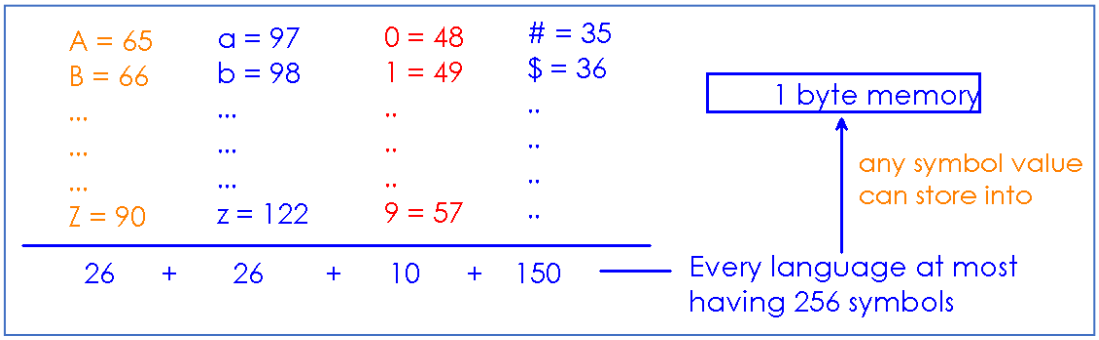
Type	Size(byte)	Symbol	Range
char/signed char	1	%c	-128 to 127
unsigned char	1	%c	0 to 255
Short int /signed short int	1	%d	-128 to 127
unsigned int	1	%u	0 to 255
int /signed int	2	%d	-32768 to 32767
unsigned int	2	%u	0 to 65535
long int/signed long int	4	%ld	-2147483648 to 214748647
unsigned long int	4	%lu	0 to 4294967295
float	4	%f	-3.4e38 to 3.4e38
double	8	%lf	-3.4e308 to 3.4e308
long double	10	%lf	-1.7e4982 to 1.7e4982

Character data type: Char type is used to store alphabets, symbols and digits. We represents character with single quotes.

```
#include <stdio.h>
int main(){
        char v1 = 'a';
        char v2 = '5';
        char v3 = '$';
        printf("v1 value is : %c \n", v1);
        printf("v2 value is : %c \n", v2);
        printf("v3 value is : %c \n", v3);
        return 0;
}
```

How character stores into memory?

- Every character represented by an integer value called ASCII.
- The range of ASCII from 0 to 255(1 byte limits)
- Each character occupies 1 byte memory to store the corresponding integer value.




```
#include <stdio.h>
int main(){
        char v1 = 'a';
        char v2 = '5';
        char v3 = '$';
        printf("%c ascii value is : %d \n", v1, v1);
        printf("%c ascii value is : %d \n", v2, v2);
        printf("%c ascii value is : %d \n", v3, v3);
        return 0;
}
```

Output:

```
a ascii value is : 97
5 ascii value is : 53
$ ascii value is : 36
```

limits.h: It is a header file providing constant variables assigned with limits of each data type..

Program to display short int information (size and limits):

```
#include <stdio.h>
#include <limits.h>
int main()
{
        printf("Signed short min : %d \n", SHRT_MIN);
        printf("Signed short min : %d \n", SHRT_MAX);
        printf("Unsigned short max : %d \n", USHRT_MAX);
        return 0;
}
```

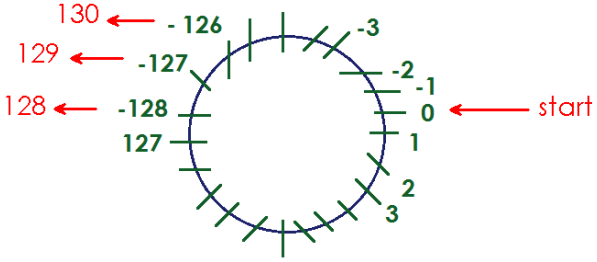
Program to display Character type info:

```
#include <stdio.h>
#include <limits.h>
int main(){
        printf("signed char min val : %d \n", SCHAR_MIN);
        printf("signed char max val : %d \n", SCHAR_MAX);
        printf("unsigned char max val : %d \n", UCHAR_MAX);
        return 0;
}
```

What happens when we try to store the value beyond the limits of datatype?

- Every variable can store the value within the limits only.
- When limit exceeded, it counts the value in the specified limit and store some other value.

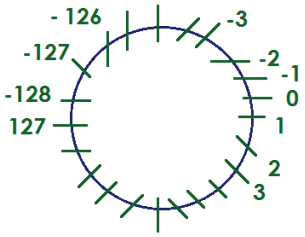
```
#include<stdio.h>
int main(){
        char ch = 130;
        printf("ch val : %d \n", ch);     // prints -126
        return 0;
}
```



```
#include<stdio.h>
int main()
{
        char c1=256, c2=1024;
        printf("c1 : %d \n", c1);
        printf("c2 : %d \n", c2);
        return 0;
}
```

Output:

c1 : 0
c2 : 0

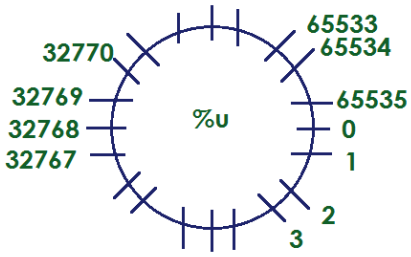


1 circle = 256

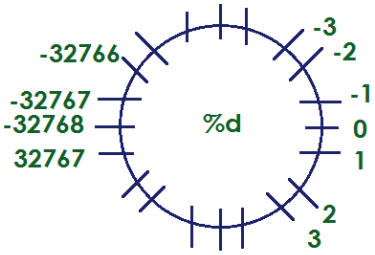
256 = 0 in circle
1024 = 4*256 = 0
65536 = 256*256 = 0

Short int limits:

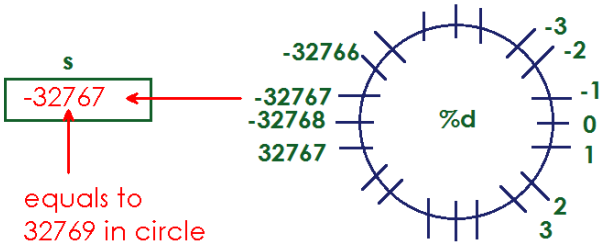
unsigned short : 0 to 65535



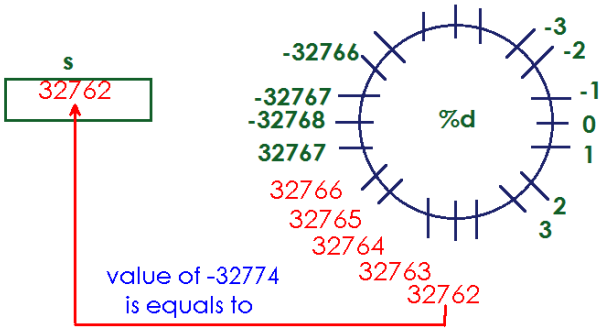
signed short : -32768 to + 32767




```
#include<stdio.h>
int main()
{
        short s = 32769;
        printf("s val : %d \n", s);
        return 0;
}
```




```
#include<stdio.h>
int main()
{
        short s = -32744;
        printf("s val : %d \n", s);
        return 0;
}
```



sizeof(): a pre-defined function used to find the size of Variable, value, Datatype etc.

Program to find the sizes of different variables:

```
#include <stdio.h>
int main(){
        char c;
        short s;
        float f;
        printf("char size : %d \n", sizeof(c));  // prints-1
        printf("short size : %d \n", sizeof(s)); // prints-2
        printf("float size : %d \n", sizeof(f));  // prints-4
        return 0;
}
```

Program to find the sizes of data types directly:

```
#include <stdio.h>
int main(){
        printf("char size : %d \n", sizeof(char));
        printf("short size : %d \n", sizeof(short));
        return 0;
}
```

Finding the size of expression: In expression, which type occupies highest memory will be the output.

short + long → long

float + double → double

```
#include <stdio.h>
int main(){
        char c;
        short s;
        printf("char + short : %d \n", sizeof(c+s));
        return 0;
}
```

ASCII(American Standards Code of Information Interchange): ASCII represents all the characters of language with constant integer values.

A-65	a-97	0-48	#-35
B-66	b-98	1-49	$-36
..	..	..	..
..	..	..	..
Z-90	z-122	9-57	..

Operators in C

Operator: It is a symbol that performs operation on data.

Assignment operator: This operator is used to store value into variable.

Syntax:	**Example:**
variable = value;	a = 10 a = b = 10;

Example codes on Assignment operator:

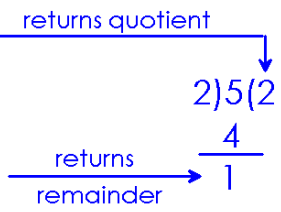
```
#include<stdio.h>
int main(){
        short a, b, c, d;
        a = 10;
        b = a;
        c = a+b;
        d = sizeof(c);
        printf("%d,%d,%d,%d \n", a, b, c, d);
        return 0;
}
```

```
#include<stdio.h>
int main()
{
        short a, b, c, d;
        a = 10;
        b = c = a;
        d = a+b+c;
        printf("d val : %d \n", d);
        return 0;
}
```

Arithmetic operators: performs all arithmetic operations on data.

- Operators are +, -, *, / , %
- Division(/) operator returns the quotient after division.
- Mod(%) operator returns remainder after division.

```
#include<stdio.h>
int main()
{
    int a=5, b=2;
    printf("%d \n", a/b);
    printf("%d \n", a%b);
    return 0;
}
```



We cannot apply Mod(%) operation on decimal data:

```
#include <stdio.h>
int main(){
        float a=5, b=2;
        printf("Quotient : %f \n", a/b);  // prints 2.500000
        printf("Remainder : %f \n", a%b);  // Error:
        return 0;
}
```

We can specify the length of digits while displaying decimal value:

```
#include<stdio.h>
int main(){
        float a=5, b=2;
        printf("%.2f \n", a/b);  // prints 2.50
        return 0;
}
```

Operators Priority: We need to consider the priority of operators if the expression has more than one operator. Arithmetic operators follow BODMAS rule.

Priority	Operator
()	First. If nested then inner most is first
*, / and %	Next to (). If several, from left to right
+, -	Next to *, / , %. If several, from left to right

Examples expressions with evaluations:

5+3-2 8-2 6	5*3%2 15%2 1	5+3*2 5+6 11	(5+3)*2 8*2 16

Basic programs using Arithmetic Operators

Program to add 2 numbers:

```
int main() {
        int a, b, c;
        printf("Enter two numbers : \n");
        scanf("%d%d", &a, &b);
        c = a+b;
        printf("Sum is : %d \n", c);
}
```

Program to find the sum and average of 3 numbers:

```
int main(){
        int a, b, c, sum, average;
        printf("Enter two numbers : \n");
        scanf("%d%d%d", &a, &b, &c);
        sum = a+b+c;
        average = sum/3;
        printf("Sum is : %d \n", sum);
        printf("Average is : %d \n", average);
}
```

Program to find the sum of First N numbers:

```
int main() {
        int n, sum;
        printf("Enter n value : ");
        scanf("%d", &n);
        sum = n*(n+1)/2;
        printf("Sum of first %d numbers is : %d \n", n, sum);
        return 0;
}
```

Program to swap two numbers:

```
int main(){
        int a, b, c;
        printf("Enter 2 numbers : \n");
        scanf("%d%d", &a, &b);
        printf("Before swap : %d \t %d \n", a, b);
        c = a;
        a = b;
        b = c;
        printf("After swap : %d \t %d \n", a, b);
        return 0;
}
```

Program to display the last digit of given number:

```
int main(){
        int n, last;
        printf("Enter number : ");
        scanf("%d", &n);
        last = n%10;
        printf("Last Digit is : %d \n", last);
}
```

Program to remove the last digit of given number:

```
int main()
{
        int n;
        printf("Enter number : ");
        scanf("%d", &n);
        n = n/10;
        printf("Removed last digit : %d \n", n);
}
```

Program to swap 2 numbers without using 3rd variable:

```
#include <stdio.h>
int main(){
        int a, b;
        printf("Enter 2 numbers : \n");
        scanf("%d%d", &a, &b);
        printf("Before swap : %d \t %d \n", a, b);
        a = a+b;
        b = a-b;
        a = a-b;
        printf("After swap : %d \t %d \n", a, b);
        return 0;
}
```

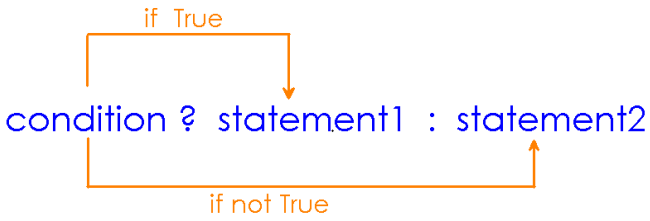
Calculate total amount to pay based on quantity of fruits purchased:

```
#include <stdio.h>
int main(){
        float quantity, price, amount;
        printf("Enter quantity of fruits : ");
        scanf("%f", &quantity);
        printf("Enter price per dozen : ");
        scanf("%f", &price);
        price = (quantity/12) * price;
        printf("Total amount to pay : %f \n", price);
        return 0;
}
```

Program to display total salary for given basic salary:

```
#include <stdio.h>
int main(){
        float basic, ta, da, hra, total;
        printf("Enter basic salary : ");
        scanf("%f", &basic);
        ta = 0.2*basic;
        da = 0.15*basic;
        hra = 0.25*basic;
        total = basic + ta + da + hra;
        printf("Total salary : %f \n", total);
        return 0;
}
```

Conditional operator: This operator validates the condition and execution corresponding statement. It is also called Ternary operator.



Program to display Biggest of 2 numbers:

```
#include <stdio.h>
int main(){
        int a, b;
        printf("Enter 2 numbers : \n");
        scanf("%d%d", &a, &b);
        a>b ? printf("A is big \n") : printf("B is big \n");
        return 0;
}
```

Program to check the given number is Even or not:

```
#include <stdio.h>
int main(){
        int n;
        printf("Enter number : \n");
        scanf("%d", &n);
        n%2==0 ? printf("Even \n") : printf("Not even \n");
        return 0;
}
```

We can define conditional statement inside another conditional statement:
Program to check the biggest of 3 numbers:

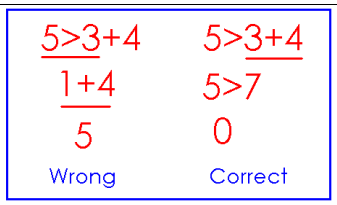
```
#include <stdio.h>
int main(){
        int a, b, c, big;
        printf("Enter 3 numbers : \n");
        scanf("%d%d%d", &a, &b, &c);
        big = a>b && a>c ? a : b>c ? b : c;
        printf("Big one is : %d \n", big);
        return 0;
}
```

Relational operators: These operator returns true(1) or false(0) by validating the relation among the data. Operators are >, <, >=, <=, ==, !=

```
#include<stdio.h>
int main()
{
        printf("5>3 : %d \n", 5>3);
        printf("5<3 : %d \n", 5<3);
        printf("5==3 : %d \n", 5==3);
        printf("5!=3 : %d \n", 5!=3);
        return 0;
}
```

In expression, arithmetic operators evaluates before relational operators:

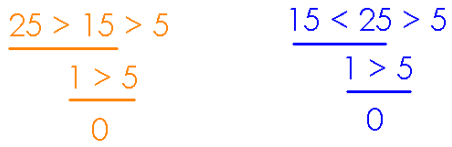
```
#include<stdio.h>
int main()
{
        printf("5>3+4 : %d \n", 5>3+4);
        return 0;
}
```



Note: All relational operators having same priority. Hence these operators execute from left to right

```
#include<stdio.h>
int main()
{
        int a=25, b=15, c=5, d, e;
        d = a>b>c;
        e = b<a>c;
        printf("%d, %d \n", d, e);
        return 0;
}
```

Explanation:




```
#include<stdio.h>
int main()
{
        int a=0, b=0, c=0, d;
        d = a>b==c;
        printf("Output : %d\n", d);
        return 0;
}
```

Explain here:

Modify operators:

- Modify operators increase or decrease the value of variable by 1
- Operators are ++, --
- Modify operators also called unary operators (operate on single operand)

Modify Operators			
Increment Operators		Decrements Operators	
Pre-Increment	Post-Decrement	Pre-Decrement	Pos-Decrement

Example:
int a = 5;
print(++a); // 6
print(a); //6
	Example:
	int a = 5;
	print(a++); // 5
	print(a); //6
		Example:
		int a = 5;
		print(--a); // 4
		print(a); //4
			Example:
			int a = 5;
			print(a--); // 5
			print(a); //4

Expressions with modify operators evaluate as follows:

1. Pre-Increment and Pre-Decrement
2. Substitute values in expression
3. Evaluation of expression
4. Assignment
5. Post-Increment and Post-Decrement

Code snippets to understand the execution flow of Modify operators:

```
int main(){
        int a=5;
        printf("%d\n", a--);
        printf("%d\n", a);
        return 0;
}
```

```
int main(){
        int a=5, b;
        b = ++a;
        printf("a=%d\n", a);
        printf("b=%d\n", b);
        return 0;
}
```

```
int main(){
        int a=5, b;
        b = a++;
        printf("a=%d\n", a);
        printf("b=%d\n", b);
        return 0;
}
```

```
int main(){
        int a=5, b;
        b = ++a + a++;
        printf("a=%d\n", a);
        printf("b=%d\n", b);}
```

```
int main(){
        int a=5, b;
        b = a++ + a++;
        printf("a=%d\n", a);
        printf("b=%d\n", b);}
```

```
int main(){
        int a=5, b;
        b = --a + --a;
        printf("a=%d\n", a);
        printf("b=%d\n", b);}
```

```
int main(){
        int a=5, b;
        b = --a + a--;
        printf("a=%d\n", a);
        printf("b=%d\n", b);
        return 0;
}
```

```
int main(){
        int a=5, b;
        b = a-- + a--;
        printf("a=%d\n", a);
        printf("b=%d\n", b);
        return 0;
}
```

```
int main(){
        int a=5, b;
b = a++ + ++a + a--;
        printf("a=%d\n", a);
        printf("b=%d\n", b);
        return 0;
}
```

Logical Operators:

- These operators return true (1) or false (0) by evaluating more than one expression.
- Operators are,
    - Logical-AND (&&)
    - Logical-OR (||)
    - Logical-NOT (!)
- Following truth table represents the results of Logical operators' evaluation

A	B	A&&B	A||B	!A	!B
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

Compound assignment Operators:

- Compound assignment operators reduce the size of modify expression.
- Operators are +=, -=, *=, &&=, >>=

A+=10 → A=A+10 A+=B → A=A+B X*=Y → X=X*Y	Balance += Deposit → Balance = Balance + Deposit Balance -= Withdraw → Balance = Balance - Withdraw

Bitwise operators:

- These operators convert the input into binary data and perform operations.
- Operators are,
    a. Bitwise AND (&)
    b. Bitwise OR (|)
    c. Bitwise XOR (^)
- The following table explains the how Bitwise operations returns the results.

Truth table:

A	B	A&B	A|B	A^B
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Note: After performing bitwise operations, results will be displayed in Decimal format only.

Bitwise – AND operator:

```
#include <stdio.h>
int main()
{
        int a = 12, b = 25;
        printf("Output = %d", a & b);
        return 0;
}
```

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

   00001100
& 00011001
   ----------
   00001000  = 8 (In decimal)
```

Bitwise – OR operator:

```
#include <stdio.h>
int main()
{
        int a = 12, b = 25;
        printf("Output = %d", a | b);
        return 0;
}
```

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

  00001100
| 00011001
  ----------
  00011101  = 29 (In decimal)
```

Bitwise – XOR operator:

```
#include <stdio.h>
int main()
{
        int a = 12, b = 25;
        printf("Output = %d", a ^ b);
        return 0;
}
```

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

  00001100
^ 00011001
  -----------
  00010101  = 21 (In decimal)
```

Shift Operators:

- These operators are used to shift binary bits either to Left or Right
- The operators are,
    - Left Shift (<<)
    - Right Shift(>>)

Theoretical formulas:

- For Right Shift: $n/2^s$
- For Left Shift: $n*2^s$
    - Where 'n' is the data and 's' is the number of bits to shift

Right Shift operator(>>):

int main(){
int x=32;
int y=x>>2;
printf("y val : %d\n", y);
return 0;
}
	
	$32/2^2$
	32/4
	8

Left Shift operator(<<):

int main(){
int x=2;
int y=x<<3;
printf("y val : %d\n", y);
return 0;
}
	
	$2*2^3$
	2*8
	16

Type Casting: Conversion of data from one type to another type.

- If we perform division operations on integer types, the result will be an integer.
- We need to convert the integer type into float type before division operations.

int main()
{
int a=5, b=2;
float res=a/b;
printf("Res : %f\n",res);
return 0;
}
	int main()
	{
	int a=5, b=2;
	float res=(float)a/b;
	printf("Res : %f\n",res);
	return 0;
	}
		int main()
		{
		int a=5, b=2;
		float res=(float)(a/b);
		printf("Res : %f\n",res);
		return 0;
		}

Output: 2.000000	**Output:** 2.500000	**Output:** 2.000000

Module-2

(Flow Of Control and Arrays)

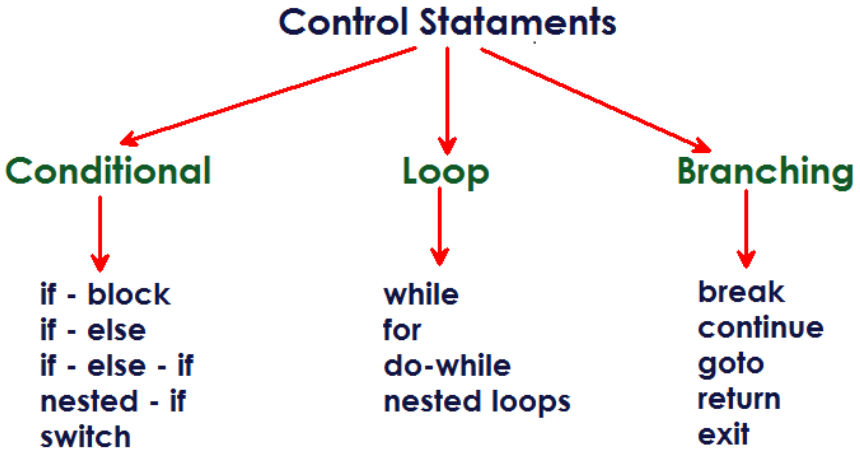
Control Statements in C

Sequential statements:

- Statement is a line of code.
- Sequential Statements execute one by one from top-down approach.

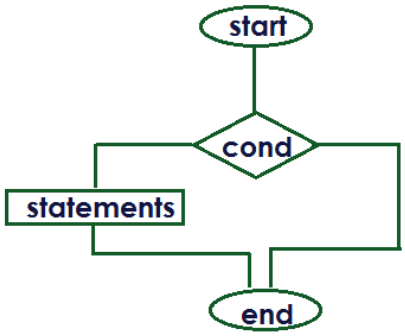
```
#include<stdio.h>
int main()
{
        int a, b, c;
        a = 10;
        b = a;
        c = a*b;
        printf("c value : %d \n", c);
        return 0;
}
```

Control statements: Statements execute randomly and repeatedly based on conditions.



If block: execute a block of instructions only when condition is valid.

Syntax	Flow Chart

if(condition)
{
statements;
}
	

Program to give 15% discount to Customer if the bill amount is >= 5000

```
#include<stdio.h>
int main(){
        float bill=0, discount=0;
        printf("Enter Bill amount : ");
        scanf("%f", &bill);
        if(bill>=5000)
        {
                discount = 0.15*bill;
                bill = bill-discount;
        }
        printf("Final bill to pay : %f \n", bill);
        printf("Your discount is : %f \n", discount);
        return 0;
}
```

Output:

```
Enter Bill amount : 6000.0
Final bill to pay : 5100.0
Your discount is :  900.0
```

Program to give 20% bonus to Employees having more than 5 years' experience

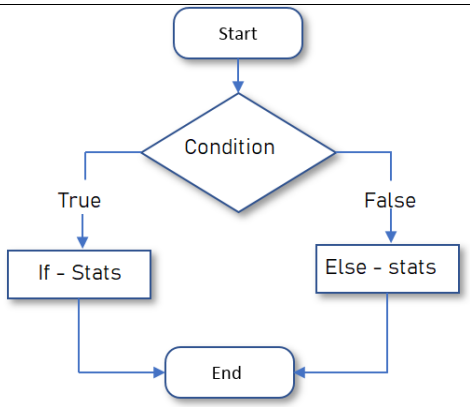
```
#include<stdio.h>
int main(){
        int exp=0;
        float salary=0, bonus=0;
        printf("Enter Employee salary : ");
        scanf("%f", &salary);
        printf("Enter Expierience : ");
        scanf("%d", &exp);
        if(exp>5)
        {
                bonus = 0.20 * salary;
                salary = salary + bonus;
        }
        printf("Your salary : %f \n", salary);
        printf("Bonus : %f \n", bonus);
        return 0;
}
```

Output:

```
Enter Employee salary : 25000
Enter Expierience : 6
Your salary : 30000.000000
Bonus : 5000.000000
```

if - else block: Else block can be used to execute an optional logic if the given condition has failed.

Syntax	Flow Chart

if (condition){
If – Stats;
}
else{
Else – Stats;
}
	

Program to check the input number is Positive or Not:

Positive Number: The number greater than 0 is called positive number.

```
#include<stdio.h>
int main()
{
        int num;
        printf("Enter number : ");
        scanf("%d", &num);
        if(num>0)
                printf("Positive number \n");
        else
                printf("Not Positive number \n");
        return 0;
}
```

Program to check the Person eligible for Vote or Not:

```
#include<stdio.h>
int main(){
        int age;
        printf("Enter age : ");
        scanf("%d", &age);
        if(age>=18)
                printf("You are eligible for Vote \n");
        else
                printf("You are not eligible for Vote \n");
        return 0;
}
```

Program to find the biggest of 2 numbers:

```
#include<stdio.h>
int main(){
	int a, b;
	printf("Enter a value : ");
	scanf("%d", &a);
	printf("Enter b value : ");
	scanf("%d", &b);
	if(a>b)
		printf("a is big \n");
	else
		printf("b is big \n");
	return 0;
}
```

Program to check the number is Even or not:

Even Number: The number which is divisible by 2 is called Even number.

```
#include<stdio.h>
int main(){
	int n;
	printf("Enter number : ");
	scanf("%d", &n);
	if(n%2==0)
		printf("%d is Even \n", n);
	else
		printf("%d is not Even \n", n);
	return 0;
}
```

Program to check the number divisible by both 3 and 5 or not:

We use logical AND(&&) operator to check more than 1 condition is True.

```
#include<stdio.h>
int main(){
	int n;
	printf("Enter number : ");
	scanf("%d", &n);
	if(n%3==0 && n%5==0)
		printf("%d divisible by both 3 and 5 \n", n);
	else
		printf("%d is not divisible by both 3 and 5 \n", n);
	return 0;
}
```

Program to check the character is Vowel or not:

Vowels are a, e, i, o, u.

```
#include<stdio.h>
int main()
{
        char ch;
        printf("Enter character : ");
        scanf("%c", &ch);
        if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u')
                printf("Given character is Vowel \n");
        else
                printf("Given character is not Vowel \n");
        return 0;
}
```

Program to check the character is Upper case alphabet or not:

```
int main()
{
        char ch;
        printf("Enter character : ");
        scanf("%c", &ch);
        if(ch>='A' && ch<='Z')
                printf("Upper case Alphabet character \n");
        else
                printf("Not an Upper case Alphabet character \n");
        return 0;
}
```

Program to check the number between 30 and 50 or not:

```
int main()
{
        int n;
        printf("Enter number : ");
        scanf("%d", &n);
        if(n>=30 && n<=50)
                printf("Number between 30 and 50\n");
        else
                printf("Not between 30 and 50 \n");
        return 0;
}
```

Program to check the login details are correct or not:

```
#include<stdio.h>
int main(){
	char user[20];
	int pwd;
	printf("Enter user name : ");
	scanf("%s", user);
	printf("Enter password : ");
	scanf("%d", &pwd);
	if(strcmp(user, "amar")==0 && pwd==1234)
		printf("Login success\n");
	else
		printf("Login failed\n");
	return 0;
}
```

Program to check the character is alphabet or not:

```
#include<stdio.h>
int main(){
	char ch;
	printf("Enter character : ");
	scanf("%c", &ch);
	if((ch>='A' && ch<='Z') || (ch>='a' && ch<='z'))
		printf("Alphabet character \n");
	else
		printf("Not an Alphabet character \n");
	return 0;
}
```

Program to check the character is Symbol or not:

```
#include<stdio.h>
int main(){
	char ch;
	printf("Enter character : ");
	scanf("%c", &ch);
	if(!((ch>='A' && ch<='Z') || (ch>='a' && ch<='z') || (ch>='0'&&ch<='9')))
		printf("It's a symbol\n");
	else
		printf("It's not a symbol\n");
	return 0;
}
```

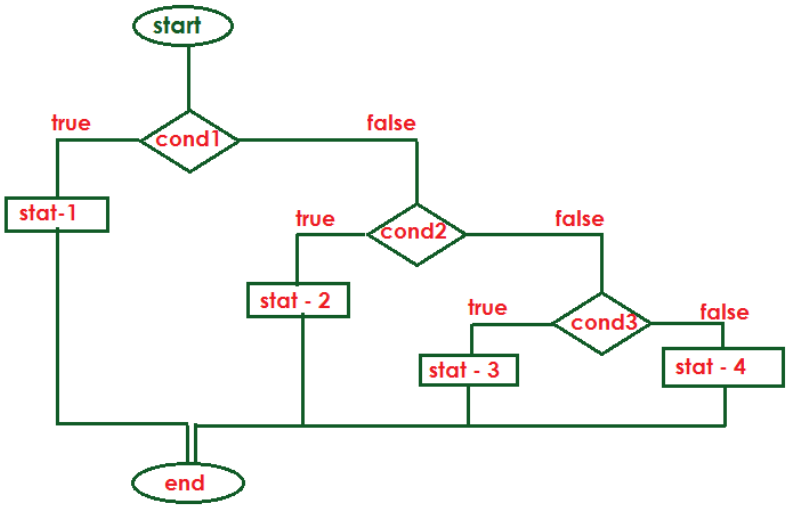
If-else-if ladder:

- It is allowed to defined multiple if blocks sequentially.
- It executes only one block among the multiple blocks defined.

Syntax :

```
if(cond1)
{
    statements1;
}
else if(cond2)
{
    statements2;
}
else if(cond3)
{
    statements3;
}
else
{
    statements4;
}
```

Flow Chart:



Program to check the number is positive or negative or zero:

```
#include<stdio.h>
int main(){
        int n;
        printf("Enter a number : ");
        scanf("%d", &n);
        if(n>0)
                printf("Positive number\n");
        else if(n<0)
                printf("Negative number\n");
        else
                printf("zero");
        return 0;
}
```

Program to check the Character is Alphabet or Digit or Symbol:

```
#include<stdio.h>
int main(){
        char ch;
        printf("Enter character : ");
        scanf("%c", &ch);
        if(ch>='A' && ch<='Z')
                printf("Upper case alphabet\n");
        else if(ch>='a' && ch<='z')
                printf("Lower case alphabet\n");
```

```
        else if(ch>='0' && ch<='9')
                printf("Digit\n");
        else
                printf("Symbol \n");
        return 0;
}
```

Program to find the biggest of 3 numbers:

```
#include<stdio.h>
int main(){
        int a, b, c;
        printf("Enter 3 numbers : ");
        scanf("%d%d%d", &a, &b, &c);
        if(a>b && a>c)
                printf("A is big\n");
        else if(b>c)
                printf("B is big\n");
        else
                printf("C is big\n");
        return 0;
}
```

Program to check the input year is Leap or not:

- If the year divisible by 400 is called Leap year
- If the year divisible by 4 is called Leap year.
- If the year divisible by 100 is not leap year (except 400 multiples)

```
#include<stdio.h>
int main()
{
        int n;
        printf("Enter year : ");
        scanf("%d", &n);
        if(n%400==0)
                printf("Lep year \n");
        else if(n%4==0 && n%100!=0)
                printf("Lep year \n");
        else
                printf("Not leap year \n");

        return 0;
}
```

Program to check profit or loss:

```
#include<stdio.h>
int main()
{
	int originalCost, sellingPrice;
	printf("Enter Original Cost : ");
	scanf("%d", &originalCost);
	printf("Enter Selling Price : ");
	scanf("%d", &sellingPrice);
	if(originalCost < sellingPrice)
		printf("Profit\n");
	else if(originalCost > sellingPrice)
		printf("Loss\n");
	else
		printf("No Loss No Profit \n");
	return 0;
}
```

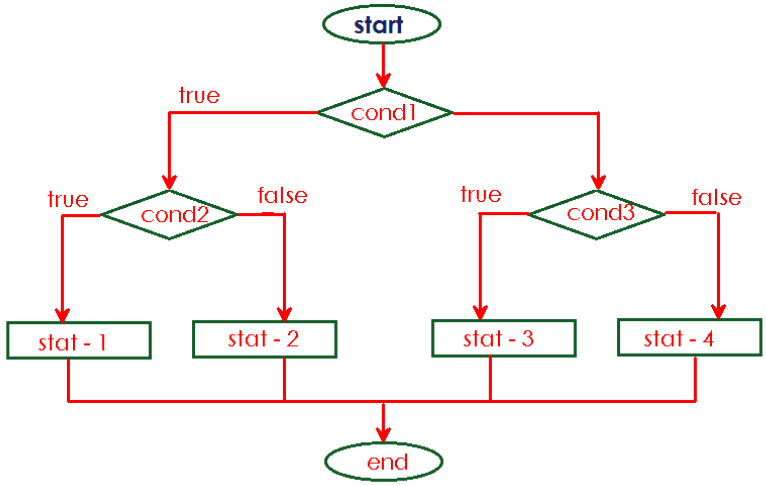
C program to calculate electricity bill based on units

0-100 units -> 0.8 per unit;
101-200 units -> 1.0 per unit;
201-300 units -> 1.2 per unit;
Above 300 -> 1.5 per unit;

```
#include<stdio.h>
int main()
{
	int units;
	float bill;
	printf("Enter units consumed : ");
	scanf("%d", &units);
	if(units>0 && units<=100)
		bill = units*0.8;
	else if(units>100 && units<=200)
		bill = (100*0.8) + (units-100)*1.0;
	else if(units>200 && units<=300)
		bill = (100*0.8) + (100*1.0) + (units-200)*1.2;
	else
		bill = (100*0.8) + (100*1.0) + (100*1.2) + (units-300)*1.5;
	printf("Bill to pay : %f \n" , bill);
	return 0;
}
```

Nested if block: Defining a block inside another block. Inner if block condition evaluates only if the outer condition is valid. It is not recommended to write many levels to avoid complexity.

```
if(cond1)
{
    if(cond2)
        stat-1;
    else
        stat-2;
}
else
{
    if(cond3)
        stat-3;
    else
        stat-4;
}
```



Check the number is Even or not only if the number is positive:

```
int main(){
        int n;
        printf("Enter number : ");
        scanf("%d", &n);
        if(n>0)
        {
                if(n%2==0)
                        printf("Even number\n");
                else
                        printf("Not even number\n");
        }
        else
                printf("Not a positive number\n");
        return 0;
}
```

Check the person is eligible to donate blood or not:

```
int main(){
        int age, weight;
        char gender;
        printf("Enter gender : ");
        scanf("%c", &gender);
        printf("Enter age : ");
        scanf("%d", &age);
        printf("Enter weight : ");
```

```
        scanf("%d", &weight);
        if(gender=='M' || gender=='m')
        {
                if(age>=18 && age<=60)
                {
                        if(weight>=50)
                                printf("You can donate blood \n");
                        else
                                printf("Under weight \n");
                }
                else
                        printf("Invalid age \n");
        }
        else
                printf("Invalid Gender\n");
        return 0;
}
```

Print student grade only if the student passed in all subjects:

Minimum pass marks must be >=35

Display the Grade only if the student passed in all subjects.

If the average >= 60 then print “A-Grade”

If the average >= 50 then print “B-Grade”

If the average >= 40 then print “C-Grade”

```
int main(){
        int m1, m2, m3;
        printf("Enter 3 marks : ");
        scanf("%d%d%d", &m1, &m2, &m3);
        if(m1>=35 && m2>=35 && m3>=35)
        {
                int avg = (m1+m2+m3)/3;
                if(avg>=60)
                        printf("Grade-A\n");
                else if(avg>=50)
                        printf("Grade-B\n");
                else
                        printf("Grade-C\n");
        }
        else
                printf("Student failed\n");
        return 0;
}
```

C program to check the triangle is Equilateral or Isosceles or Scalene

Check the type of triangle only if it is valid (sum of 3 angles equals to 180)
If all angles are equal then it is called “Equilateral”
If any 2 angles are equal then it is called “Isosceles”
If 3 angles are unique then it is called “scalene”

```
int main(){
	int a, b, c;
	printf("Enter angles of triangle : ");
	scanf("%d%d%d", &a, &b, &c);
	if(a+b+c==180){
		if(a==b && b==c)
			printf("Equilateral triangle \n");
		else if(a==b || b==c || a==c)
			printf("Isosceles triangle \n");
		else
			printf("Scalene triangle \n");
	}
	else
		printf("Invalid angles given \n");
}
```

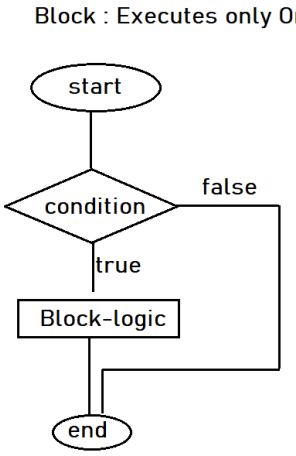
Read month number and display days:

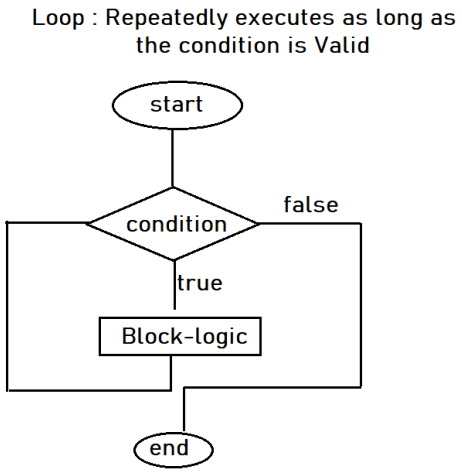
If the month is invalid (not from 1 to 12) then display “Invalid month”
Month-2 has 28 or 29 days
Months-4,6,9,11 has 30 days
Remaining months has 31 days

```
int main(){
	int month;
	printf("Enter month : ");
	scanf("%d", &month);
	if(month>=1 && month<=12){
		if(month==2)
			printf("28 days \n");
		else if(month==4 || month==6 || month==9 || month==11)
			printf("30 days \n");
		else
			printf("31 days \n");
	}
	else
		printf("Invalid month given \n");
}
```

Introduction to Loops

Loop: A Block of instructions execute repeatedly as long the condition is valid.





Note: Block executes only once whereas Loop executes until condition become False

Java Supports 3 types of Loops:

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

For Loop: We use for loop only when we know the number of repetitions. For example,

- Print 1 to 10 numbers
- Print Array elements
- Print Multiplication table
- Print String character by character in reverse order

While loop: We use while loop when we don't know the number of repetitions.

- Display contents of File
- Display records of Database table

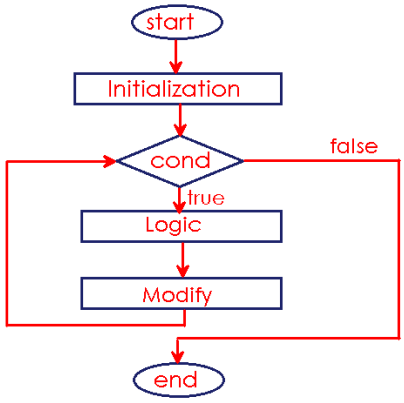
Do while Loop: Execute a block at least once and repeat based on the condition.

- ATM transaction: When we swipe the ATM card, it starts the first transaction. Once the first transaction has been completed, it asks the customer to continue with another transaction and quit.

For Loop

for loop: Execute a block of instructions repeatedly as long as the condition is valid. We use for loop only when we know the number of iterations to do.

Syntax	Flow Chart

for(init ; condition ; modify)
{
statements;
}
 	
for(start ; stop ; step)
{
statements;
}
	

C Program to Print 1 to 10 Numbers:

```
int main() {
        int i;
        for(i=1 ; i<=10 ; i++){
                printf("%d \n", i);
        }
        return 0;
}
```

C Program to Print Numbers from 10 to 1:

```
int main(){
        int i;
        for(i=10 ; i>=1 ; i--){
                printf("%d \n", i);
        }
        return 0;
}
```

C Program to Print 1 to N:

```
int main(){
        int n, i;
        printf("Enter n value : ");
        scanf("%d", &n);
        for(i=1 ; i<=n ; i++){
                printf("%d \n", i);
        }
        return 0;
}
```

C Program to display A to Z

```
#include<stdio.h>
int main(){
        char ch;
        for(ch='A' ; ch<='Z' ; ch++){
                printf("%c", ch);
        }
        return 0;
}
```

C Program to display ASCII values from A to Z

```
#include<stdio.h>
int main(){
        char ch;
        for(ch='A' ; ch<='Z' ; ch++){
                printf("%c : %d \n", ch, ch);
        }
        return 0;
}
```

C Program to display ASCII character set

```
#include<stdio.h>
int main(){
        int x;
        for(x=0 ; x<256 ; x++){
                printf("%d : %c \n", x, x);
        }
        return 0;
}
```

C program to display Multiplication table for given number

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

C Program to display Sum of First N numbers

```
int main(){
        int n, i, sum=0;
        printf("Enter n value : ");
        scanf("%d", &n);
        for(i=1 ; i<=n ; i++){
                sum = sum+i;
        }
        printf("Sum of first %d numbers is : %d \n", n, sum);
}
```

C Program to display Factorial of Given Number

```
int main(){
        int n, i, fact=1;
        printf("Enter n value : ");
        scanf("%d", &n);
        for(i=1 ; i<=n ; i++){
                fact = fact*i;
        }
        printf("Factorial of %d is : %d \n", n, fact);
        return 0;
}
```

C Program to display Even numbers from 1 to 10

```
int main(){
        int i;
        for(i=1 ; i<=10 ; i++){
                if(i%2==0)
                        printf("%d is even \n", i);
        }
        return 0;
}
```

C Program to display sum of even numbers from 1 to 10

```
int main(){
        int i, sum=0;
        for(i=1 ; i<=10 ; i++){
                if(i%2==0)
                        sum = sum+i;
        }
        printf("Sum is : %d \n", sum);
}
```

C program to display factors of given number

```
int main(){
        int i, n;
        printf("Enter n value : ");
        scanf("%d", &n);
        for(i=1 ; i<=n ; i++){
                if(n%i==0)
                        printf("%d is a factor for %d \n", i, n);
        }
        return 0;
}
```

C program to count the factors of given number

```
int main(){
        int i, n, count=0;
        printf("Enter n value : ");
        scanf("%d", &n);
        for(i=1 ; i<=n ; i++){
                if(n%i==0)
                        count++;
        }
        printf("Number of Factors for %d is %d \n", n, count);
        return 0;
}
```

C program to check the input number is prime or not

Prime Number: The Number which is having 2 Factors

```
int main(){
        int i, n, count=0;
        printf("Enter n value : ");
        scanf("%d", &n);
        for(i=1 ; i<=n ; i++){
                if(n%i==0)
                        count++;
        }
        if(count==2)
                printf("%d is Prime number \n", n);
        else
                printf("%d is not Prime number \n", n);
        return 0;
}
```

C program to find the sum of factors for input number

```
int main(){
        int i, n, sum=0;
        printf("Enter n value : ");
        scanf("%d", &n);
        for(i=1 ; i<=n ; i++){
                if(n%i==0)
                        sum=sum+i;
        }
        printf("Sum of factors of %d is %d \n", n, sum);
}
```

Perfect Number Program: Sum of factors except itself is equals to the same number.

```
int main(){
        int i, n, sum=0;
        printf("Enter n value : ");
        scanf("%d", &n);
        for(i=1 ; i<n ; i++){
                if(n%i==0)
                        sum=sum+i;
        }
        if(n==sum)
                printf("%d is Perfect number \n", n);
        else
                printf("%d is not a perfect number \n", n);
}
```

Fibonacci series: The series in which sum of 2 consecutive numbers will be the next number

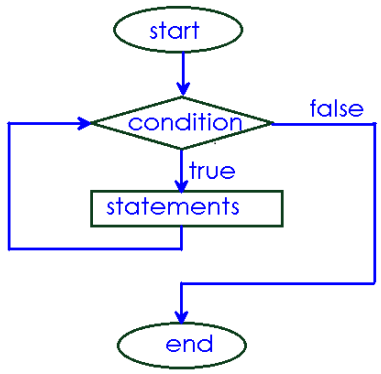
Series: 0, 1, 1, 2, 3, 5, 8, 13, 21,…

```
int main(){
        int i, n, a=0, b=1, c;
        printf("Enter limit : ");
        scanf("%d", &n);
        for(i=1 ; i<=n ; i++){
                printf("%d  ", a);
                c = a+b ;
                a = b ;
                b = c ;
        }
}
```

While Loop

While loop: Execute a block of instructions repeatedly until the condition is false. We use while loop only when don't know the number of iterations to do.

Syntax	Flow Chart

while(condition)
{
statements;
}
	

Program to display digits of number in reverse order:

```
int main(){
        int n;
        printf("Enter num : ");
        scanf("%d", &n);
        while(n!=0)
        {
                printf("%d \n", n%10);
                n=n/10;
        }
        return 0;
}
```

Program to count the digits in given number:

```
int main(){
        int n, count=0;
        printf("Enter num : ");
        scanf("%d", &n);
        while(n!=0)
        {
                n=n/10;
                count++;
        }
        printf("Digits count : %d \n", count);
        return 0;
}
```

Program to display sum of the digits in the given number:

```
int main(){
        int n, sum=0;
        printf("Enter num : ");
        scanf("%d", &n);
        while(n!=0){
                sum = sum + n%10;
                n = n/10;
        }
        printf("Digits sum : %d \n", sum);
        return 0;
}
```

Program to find the sum of even digits in the given number:

```
int main(){
        int n, r, sum=0;
        printf("Enter num : ");
        scanf("%d", &n);
        while(n>0){
                r = n%10;
                if(r%2==0){
                        sum = sum+r;
                }
                n = n/10;
        }
        printf("Sum of even digits : %d \n", sum);
}
```

Program to count number of zeros in the given number:

```
int main (){
        int n, count=0, d;
        printf("Enter a number : ");
        scanf("%d", &n);
        while(n>0){
                d = n%10;
                if(d==0)
                        count++;
                n=n/10;
        }
        printf("Number of Zeros : %d \n", count);
}
```

C Program to display the first digit of given number:

```
int main(){
        int n;
        printf("Enter num : ");
        scanf("%d", &n);
        while(n>=10){
                n = n/10;
        }
        printf("First Digit : %d \n", n);
        return 0;
}
```

C Program to Find the Sum of First and Last digits of given number:

```
int main(){
        int n, first, last;
        printf("Enter num : ");
        scanf("%d", &n);
        first = n%10;
        n=n/10;
        while(n>=10){
                n = n/10;
        }
        last = n%10;
        printf("Sum of First and Last digits : %d \n", first+last);
        return 0;
}
```

C Program to reverse the given number:

```
int main(){
        int n, rev=0, r;
        printf("Enter num : ");
        scanf("%d", &n);
        while(n>0){
                r = n%10;
                rev = rev*10 + r;
                n = n/10;
        }

        printf("Reverse number is : %d \n", rev);
        return 0;
}
```

Palindrome Number Program in C:

The number become same when we reverse called Palindrome number(121, 1001)

```
int main(){
        int n, rev=0, r, temp;
        printf("Enter num : ");
        scanf("%d", &n);
        temp=n;
        while(n>0){
                r = n%10;
                rev = rev*10 + r;
                n = n/10;
        }
        if(temp==rev)
                printf("Palindrome Number \n");
        else
                printf("Not a palindrome number \n");
}
```

Strong Number Program in C:

Sum of factorials of individual digits equals to the same number is called strong number.

Example: 145 → 1! + 4! + 5! = 145

```
int main(){
        int n, r, fact, sum=0, temp, i;
        printf("Enter num : ");
        scanf("%d", &n);
        temp=n;
        while(n>0){
                r = n%10;
                fact=1;
                for(i=1 ; i<=r ; i++){
                        fact = fact*i;
                }
                sum = sum + fact;
                n = n/10;
        }
        if(temp==sum)
                printf("Strong Number \n");
        else
                printf("Not a strong number \n");
}
```

C Program to display highest digit in the number:

```
int main(){
        int n, r, large=0;
        printf("Enter num : ");
        scanf("%d", &n);
        while(n>0){
                r = n%10;
                if(r>large){
                        large = r;
                }
                n = n/10;
        }
        printf("Largest digit is : %d \n", large);
        return 0;
}
```

C Program to check the 3 digits number is Armstrong number or not:

Example: 153 → 1^3 + 5^3 + 3^3 = 153

```
int main(){
        int n, r, sum=0, temp;
        printf("Enter num : ");
        scanf("%d", &n);
        temp=n;
        while(n>0){
                r = n%10;
                sum = sum + r*r*r;
                n = n/10;
        }
        if(temp==sum)
                printf("Armstrong number \n");
        else
                printf("Not an Armstrong number \n");
        return 0;
}
```

C Program to check the given number is Armstrong number or not:

2-digit number: Sum of squares of individual digits equals to the same number
3-digit number: Sum of cubes of individual digits equals to the same number
4-digit number: Sum of individual digits power 4 equals to the same number
153 = 1^3 + 5^3 + 3^3 = 153
1634 = 1^4 + 6^4 + 3^4 + 4^4 = 1634

```
int main(){
        int n, r, sum=0, temp, c=0, s, i;
        printf("Enter num : ");
        scanf("%d", &n);
        temp=n;
        while(n>0){
                n=n/10;
                c++;
        }
        n=temp;
        while(n>0){
                r = n%10;
                s=1;
                for(i=1 ; i<=c ; i++)
                        s = s*r;
                sum = sum + s;
                n = n/10;
        }
        if(temp==sum)
                printf("Armstrong number \n");
        else
                printf("Not an Armstrong number \n");
}
```

C Program to find Sum of Digits till Single Digit:

```
        9657 -> 9+6+5+7 -> 27 -> 2+7 -> 9
int main(){
        int num, sum, dig;
        printf("Enter num : ");
        scanf("%d", &num);
        printf("%d-> ",num);
        while(num/10!=0){
                sum = 0;
                while(num!=0){
                        dig=num%10;
                        sum+=dig;
                        num/=10;
                }
                printf("%d->", sum);
                num=sum;
        }
}
```

Break and Continue

break: A branching statement that terminates the execution flow of a Loop or Switch case.

```
#include<stdio.h>
int main(){
	int i;
	for(i=1 ; i<=10 ; i++)
	{
		if(i==5)
			break;
		printf("i val : %d \n", i);
	}
	return 0;
}
```

Break statement terminates the flow of infinite loop also:

```
#include<stdio.h>
int main()
{
	while(1)
	{
		printf("Infinite Loop \n");
		break;
	}
	return 0;
}
```

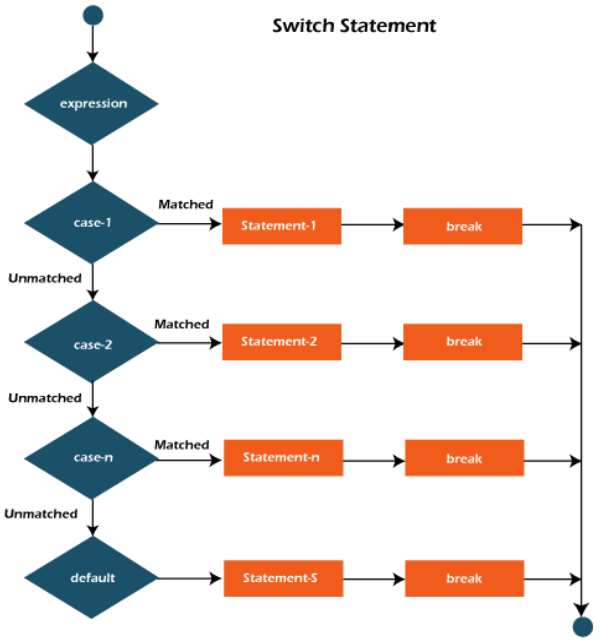
Continue: A branching statement that terminates the current iteration of loop execution.

```
#include<stdio.h>
int main()
{
	int i;
	for(i=1 ; i<=10 ; i++)
	{
		if(i==5)
			continue;
		printf("i val : %d \n", i);
	}
	return 0;
}
```

Switch case

Switch: It is a conditional statement that executes specific set of statements(case) based on given choice. Default case executes if the user entered invalid choice. Case should terminate with break statement.

Syntax	FlowChart

switch(choice)
{
case 1 : Statements ;
break
case 2 : Statements ;
break
.....
case n : Statements ;
break
default: Statements ;
}
	

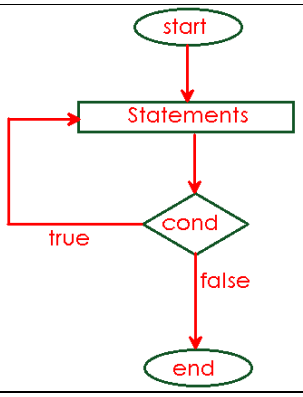

```
#include<stdio.h>
int main(){
        char ch;
        printf("Enter character(r, g, b) : ");
        scanf("%c", &ch);
        switch(ch)
        {
                case 'r' :      printf("Red \n");
                                break;
                case 'g' :      printf("Green \n");
                                break;
                case 'b' :      printf("Blue \n");
                                break;
                default :       printf("Unknown \n");
        }
        return 0;
}
```

Program to perform arithmetic operations based on given choice:

```
#include<stdio.h>
int main(){
	int a, b, c, ch;
	printf("Arithmetic Opeations");
	printf("1.Add \n");
	printf("2.Subtract \n");
	printf("3.Multiply \n");
	printf("4.Divide \n");

	printf("Enter your choice : ");
	scanf("%d",&ch);

	if(ch>=1 && ch<=4)
	{
		printf("Enter 2 numbers : \n");
		scanf("%d%d", &a, &b);
	}

	switch(ch)
	{
		case 1:		c=a+b;
				printf("Add result : %d\n", c);
				break;

		case 2:		c=a-b;
				printf("Subtract result : %d\n", c);
				break;

		case 3:		c=a*b;
				printf("Multiply result : %d\n", c);
				break;

		case 4:		c=a/b;
				printf("Division result : %d\n", c);
				break;

		default:	printf("Invalid choice\n");
	}
}
```

Do-While Loop

do-while: Executes a block at least once and continue iteration until condition is false.

Syntax	Flow Chart
do { statements; } while(condition);	

Check Even numbers until user exits:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int n;
        char ch;
        do{
                printf("\nEnter number : ");
                scanf("%d", &n);

                if(n%2==0)
                        printf("%d is even \n", n);
                else
                        printf("%d is odd \n", n);

                printf("Do you want to check another num(y/n) : ");
                ch = getch();

        }while(ch!='n');
        return 0;
}
```

getch():

- getch() is a pre-defined method belongs to conio.h header file.
- It is used to read character from the console while program is running.

goto: A branching statement used to send the control from one place to another place in the program by defining labels.

Program to display 1 to 10 numbers without loop:

```
#include<stdio.h>
int main(){
	int a=1;
	top:
		printf("a value : %d \n", a);
		++a;
		if(a<=10)
			goto top;
	return 0;
}
```

We can create multiple labels in a single program.

```
#include<stdio.h>
int main(){
	top:
		printf("Top \t");
		goto bottom;
	center:
		printf("Center \t");
	bottom:
		printf("Bottom \n");
		goto top;
	return 0;
}
```

We use goto statement to come out of infinite loop also.

```
#include<stdio.h>
int main(){
	while(1){
		printf("Infinite loop \n");
		goto end;
	}
	end:
		printf("Can break with goto\n");
	return 0;
}
```

Menu Driven Programs

Menu Driven programs are used to perform set of operations continuously until user exits.

Examples: Arithmetic operations, Stack , Queue, Linked List operations etc.

Program to perform Arithmetic operations using (while and if) :

```
int main(){
        int a, b, ch;
        while(1){
                printf("Arithmetic Opeations\n");
                printf("1.Add \n");
                printf("2.Subtract \n");
                printf("3.Multiply \n");
                printf("4.Divide \n");
                printf("5.Quit\n");

                printf("Enter your choice : ");
                scanf("%d",&ch);

                if(ch==1){
                        printf("Enter 2 numbers : \n");
                        scanf("%d%d", &a, &b);
                   printf("Add result : %d\n", a+b);
                }
                else if(ch==2){
                        printf("Enter 2 numbers : \n");
                        scanf("%d%d", &a, &b);
                        printf("Subtract result : %d\n", a-b);
                }
                else if(ch==3){
                        printf("Enter 2 numbers : \n");
                        scanf("%d%d", &a, &b);
                        printf("Multiply result : %d\n", a*b);
                }
                else if(ch==4){
                        printf("Enter 2 numbers : \n");
                        scanf("%d%d", &a, &b);
                        printf("Division result : %d\n", a/b);
                }
                else if(ch==5){
                        printf("End \n");
                        exit(1);
```

```
                }
                else
                        printf("Invalid choice \n");
        }
}
```

Program to perform Arithmetic operations using (while and switch):

```
int main()
{
        int a, b, c, ch;
        while(1){
                printf("Arithmetic Opeations\n");
                printf("1.Add \n");
                printf("2.Subtract \n");
                printf("3.Multiply \n");
                printf("4.Divide \n");
                printf("5.Quit\n");
                printf("Enter your choice : ");
                scanf("%d",&ch);
                if(ch>=1 && ch<=4){
                        printf("Enter 2 numbers : \n");
                        scanf("%d%d", &a, &b);
                }
                switch(ch){
                        case 1:         c=a+b;
                                        printf("Add result : %d\n", c);
                                        break;
                        case 2:         c=a-b;
                                        printf("Subtract result : %d\n", c);
                                        break;
                        case 3:         c=a*b;
                                        printf("Multiply result : %d\n", c);
                                        break;
                        case 4:         c=a/b;
                                        printf("Division result : %d\n", c);
                                        break;
                        case 5:         printf("End\n");
                                        exit(1);
                        default:        printf("Invalid choice\n");
                }
        }
}
```

Range based programs

C Program to print Even number in given Range:

```
int main (){
        int limit, n;
        printf("*****Even Numbers in the Given Range*****\n");
        printf("Enter Limit : ");
        scanf("%d", &limit);
        for(n=1 ; n<=limit ; n++){
                if(n%2==0)
                        printf("%d \t", n);
        }
        return 0;
}
```

C Program to Print Square and Cube values in the given Range:

```
int main (){
        int limit, n;
        printf("*****Square and Cube values in the given Range*****\n");
        printf("Enter Limit : ");
        scanf("%d", &limit);
        for(n=1 ; n<=limit ; n++){
                printf("%d -> Square=%d -> Cube=%d \n", n, n*n, n*n*n);
        }
        return 0;
}
```

Print Multiplication tables in the given range:

```
int main (){
        int limit, n, i;
        printf("*****Multiplication table in given Range*****\n");
        printf("Enter Limit : ");
        scanf("%d", &limit);
        for(n=1 ; n<=limit ; n++){
                printf("Table-%d : \n", n);
                for(i=1 ; i<=10 ; i++){
                        printf("%d x %d = %d \n", n, i, n*i);
                }
        }
        return 0;
}
```

C Program to Print Factorials in Given Range:

```
int main ()
{
        int limit, n, i;
        printf("*****Factorial values in given Range*****\n");
        printf("Enter Limit : ");
        scanf("%d", &limit);
        for(n=1 ; n<=limit ; n++){
                int fact=1;
                for(i=1 ; i<=n ; i++){
                        fact = fact*i;
                }
                printf("Factorial of %d is : %d \n", n, fact);
        }
        return 0;
}
```

C Program to print Factors of each number in the given range:

```
int main ()
{
        int limit, n, i;
        printf("Enter Limit : ");
        scanf("%d", &limit);
        for(n=1 ; n<=limit ; n++){
                printf("Factors of %d is : ", n);
                for(i=1 ; i<=n ; i++){
                        if(n%i==0)
                                printf("%d  ", i);
                }
                printf("\n");
        }
        return 0;
}
```

C Program to Print Prime Numbers in the Given range:

```
int main ()
{
        int limit, n, i;
        printf("Enter Limit : ");
        scanf("%d", &limit);
        for(n=1 ; n<=limit ; n++){
                int count=0;
```

```
                for(i=1 ; i<=n ; i++){
                        if(n%i==0){
                                count++;
                        }
                }
                if(count==2)
                        printf("%d is prime \n", n);
        }
        return 0;
}
```

C Program to print Perfect numbers in the given range:

```
int main ()
{
        int limit, n, i;
        printf("*****Perfect Numbers in the Given range*****\n");
        printf("Enter Limit : ");
        scanf("%d", &limit);
        for(n=1 ; n<=limit ; n++){
                int sum=0;
                for(i=1 ; i<n ; i++){
                        if(n%i==0)
                                sum=sum+i;
                }
                if(n==sum)
                        printf("%d is perfect \n", n);
        }
        return 0;
}
```

C Program to print Perfect numbers in the given range:

```
int main ()
{
        int limit, n, i;
        printf("*****Perfect Numbers in the Given range*****\n");
        printf("Enter Limit : ");
        scanf("%d", &limit);
        for(n=1 ; n<=limit ; n++){
                int sum=0;
                for(i=1 ; i<n ; i++){
                        if(n%i==0)
                                sum=sum+i;
```

```
                }
                if(n==sum)
                        printf("%d is perfect \n", n);
        }
        return 0;
}
```

C Program to reverse the numbers in given range:

```
int main ()
{
        int limit, n, d, x;
        printf("*****Reverse the numbers in the Given range*****\n");
        printf("Enter Limit : ");
        scanf("%d", &limit);
        for(n=1 ; n<=limit ; n++){
                int rev=0;
                x=n;
                while(x>0){
                        d = x%10;
                        rev = rev*10 + d;
                        x = x/10;
                }
                printf("%d -> %d \n", n, rev);
        }
        return 0;
}
```

C Program to display Palindrome numbers in the given range:

```
int main ()
{
        int low, high, n, d, x;
        printf("*****Palindrome numbers in the Given range*****\n");
        printf("Enter Lower Limit : ");
        scanf("%d", &low);
        printf("Enter Upper Limit : ");
        scanf("%d", &high);
        for(n=low ; n<=high ; n++)
        {
                int rev=0;
                x=n;
                while(x>0)
                {
```

```
                d = x%10;
                rev = rev*10 + d;
                x = x/10;
            }
            if(n==rev)
                printf("%d \t", n);
        }
        return 0;
}
```

C Program to Print Strong Numbers in the given range:

```
int main ()
{
        int low, high, n, d, x, i;
        printf("*****Strong numbers in the Given range*****\n");

        printf("Enter Lower Limit : ");
        scanf("%d", &low);

        printf("Enter Upper Limit : ");
        scanf("%d", &high);

        for(n=low ; n<=high ; n++)
        {
            int sum=0;
            x=n;
            while(x>0)
            {
                int fact=1;
                d = x%10;
                for(i=1 ; i<=d ; i++)
                {
                    fact = fact*i;
                }
                sum = sum + fact;
                x = x/10;
            }
            if(n==sum)
                printf("%d \t", n);
        }
}
```

Pattern Programs in C

Basic Patterns:

```
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=1 ; j<=5 ; j++){
                        printf("%d", i);
                }
                printf("\n");
        }
    return 0;
}
```

```
11111
22222
33333
44444
55555
```

```
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=1 ; j<=5 ; j++){
                        printf("%d", j);
                }
                printf("\n");
        }
    return 0;
}
```

```
12345
12345
12345
12345
12345
```

```
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=1 ; j<=5 ; j++){
                        printf("%d", i%2);
                }
                printf("\n");
        }
    return 0;
}
```

```
11111
00000
11111
00000
11111
```

```
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=1 ; j<=5 ; j++){
                        printf("%d", j%2);
                }
                printf("\n");
        }
    return 0;
}
```

```
10101
10101
10101
10101
10101
```

```
int main (){
        int i, j;
        for (i=5 ; i>=1 ; i--){
                for (j=5 ; j>=1 ; j--){
                        printf("%d", i);
                }
                printf("\n");
        }
   return 0;
}
```

```
55555
44444
33333
22222
11111
```

```
int main (){
        int i, j;
        for (i=5 ; i>=1 ; i--){
                for (j=5 ; j>=1 ; j--){
                        printf("%d", j);
                }
                printf("\n");
        }
   return 0;
}
```

```
54321
54321
54321
54321
54321
```

```
int main (){
        char x, y;
        for (x='A' ;  x<='E' ; x++){
                for (y='A' ; y<='E' ; y++){
                        printf("%c", x);
                }
                printf("\n");
        }
   return 0;
}
```

```
AAAAA
BBBBB
CCCCC
DDDDD
EEEEE
```

```
#include<stdio.h>
int main (){
        char x, y;
        for (x='A' ;  x<='E' ; x++){
                for (y='A' ; y<='E' ; y++){
                        printf("%c", y);
                }
                printf("\n");
        }
   return 0;
}
```

```
ABCDE
ABCDE
ABCDE
ABCDE
ABCDE
```

```
#include<stdio.h>
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=1 ; j<=5 ; j++){
                        printf("*");
                }
                printf("\n");
        }
   return 0;
}
```

```
*****
*****
*****
*****
*****
```

```
#include<stdio.h>
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=1 ; j<=5 ; j++){
                        if(i%2==0)
                                printf("*");
                        else
                                printf("#");
                }
                printf("\n");
        }
   return 0;
}
```

```
*****
#####
*****
#####
*****
```

```
#include<stdio.h>
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=1 ; j<=5 ; j++){
                        if(j%2==0)
                                printf("*");
                        else
                                printf("#");
                }
                printf("\n");
        }
   return 0;
}
```

```
#*#*#
#*#*#
#*#*#
#*#*#
#*#*#
```

Half Triangle Patterns:

```
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=1 ; j<=i ; j++){
                        printf("%d", i);
                }
                printf("\n");
        }
    return 0;
}
```

```
1
22
333
4444
55555
```

```
int main (){
        int i, j;
        for (i=5 ; i>=1 ; i--){
                for (j=i ; j>=1 ; j--){
                        printf("%d", i);
                }
                printf("\n");
        }
    return 0;
}
```

```
55555
4444
333
22
1
```

```
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=1 ; j<=i ; j++){
                        printf("%d", j);
                }
                printf("\n");
        }
    return 0;
}
```

```
1
12
123
1234
12345
```

```
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=i ; j>=1 ; j--){
                        printf("%d", j);
                }
                printf("\n");
        }
    return 0;
}
```

```
1
21
321
4321
54321
```

```
int main (){
        int i, j;
        for (i=5 ; i>=1 ; i--){
                for (j=1 ; j<=i ; j++){
                        printf("%d", j);
                }
                printf("\n");
        }
    return 0;
}
```

```
12345
1234
123
12
1
```

```
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=i; j<=5 ; j++){
                        printf("%d", j);
                }
                printf("\n");
        }
    return 0;
}
```

```
12345
2345
345
45
5
```

```
int main (){
        int i, j;
        for (i=5 ; i>=1 ; i--){
                for (j=5; j>=i ; j--){
                        printf("%d", j);
                }
                printf("\n");
        }
    return 0;
}
```

```
5
54
543
5432
54321
```

```
int main (){
        int i, j;
        for (i=5 ; i>=1 ; i--){
                for (j=i; j<=5 ; j++){
                        printf("%d", j);
                }
                printf("\n");
        }
    return 0;
}
```

```
5
45
345
2345
12345
```

```
#include<stdio.h>
int main ()
{
        int i, j, k=1;
        for (i=1 ; i<=5 ; i++){
                for (j=1; j<=i ; j++){
                        printf("%d", k++);
                        if(k>9)
                                k=1;
                }
                printf("\n");
        }
   return 0;
}
```

```
1
23
456
7891
23456
```

```
#include<stdio.h>
int main ()
{
        int i, j, k=1;
        for (int i=5 ; i>=1 ; i--){
                for (j=1; j<=i ; j++){
                        printf("%d", k++);
                        if(k>9)
                                k=1;
                }
                printf("\n");
        }
   return 0;
}
```

```
12345
6789
123
45
6
```

```
int main ()
{
        char i, j;
        for (i='A' ; i<='E' ; i++)
        {
                for (j='A' ; j<=i ; j++)
                {
                        printf("%d", j);
                }
                printf("\n");
        }
   return 0;
}
```

```
A
AB
ABC
ABCD
ABCDE
```

```
int main (){
        char i, j;
        for (i='E' ; i>='A' ; i--){
                for (j=i ; j>='A' ; j--){
                        printf("%d", j);
                }
                printf("\n");
        }
   return 0;
}
```

```
EDCBA
DCBA
CBA
BA
A
```

```
int main (){
        int i, j, k=1;
        for (int i=1 ; i<=5 ; i++){
                for (j=1; j<=i ; j++){
                        printf("%d", k++%2);
                }
                printf("\n");
        }
   return 0;
}
```

```
1
01
010
1010
10101
```

```
int main (){
        int i, j, k=1;
        for (int i=5 ; i>=1 ; i--){
                for (j=1; j<=i ; j++){
                        printf("%d", k++%2);
                }
                printf("\n");
        }
   return 0;
}
```

```
10101
0101
010
10
1
```

```
int main (){
        int i, j;
        for (i=1 ; i<=5 ; i++){
                for (j=1; j<=i ; j++){
                        printf("%d", i%2);
                }
                printf("\n");
        }
   return 0;
}
```

```
1
00
111
0000
11111
```

```
int main (){
        int i, j, k;
        for (i=1 ; i<=5 ; i++){
                for (j=i ; j<5 ; j++){
                        printf(" ");
                }
                for (k=1 ; k<=i ; k++){
                        printf("*");
                }
                printf("\n");
        }
    return 0;
}
```

```
    *
   **
  ***
 ****
*****
```

```
int main (){
        int i, j, k;
        for (i=1 ; i<=5 ; i++){
                for (j=1 ; j<i ; j++){
                        printf(" ");
                }
                for (k=i ; k<=5 ; k++){
                        printf("*");
                }
                printf("\n");
        }
    return 0;
}
```

```
*****
 ****
  ***
   **
    *
```

```
int main (){
        int i, j, k;
        for (i=1 ; i<=5 ; i++){
                for (j=i ; j<5 ; j++){
                        printf(" ");
                }
                for (k=1 ; k<=i ; k++){
                        printf("%d", k);
                }
                printf("\n");
        }
    return 0;
}
```

```
    1
   12
  123
 1234
12345
```

```
int main (){
        int i, j, k;
        for (i=1 ; i<=5 ; i++){
                for (j=i ; j<5 ; j++){
                        printf(" ");
                }
                for (k=i ; k>=1 ; k--){
                        printf("%d", k);
                }
                printf("\n");
        }
    return 0;
}
```

```
    1
   21
  321
 4321
54321
```

```
int main (){
        int i, j, k;
        for (i=5 ; i>=1 ; i--){
                for (j=i ; j<5 ; j++){
                        printf(" ");
                }
                for (k=1 ; k<=i ; k++){
                        printf("%d", k);
                }
                printf("\n");
        }
    return 0;
}
```

```
12345
 1234
  123
   12
    1
```

```
int main (){
        int i, j, k;
        for (i=1 ; i<=5 ; i++){
                for (j=1 ; j<i ; j++){
                        printf(" ");
                }
                for (k=i ; k<=5 ; k++){
                        printf("%d", k);
                }
                printf("\n");
        }
    return 0;
}
```

```
12345
 2345
  345
   45
    5
```

Arrays in C

Primitive type:

- Variables are used to store data.
- Primitive variable can store only one value at a time.
- For example,

```
int main(){
        int a=5;
        a=10;
        a=20;
        printf("a val : %d \n", a);   // prints - 20}
```

Array:

- Array is used to store more than one value but of same type.
- For example, storing 100 students' average marks.

Array creation:

- In array variable declaration, we need to specify the size
- Array variable stores base address of memory block.
- Array elements store in consecutive memory location.

Process Array elements:

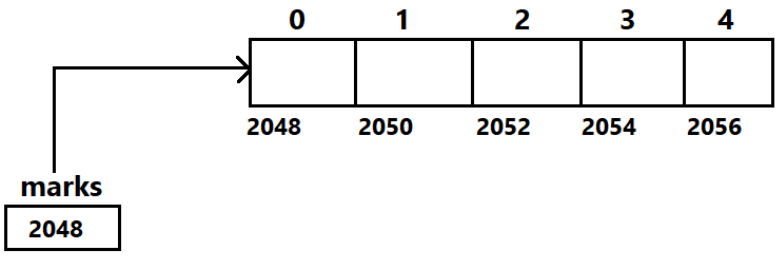
- Array memory locations must be accessed through their index.
- Index values starts from 0 to Size-1

Syntax :

datatype identity[size];

Example :

int marks[5];



How to access array elements?

- Array consists multiple values.
- We can simply use loops to process array locations.

Reading	Printing

int i;
for(i=0 ; i<5 ; i++)
{
scanf("%d", &marks[i]);
}
	int i;
	for(i=0 ; i<5 ; i++)
	{
	printf("%d", marks[i]);
	}

Array as Local variable:

- Declare array variable inside the function.
- If the array variable is local, all locations initialize with garbage values.

```
int main()
{
        int arr[5], i;
        printf("Array elements are : \n");
        for(i=0 ; i<5 ; i++){
                printf("%d \n", arr[i]);
        }
        return 0;
}
```

Array as Global variable:

- Declare array variable outside to all functions.
- Global array variable locations initialize with default values.

```
int arr[5];
int main()
{
        int i;
        printf("Array elements are : \n");
        for(i=0 ; i<5 ; i++){
                printf("%d \n", arr[i]);
        }
        return 0;
}
```

Length of array:

- sizeof() function returns the total size of array.
- We can find the length of array by dividing total size with single element size.

```
int main()
{
        float arr[5];
        printf("Total size of array : %d \n", sizeof(arr));
        printf("Array element size : %d \n", sizeof(arr[0]));
        printf("Length of Array : %d \n", sizeof(arr)/sizeof(arr[0]));
        return 0;
}
```

Note: Address of the first element is called the base address.
The address of the **i^{th}** element is given by the following formula,

$$\mathbf{Address_i = base\_address + i*size\_of\_element}$$

Array Initialization:

- We can assign values directly in the declaration of array is called Initialization.
- If we assign the elements less than the length of array, other locations get default values.

```
int main(){
        int arr[5] = {10, 20, 30};
        int i;
        printf("Array default values are : \n");
        for(i=0 ; i<5 ; i++){
                printf("%d \n", arr[i]);
        }
        return 0;
}
```

C Program to display First and Last element of Array:

```
int main ()
{
        int arr[] = {10, 20, 30, 40, 50};
        printf("First element : %d \n", arr[0]);
        printf("Last element : %d \n", arr[4]);
        return 0;
}
```

C Program to display last element using length:

```
int main (){
        int arr[] = {10, 20, 30, 40, 50};
        int len = sizeof(arr)/sizeof(int);
        printf("Length : %d \n", len);
        printf("Last element : %d \n", arr[len-1]);
}
```

C Program to Check First and Last elements Sum equals to 10 or not in Array:

```
int main (){
        int arr[] = {4, 3, 9, 2, 6};
        int first = arr[0];
        int last = arr[4];
        if(first+last==10)
                printf("Equals to 10 \n");
        else
                printf("Not equals to 10 \n");
}
```

C program to check the First element of Array is Even or Not:

```
int main ()
{
	int arr[] = {4, 3, 9, 2, 6};
	if(arr[0]%2==0)
		printf("First element is Even \n");
	else
		printf("First element is not Even \n");
	return 0;
}
```

C Program to replace the First and Last elements of array with 10:

```
int main ()
{
	int arr[] = {4, 3, 9, 2, 6}, i;
	arr[0] = 10;
	arr[4] = 10;
	printf("Array elements are : \n");
	for(i=0 ; i<5 ; i++){
		printf("%d \n", arr[i]);
	}
	return 0;
}
```

C program to swap first and last elements of array:

```
int main ()
{
	int arr[] = {4, 3, 9, 2, 6};
	int i, temp;
	temp = arr[0];
	arr[0] = arr[4];
	arr[4] = temp;
	printf("Array elements are : \n");
	for(i=0 ; i<5 ; i++){
		printf("%d \n", arr[i]);
	}
	return 0;
}
```

C program to read elements into array and display:

```
#include<stdio.h>
int main ()
{
        int arr[5], i;
        printf("Enter 5 elements : \n");
        for(i=0 ; i<5 ; i++){
                scanf("%d", &arr[i]);
        }
        printf("Array elements are : \n");
        for(i=0 ; i<5 ; i++){
                printf("%d \n", arr[i]);
        }
        return 0;
}
```

C program to display array elements in reverse order:

```
#include<stdio.h>
int main ()
{
        int arr[5] = {10, 20, 30, 40, 50}, i;
        printf("Reverse Array : \n");
        for(i=4 ; i>=0 ; i--){
                printf("%d \n", arr[i]);
        }
        return 0;
}
```

C Program to find sum of array elements:

```
#include<stdio.h>
int main ()
{
        int arr[5] = {10, 20, 30, 40, 50}, i, sum=0;
        for(i=0 ; i<5 ; i++){
                sum = sum + arr[i];
        }
        printf("Sum of Array elements : %d \n", sum);
        return 0;
}
```

C Program to find average of array elements:

```
#include<stdio.h>
int main ()
{
        int arr[5] = {10, 20, 30, 40, 50}, i, sum=0, avg;
        for(i=0 ; i<5 ; i++){
                sum = sum + arr[i];
        }
        printf("Sum of Array elements : %d \n", sum);
        avg = sum/5;
        printf("Average of array elements : %d \n", avg);
}
```

C Program to display only even numbers in the array:

```
#include<stdio.h>
int main ()
{
        int arr[8] = {8, 5, 1, 7, 4, 3, 2, 9}, i;
        printf("Even numbers of Array : \n");
        for(i=0 ; i<8 ; i++){
                if(arr[i]%2==0)
                        printf("%d \t", arr[i]);
        }
}
```

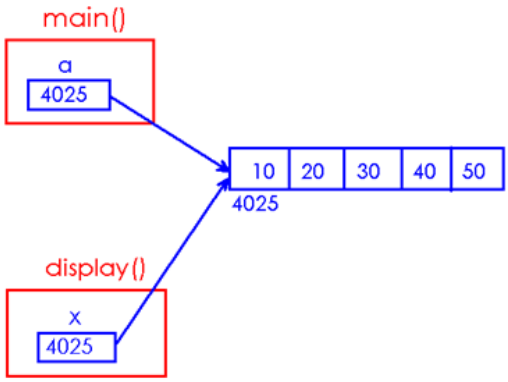
C program to display the largest element in the array:

```
#include<stdio.h>
int main()
{
        int arr[8] = {7,2,9,14,3,27,5,4};
        int large=arr[0], i;
        for(i=1 ; i<8 ; i++){
                if(arr[i]>large)
                        large = arr[i];
        }
        printf("Largest element is : %d \n", large);
        return 0;
}
```

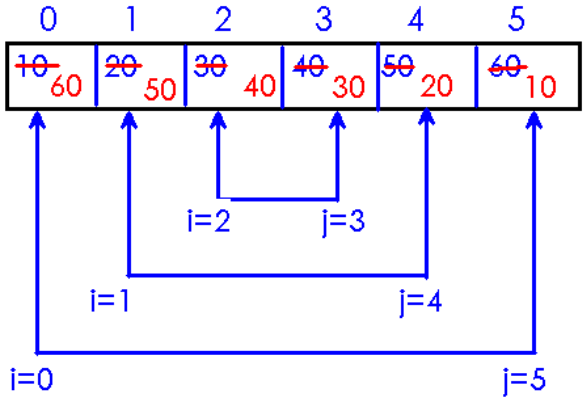
Pass Array as a parameter to a function:

- We can pass array as input parameter to function.
- By passing array name, the address will be passed as parameter.
- We collect the address in Called Function by declaring same type array.

```
void display(int[]);
int main(){
        int a[5] = {10, 20, 30, 40, 50};
        display(a);
        return 0;
}
void display(int x[]){
        int i;
        printf("Array is :\n");
        for(i=0 ; i<5 ; i++){
                printf("%d\n", x[i]);
        }
}
```



Write a program to reverse all elements in the array:



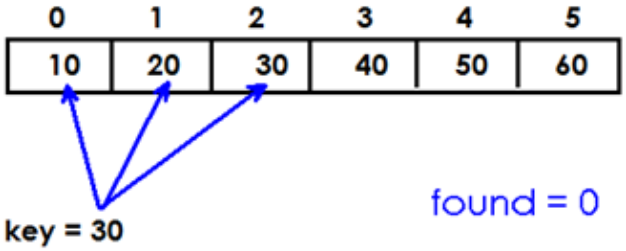

```
n = 6
i=0
j=n-1
while(i< j)
{
   temp =  a[i];
   a[i] = a[ j];
   a[ j] = temp;
   i++ ;
   j-- ;
}
```

```
#include<stdio.h>
void reverse(int[], int);
int main(){
        int a[5] = {10,20,30,40,50}, i;
        printf("Before reverse : \n");
        for(i=0 ; i<5 ; i++){
                printf("%d \n", a[i]);
        }
        reverse(a,5);
        return 0;
}
```

```
void reverse(int a[], int n)
{
        int i,j,temp;
        i=0;
        j=n-1;
        while(i<j)
        {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
                i++;
                j--;
        }
        printf("After reverse : \n");
        for(i=0 ; i<5 ; i++)
        {
                printf("%d \n", a[i]);
        }
}
```

Linear Search:

- Linear Search is used to search for an element in the array.
- Linear Search can be applied on Sorted array / Unsorted array.
- In linear search, index element starts with key element. If element found return index else display Error message.




```
for i -> 0 to n-1{
        if(key==a[i]){
            found = 1 ;
            break;
        }
}
if (!found)
   not found
```

```
int main()
{
        int a[30], m, key, i, n, found=0;
        printf("Enter size of array : ");
        scanf("%d",&m);

        printf("Enter Elements in array : \n");
```

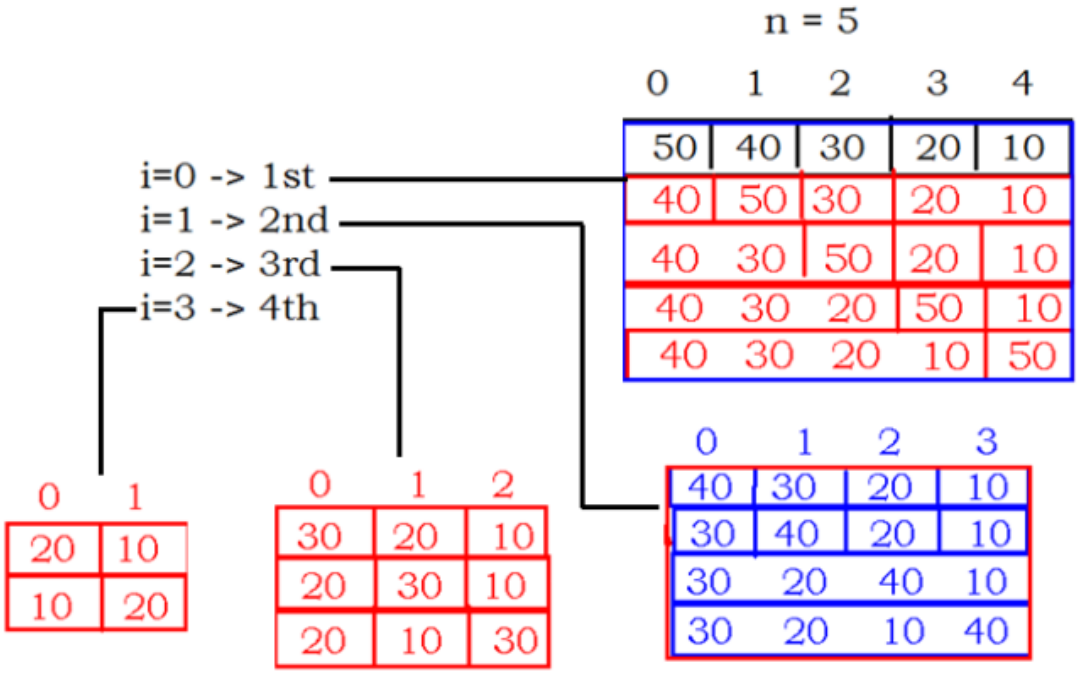
```
        for(i=0; i<m; i++)
    {
        scanf("%d",&a[i]);
    }

    printf("Enter the key to be searched: \n");
    scanf("%d",&key);

    for(i=0; i<n; i++)
    {
        if(key==a[i])
        {
            found=1;
            printf("Found @ location : %d \n", i);
            break;
        }
    }
    if(!found)
    {
        printf("Element not found \n");
    }
    return 0;
}
```

Bubble Sort:

- Simple sorting technique to arrange elements either in Ascending order or Descending order.
- In Bubble sort, the index element compares with the next element and swapping them if required.
- For each pass(inner loop completion) highest element bubbled to last location.
- This process continuous until the complete array get sorted.

```
#include<stdio.h>
int main()
{
    int arr[7] = {7, 3, 9, 4, 2, 5, 1}, i, j, n=7, t;
    for(i=0 ; i<n-1 ; i++)
    {
        for(j=0 ; j<n-1-i ; j++)
        {
            if(arr[j]>arr[j+1])
            {
```

```
                        t = arr[j];
                        arr[j] = arr[j+1];
                        arr[j+1] = t;
                    }
                }
        }
        printf("Sorted array : ");
        for(i=0 ; i<n ; i++)
        {
                printf("%d ", arr[i]);
        }
        return 0;
}
```



Two dimensional arrays:

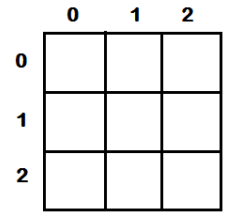
- Two-dimensional array stores data of format (Rows & Columns).
- Nested loops are used to access locations.

Syntax :

```
datatype  name[rows][columns];
```

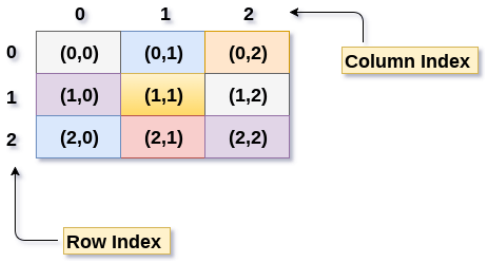
Example :

```
int   arr[3][3];
```



Print default values of 2-dimensional array:

```
#include<stdio.h>
int main()
{
        int arr[3][3], i, j;
        printf("Array elements are : \n");
        for(i=0 ; i<3 ; i++){
                for(j=0 ; j<3 ; j++){
                        printf("%d \t", arr[i][j]);
                }
                printf("\n");
        }
        return 0;
}
```



Two-dimensional array initialization:

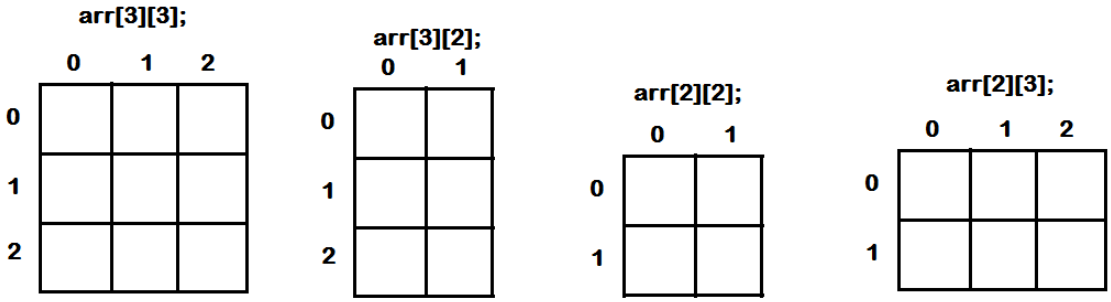
```
#include<stdio.h>
int main()
{
        int arr[3][3] = {10, 20, 30, 40, 50, 60, 70, 80, 90}, i, j;
        printf("Array elements are : \n");
        for(i=0 ; i<3 ; i++){
                for(j=0 ; j<3 ; j++){
                        printf("%d \t", arr[i][j]);
                }
                printf("\n");
        }
}
```

We can also assign values row wise as follows:

```
int main(){
        int arr[3][3] = {{10}, {20,30}, {0,40}}, i, j;
        printf("Array elements are : \n");
        for(i=0 ; i<3 ; i++){
                for(j=0 ; j<3 ; j++){
                        printf("%d \t", arr[i][j]);
                }
                printf("\n");
        }
}
```

Write a program to find the size of two-dimensional array:

```
#include<stdio.h>
int main()
{
        int a[3][3];
        printf("Total size : %d \n", sizeof(a));
        printf("Element size : %d \n", sizeof(a[0][0]));
        printf("Length : %d \n", sizeof(a)/sizeof(a[0][0]));
        printf("Row size : %d \n", sizeof(a[0]));
        printf("Row length : %d \n", sizeof(a[0])/sizeof(a[0][0]));
        return 0;
}
```

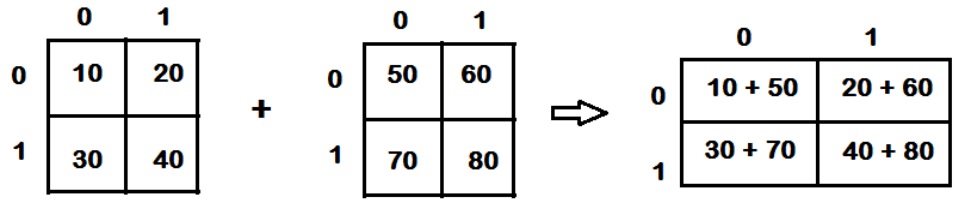


Read and Display elements of 2-Dimensional array:

```
int arr[2][2];
int main()
{
        int i, j;
        printf("Enter array elements : \n");
        for(i=0 ; i<2 ; i++){
                for(j=0 ; j<2 ; j++){
                        scanf("%d", &arr[i][j]);
```

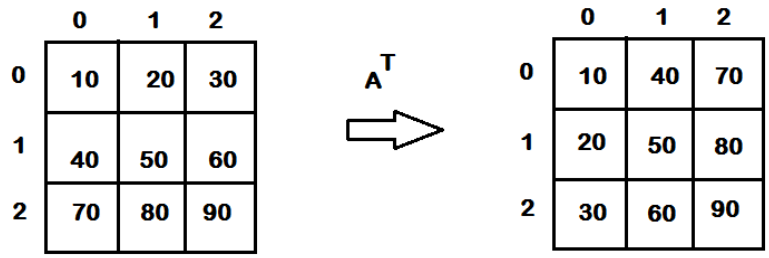
```
                }
                printf("\n");
        }
        printf("Array elements are : \n");
        for(i=0 ; i<2 ; i++){
                for(j=0 ; j<2 ; j++){
                        printf("%d \t", arr[i][j]);
                }
                printf("\n");
        }
}
```

Program to perform Matrix addition in C:



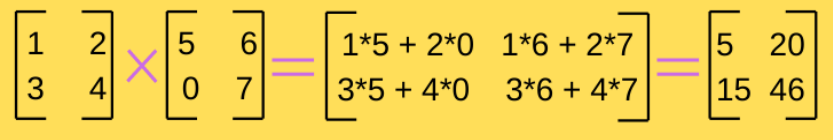

```
int main(){
        int a[2][2], b[2][2], c[2][2], i, j;
        printf("Enter elements of A : \n");
        for(i=0 ; i<2 ; i++)
                for(j=0 ; j<2 ; j++)
                        scanf("%d", &a[i][j]);

        printf("Enter elements of B : \n");
        for(i=0 ; i<2 ; i++)
                for(j=0 ; j<2 ; j++)
                        scanf("%d", &b[i][j]);
        for(i=0 ; i<2 ; i++)
                for(j=0 ; j<2 ; j++)
                        c[i][j] = a[i][j] + b[i][j];

        printf("Added matrix is : \n");
        for(i=0 ; i<2 ; i++){
                for(j=0 ; j<2 ; j++){
                        printf("%d \t", c[i][j]);
                }
                printf("\n");
        }
}
```

Transpose of a Matrix program in C:




```
#include<stdio.h>
int main()
{
	int arr[3][3]={{10,20,30},{40,50,60},{70,80,90}} ,I ,j ,temp;
	printf("Matrix is: \n");
	for(i=0 ; i<3 ; i++){
		for(j=0 ; j<3 ; j++){
			printf("%d\t", arr[i][j]);
		}
		printf("\n");
	}

	for(i=0 ; i<3 ; i++){
		for(j=0 ; j<3 ; j++){
			if(i<j){
				temp=arr[i][j];
				arr[i][j]=arr[j][i];
				arr[j][i]=temp;
			}
		}
		printf("\n");
	}

	printf("Transpose of matrix: \n");
	for(i=0 ; i<3 ; i++){
		for(j=0 ; j<3 ; j++){
			printf("%d\t", arr[i][j]);
		}
		printf("\n");
	}
	return 0;
}
```

Program to perform Matrix Multiplication in C:




```
int main(){
        int a[2][2],b[2][2],c[2][2],i,j,k;
        printf("Enter array A \n");
        for(i=0;i<2;i++){
                for(j=0;j<2;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        printf("Enter array B \n");
        for(i=0;i<2;i++){
                for(j=0;j<2;j++)
                {
                        scanf("%d",&b[i][j]);
                }
        }
        for(i=0;i<2;i++){
                for(j=0;j<2;j++)
                {
                        c[i][j]=0;
                        for(k=0;k<2;k++)
                        {
                                c[i][j]+=a[i][k]*b[k][j];
                        }
                }
        }
        printf("Multiplication array \n");
        for(i=0;i<2;i++){
                for(j=0;j<2;j++)
                {
                        printf("%d\t",c[i][j]);
                }
                printf("\n");
        }
        return 0;
}
```

Module-3

(Strings, Structures and Pointers)

String Handling in C

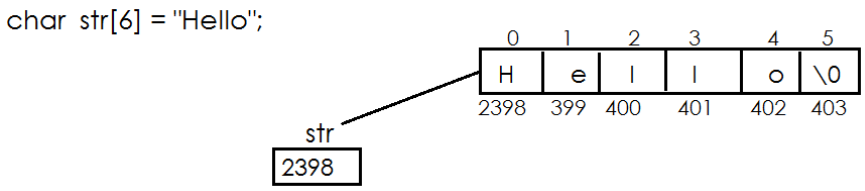
String:

- String is a one-dimensional character array.
- String is a collection symbols (alphabets, digits and special symbols).
- Strings must be represented with double quotes.

Syntax:	**Example:**
char identity[size];	char name[20];

Memory representation:

- We can store only 'n-1' characters into an array of size 'n'.
- Last character of every string is '\0'
- ASCII value of '\0' character is 0.



String Format Specifier(%s):

- We use %s to read and display strings.
- To process character by character, we use %c specifier.

Program to display String:

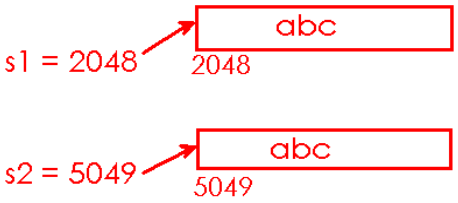
```
#include<stdio.h>
int main()
{
        char str[20] = "Hello" ;
        printf("%s all \n" , str);
        return 0;
}
```

We can display complete character array in single statement using %s:

```
#include<stdio.h>
int main()
{
        char str[4] = {'a','b','c'};
        printf("String is : %s\n" , str);
        return 0;
}
```

In C, duplicate strings get different memory locations:

```
int main()
{
    char s1[10] = "abc";
    char s2[10] = "abc";

    printf("s1 addr : %u\n" , s1);
    printf("s2 addr : %u\n" , s2);
    return 0;
}
```



== operator only compares the addresses of strings, hence following program outputs "Strings are not equal"

```
int main(){
        char s1[10] = "abc";
        char s2[10] = "abc";
        if(s1==s2)
                printf("Strings are equal \n");
        else
                printf("Strings are not equal \n");
        return 0;
}
```

String will be terminated when it reaches '\0' character:

```
int main(){
        printf("Hello \0 World \n");
        return 0;
}
```

Analyze the output of following code:

```
int main(){
        printf("Hello World \0 \n");
        printf("Hello \0 World \n");
        printf("\0Hello  World \n");
        return 0;
}
```

Note: Strings can be processed with \0 character only. ASCII value as follows.

```
#include<stdio.h>
int main(){
        printf("\0 ascii value is : %d \n", '\0');
        return 0;
}
```

Note: In the above program, printf() function stops printing string when \0 reached.

Program to print null character(\0) on console:

```
int main(){
        printf("\\0 ascii value is : %d \n", '\0');
        return 0;
}
```

Program to display Strings with quotations:

Expected Output: It is 'C-lang' class

```
int main(){
        printf("This is 'C-lang' class \n");
        return 0;
}
```

Expected Output: It's a C class

```
int main(){
        printf("It's a C class \n");
        return 0;
}
```

Expected Output: I am attending "C - Online" class

```
int main(){
        printf("I am attending \"C - Online\" class \n");
        return 0;
}
```

Expected Output: She said, "It's a good one"

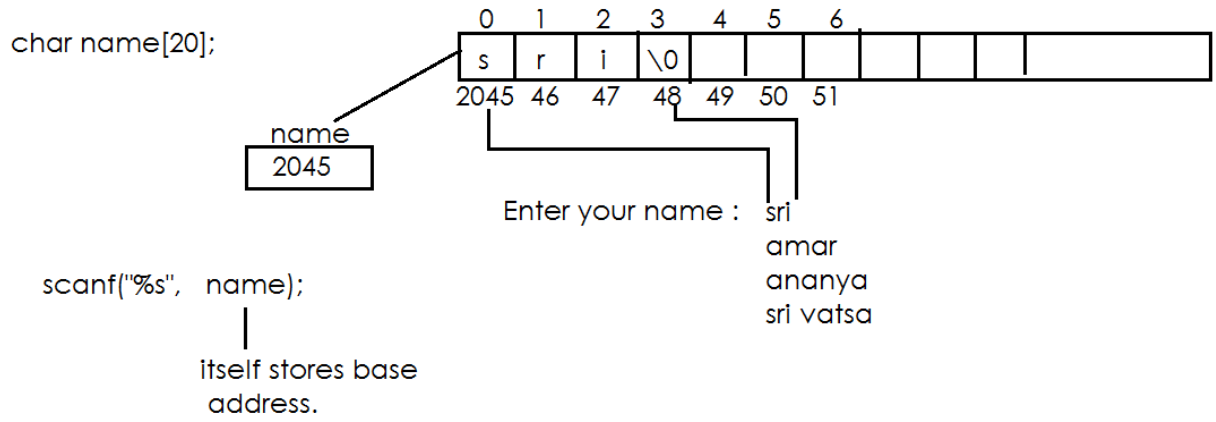
```
int main(){
        printf("She said, \"It's a good one\" \n");
        return 0;
}
```

Program to read and display string:

```
int main(){
        char name[20];
        printf("Enter your name : ");
        scanf("%s", name);
        printf("Hello %s, Welcome to Strings \n", name);
}
```

How %s read input into String variable:

- "%s" takes the base address of memory block.
- It reads character by character and stores from the base address.
- Once input is over, it will add null character at the end of the string.



Program to read multi-word string in C:

- gets() function can read a string with multiple spaces.
- It stops reading characters into array only when we hit "enter" key.

```
int main(){
        char name[20];
        printf("Enter multi word name : ");
        gets(name);
        printf("Hello %s \n", name);
}
```

Program to display the String character by character:

```
int main(){
        char name[20];
        int i;
        printf("Enter your name : ");
        gets(name);
        i=0;
        while(name[i] != '\0'){
                printf("%c \n", name[i]);
                i++;
        }
        return 0;
}
```

string.h functions

strlen() :

- strlen() returns length of string excluding null character.
- Return type is size_t (positive integer)

Prototype is *size_t strlen(char s[]);*

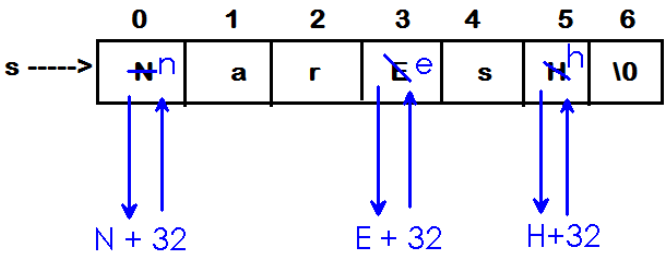
```
#include<string.h>
int main(){
        char str[20];
        size_t len;
        printf("Enter string : ");
        gets(str);
        len = strlen(str);
        printf("Length is : %u \n", len);
        return 0;
}
```

Program to find the length of String without using library function:

```
int main(){
        char str[20];
        size_t len=0, i=0;
        printf("Enter string : ");
        gets(str);
        while(str[i] != '\0'){
                len++;
                i++;
        }
        printf("Length is : %u \n", len);
        return 0;
}
```

Program to convert Upper case characters into Lower case in String:

```
#include<string.h>
int main(){
        char src[20];
        printf("Enter Upper string : ");
        gets(src);
        strlwr(src);
        printf("Lower string : %s \n", src);
        return 0;
}
```



ASCII

A-65	a-97
B-66	b-98
..	..
..	..
..	..
..	..
Z-90	z-122

A + 32 --> a **a - 32 --> A**
... **...**
Z + 32 --> z **z - 32 --> Z**

Program to convert Upper case characters into Lower case without Library function:

```
#include<stdio.h>
#include<string.h>
int main(){
        char src[20];
        int i=0;
        printf("Enter Upper string : ");
        gets(src);

        while(src[i] != '\0')
        {
                if(src[i]>='A' && src[i]<='Z')
                {
                        src[i] = src[i]+32;
                }
                i++;
        }
        printf("Lower string : %s \n", src);
}
```

Program to reverse the given string in C:

```
#include<string.h>
int main()
{
        char src[20];
        printf("Enter string : ");
        gets(src);
        strrev(src);
```

```
        printf("Reverse string : %s \n", src);
}
```

Program to reverse the String without using Library function:

```
#include<string.h>
int main()
{
        char src[20], temp;
        int i, j;

        printf("Enter string : ");
        gets(src);

        i=0;
        j=strlen(src)-1;
        while(i<j)
        {
                temp = src[i];
                src[i] = src[j];
                src[j] = temp;
                i++;
                j--;
        }
        printf("Reverse string : %s \n", src);
}
```

strcmp(): Compare strings are return 0 if equal else return the ASCII difference between mismatching characters.

```
#include<string.h>
int main()
{
        char s1[20] = "hello";
        char s2[20] = "Hello";

        if(strcmp(s1,s2)==0)
                printf("Strings are equal \n");
        else
                printf("String are not equal \n");
}
```

Compare Strings without considering the Case using stricmp() function:

```
#include<stdio.h>
#include<string.h>
int main()
{
        char s1[20] = "HELLO";
        char s2[20] = "Hello";

        if(stricmp(s1,s2)==0)
                printf("Strings are equal \n");
        else
                printf("String are not equal \n");
}
```

strstr(): is used to find the first occurrence of a substring.

```
#include <stdio.h>
#include <string.h>
int main()
{
   char text[] = "Welcome to C programming";
   char word[] = "C";

   char *result = strstr(text, word);

   if (result != NULL)
   {
      printf("Substring found: %s\n", result);
   }
   else
   {
      printf("Substring not found.\n");
   }
   return 0;
}
```

Structures in C

Introduction:

- Applications are used to store and process the information
- We store information using variables and process using functions.
- We use different types of variables to store the information as follows.

Primitive variable: stores only one value at a time.

int a = 10;	a 10

Array variable: stores more than one value but of same type.

int arr[5] = {10, 20, 30, 40, 50};	arr ---> 10 | 20 | 30 | 40 | 50

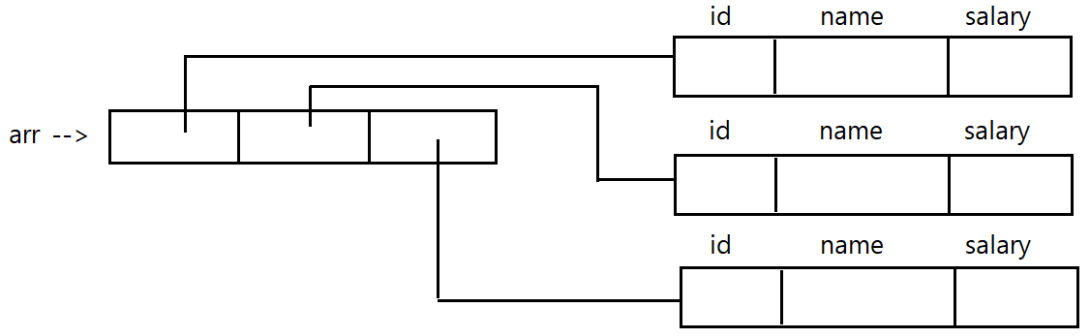
Structure variable: stores more than one value of different data types. We define structures using struct keyword.

struct Employee
{
int id;
char name[20];
float salary;
};
struct Employee e = {101, "Amar", 35000};
	emp --> id | name | salary

Array of Structures: used to store more than one record details.

```
struct Employee{
        int id;
        char name[20];
        float salary;
};
struct Employee arr[3];
```

Memory representation:



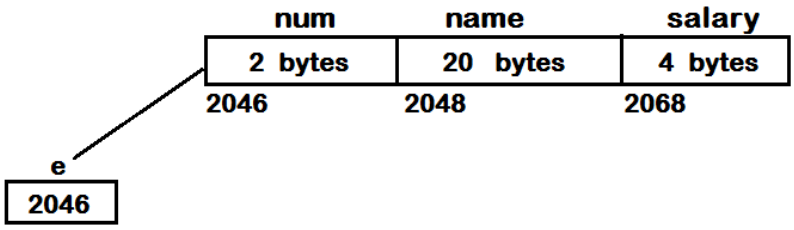
Note: Structure is a data type and memory allocate to structure only when we create variable

Structure (Data type)	Memory allocation (create variable)

struct Employee
{
int id;
char name[20];
float salary;
};
	struct Employee e;

Program to find the size (memory allocated to structure): sizeof() functions returns the total number bytes allocated to structure.

```
struct Emp
{
    int num ;
    char name[20];
    float salary ;
};

struct Emp e ;
```



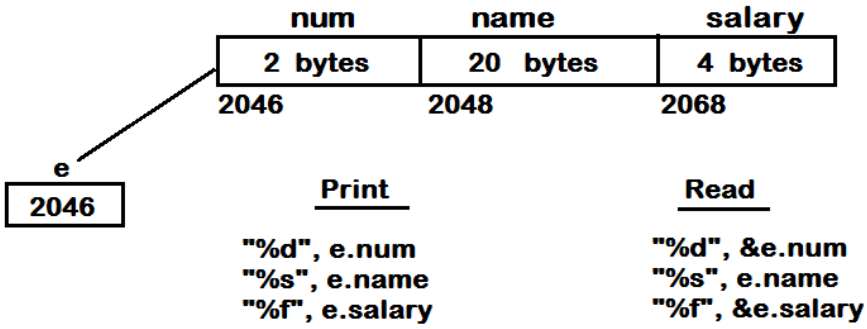

```
#include<stdio.h>
struct Emp
{
        int id;
        char name[20];
        float salary;
};
int main()
{
        struct Emp x;
        printf("size of Emp : %d \n", sizeof(x));
        printf("size of Emp : %d \n", sizeof(struct Emp));
        return 0;
}
```

Structure initialization: Like Primitive types and Arrays, we can assign values directly to structure variables at the time of declaration.

struct identity var = {val1, val2, val3 ⋯};

Accessor (dot operator): We must access the locations of structure through dot(.) operator

```
#include<stdio.h>
struct Employee
{
        int id;
        char name[20];
        float salary;
};
int main()
{
        struct Employee e = {101, "amar", 35000};
        printf("Employee id : %d \n", e.id);
        printf("Employee name : %s \n", e.name);
        printf("Employee salary : %f \n", e.salary);
        return 0;
}
```



Program to pass structure as parameter to function:

- Functions can take structure as input parameter.
- We pass the structure variable as parameter and collect into same type of structure variable.

Passing int	Passing array	Passing String	Passing structure

void abc(int);
main()
{
int a=10;
abc(a);
}
abc(int a)
{
logic
}
	void abc(int[ ]);
	main()
	{
	int a[ ] ={10,20,30};
	abc(a);
	}
	abc(int a[ ])
	{
	logic
	}
		void abc(char[ ]);
		main()
		{
		char s[ ] = "abc" ;
		abc(s);
		}
		abc(char s[ ])
		{
		logic
		}
			void abc(struct Emp);
			main()
			{
			struct Emp e = { , , } ;
			abc(e);
			}
			abc(struct Emp e)
			{
			logic
			}

```
#include<stdio.h>
struct Employee {
        int id;
        char name[20];
        float salary;
};
void display(struct Employee);
int main()
{
        struct Employee x = {101, "Amar", 35000};
        display(x);
        return 0;
}
void display(struct Employee y){
        printf("Emp id : %d \n", y.id);
        printf("Emp name : %s \n", y.name);
        printf("Emp salary : %f \n", y.salary);
}
```

Program to define a function that returns structure:

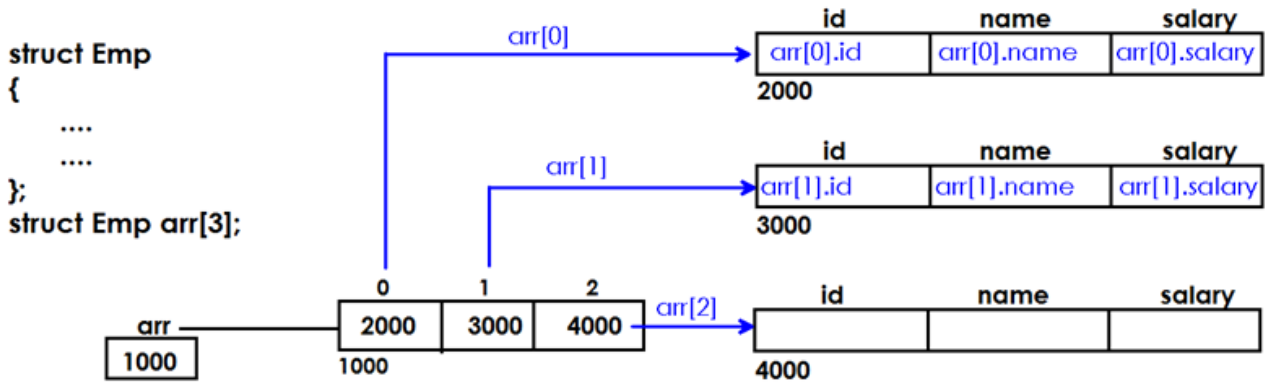
```
#include<stdio.h>
struct Emp{
        int id;
        char name[20];
        float salary;
};
struct Emp collect();
int main()
{
        struct Emp y;
        y = collect();
        printf("Emp id : %d \n", y.id);
        printf("Emp name : %s \n", y.name);
        printf("Emp salary : %f \n", y.salary);
        return 0;
}
struct Emp collect(){
        struct Emp x = {101, "Amar", 35000};
        return x;
}
```

Array of Structures: by creating array type variable to structure data type, we can store multiple records like Employee details, Customer details, Student details, Account details etc.

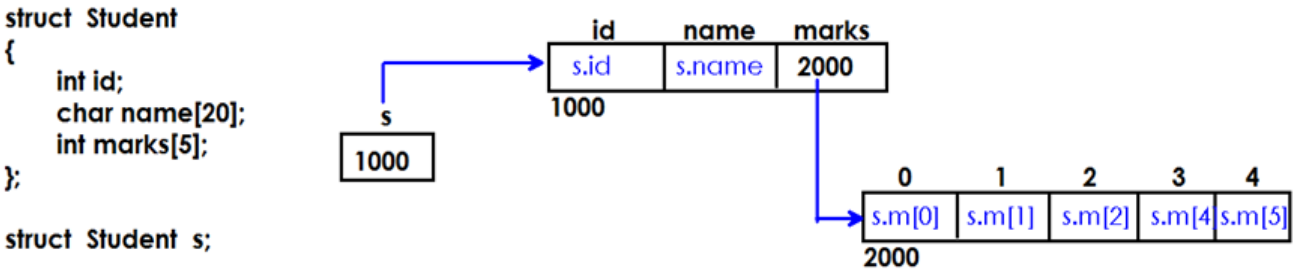
struct Emp{
....
};
	struct Emp e; -> store 1 record
	struct Emp e1, e2, e3; -> store 3 records
	struct Emp arr[100]; -> store 100 records

Program to store multiple records into Array variable of structure type:

```
#include<stdio.h>
struct Emp{
        int id;
        char name[20];
        float salary;
};
int main(){
        struct Emp arr[3];
        int i;

        printf("Enter 3 Emp records : \n");
        for(i=0 ; i<3 ; i++)
        {
                printf("Enter Emp-%d details : \n", i+1);
                scanf("%d%s%f", &arr[i].id, arr[i].name, &arr[i].salary);
        }
        printf("Employee details are : \n");
        for(i=0 ; i<3 ; i++)
        {
                printf("%d \t %s \t %f \n", arr[i].id, arr[i].name, arr[i].salary);
        }
        return 0;
}
```

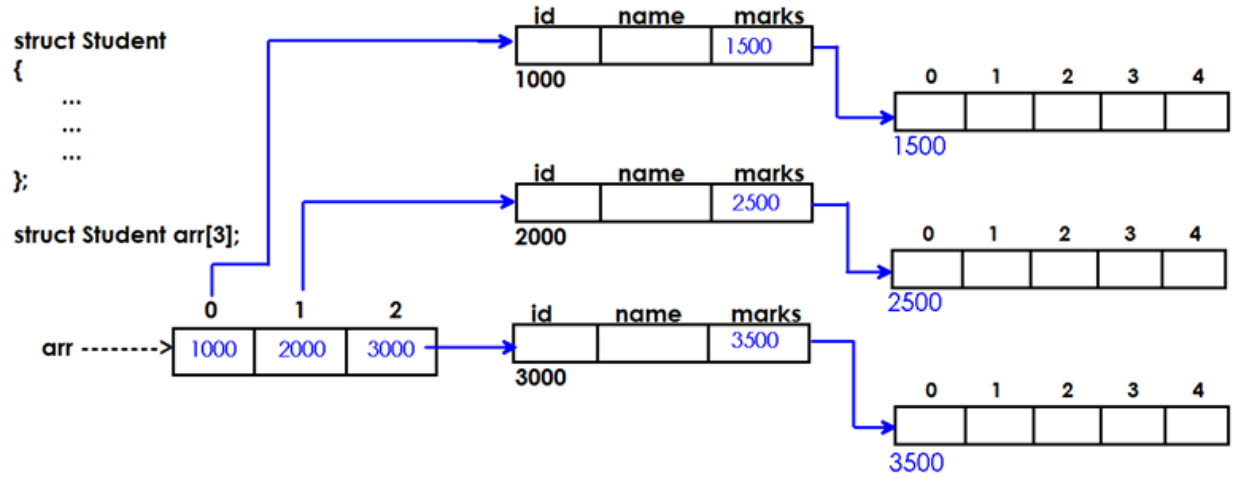


Arrays in structure: If the structure contains Array type variable, we need to process the data using loops.




```
#include<stdio.h>
struct Student
{
        int id;
        char name[20];
        int marks[5];
};

int main()
{
        struct Student s;
        int i;
        printf("Enter Student id and name : \n");
        scanf("%d%s", &s.id, s.name);
        printf("Enter student marks of 5 subjects : \n");
        for(i=0 ; i<5 ; i++)
        {
                scanf("%d", &s.marks[i]);
        }
        printf("Student details are : \n");
        printf("%d, %s, ", s.id, s.name);
        for(i=0 ; i<5 ; i++)
        {
                printf("%d ,", s.marks[i]);
        }
        return 0;
}
```

Store more than one student records including their marks:

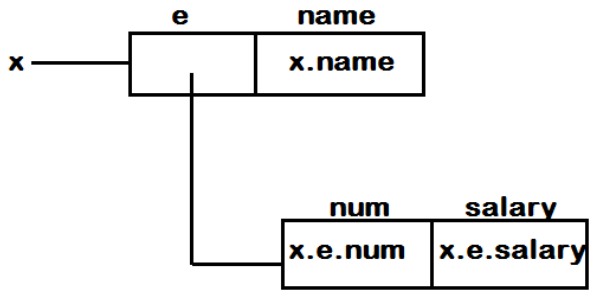



```
#include<stdio.h>
struct student
{
        int id;
        char name[20];
        int marks[5];
};
int main()
{
        struct student a[3];
        int i,j;
        printf("Enter student details\n");
        for(i=0;i<3;i++) {
                printf("Enter student id and name\n");
                scanf("%d%s",&a[i].id,a[i].name);
                printf("Enter marks of student\n");
                for(j=0;j<5;j++)
                        scanf("%d",&a[i].marks[j]);
        }
        printf("Student details are \n");
        for(i=0 ; i<3 ; i++) {
                printf("%d \t %s \t ",a[i].id,a[i].name);
                for(j=0;j<5;j++)
                        printf("%d\t",a[i].marks[j]);
                printf("\n");
        }
        return 0;
}
```

Nested structures: Defining a structure inside another structure. We access the elements of nested structure using outer structure reference variable.

```
#include<stdio.h>
struct Emp
{
        int num;
        float salary;
};
struct Employee
{
        struct Emp e;
        char name[20];
};
int main()
{
        struct Employee x;

        printf("Enter emp details(num, name, salary) : ");
        scanf("%d%s%f", &x.e.num, x.name, &x.e.salary);

        printf("Emp name is : %d \n",x.e.num);
        printf("Emp name is : %s \n",x.name);
        printf("Emp name is : %f \n",x.e.salary);
        return 0;
}
```

```
struct  Emp
{
    int num ;
    float salary ;
};

struct  Employee
{
    struct Emp e ;
    char name[20];
}

struct  Employee x ;
```



Unions in C

Introduction:

- Union is a user defined data type.
- Unions were used when memory at premium.
- A union basically allows a variable to have more than one type; but only one of these types can be used.

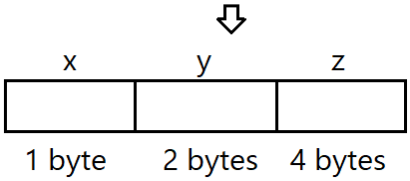
Syntax:	**Example:**

union identity
{
Members;
} ;
	union Test
	{
	char c;
	short s;
	};

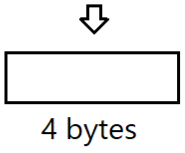
STRUCTURE	UNION
Define with struct keyword.	Define with union keyword
The size of the structure is equal to the sum of the sizes of its members.	The size of the union is equal to the size of the largest member.
Each member within a structure is assigned a unique storage area.	Memory allocated is shared by individual members of the union.
Individual members can be accessed at a time.	Only one member can be accessed at a time.
Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.

Program to display size of union: The sum of all sizes of variables defined in structures is the size of total structure.

```
struct Test
{
    char x ;
    int y ;
    float z ;
};
struct Test t ;
```



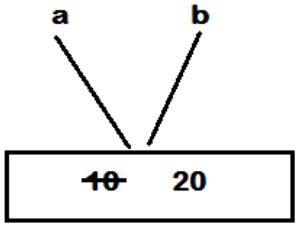

```
union Test
{
    char x ;
    int y ;
    float z ;
};
union Test u ;
```



```
#include<stdio.h>
struct st{
        char a;
        short b;
        float c;
};
union un{
        char a;
        short b;
        float c;
};
int main(){
        struct st s ;
        union un u ;
        printf("Size of structure : %d \n", sizeof(s));
        printf("Size of union : %d \n", sizeof(u));
        return 0;
}
```

Accessing members of union: We use dot(.) operator to access the members of union.

Note: We can define any number of variables inside the union, but we cannot use multiple variables at a time. We will get odd results like follows.



a = 10 ;
print(b); --> 10

b = 20 ;
print(a); --> 20

```
#include<stdio.h>
union un{
        int a, b;
};
int main(){
        union un u ;
        u.a = 10;
        printf("b value is : %d \n", u.b);
        u.b = 20;
        printf("a value is : %d \n", u.a);
        return 0;
}
```

Pointers in C

Introduction:

- Pointers is a derived data type in C.
- Pointer type variables store addresses of memory locations.

Syntax : **datatype *name ;** **Example :** **int *p ;** **int* p ;**	**int* p, q ;** **pointer type** — **integer type**	**int* p, *q ;** **Both are pointers**

Pointers classified into:

1. **Typed:** These pointers can point to specific type data
   int* -> points to integer data
   float* -> points to float data
   struct Emp* -> points to Emp data

2. **Un-typed:** Can points to any type of data. It is also called Generic pointer.
   void* -> can points to any data.

Operators using with Pointers: Every operation in data processing using pointers is possible through 2 operators.

1. **Address Operator (&):** It returns the memory address of specified variable.
2. **Pointer operator (*):** It returns the value in the specified address.

Following program explains the memory representation of pointer variables:

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

Output:

```
10
37814056
37814056
37814052
10
37814056
10
```

```
**(&p)
*4052
*4056
10
```

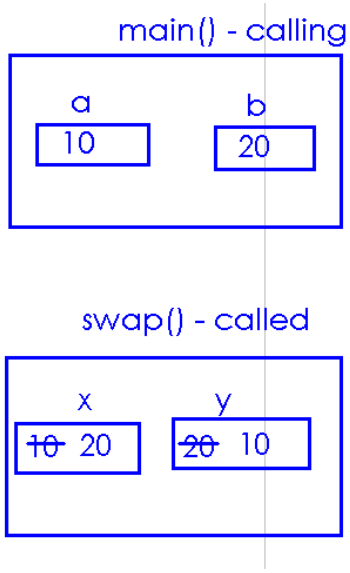
Call by Value & Call by Reference

Call by value: A copy of the variable is passed to the function.
Call by reference: An address of the variable is passed to the function.

Note: Call by reference is preferred when we have to return more than one variable, like in C programming where we can only return one variable at a time. Call by reference can be achieved via pointers.

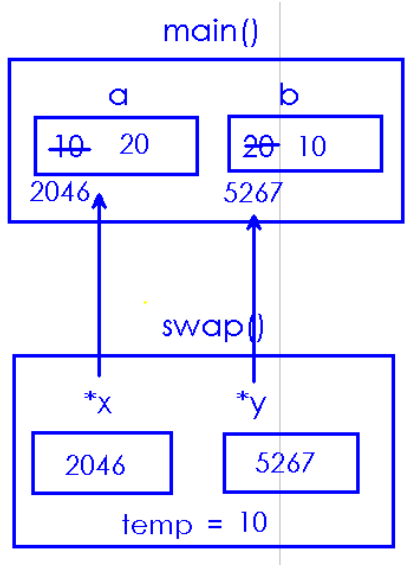
Swapping two numbers:

```
#include<stdio.h>
void swap(int,int);
int main()
{
    int a=10, b=20;
    printf("Before swap : %d, %d \n", a, b);   --> 10, 20
    swap(a,b);
    printf("After swap : %d, %d \n", a, b);    --> 10, 20
    return 0;
}
void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
    printf("After swap : %d, %d \n", x, y);    --> 20, 10
}
```



Call By Reference: In this example, the output shows the correct result. This happens because we have sent the exact address of the variable.

```
#include<stdio.h>
void swap(int*,int*);
int main()
{
    int a=10, b=20;
    printf("Before swap : %d, %d \n", a, b);   --> 10, 20
    swap(&a,&b);
    printf("After swap : %d, %d \n", a, b);    --> 20, 10
    return 0;
}
void swap(int* x, int* y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
    printf("After swap : %d, %d \n", *x, *y);  --> 20, 10
}
```



Size of Pointer:

- Pointer variable stores address(integer) hence the size of pointer equals to integer size.
- sizeof() function can be used to display the size of each pointer type.

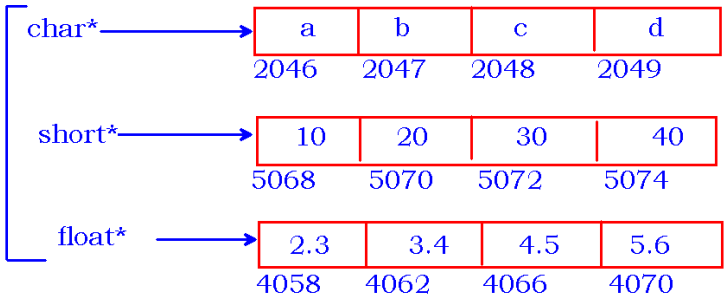
```
#include<stdio.h>
int main(){
        char* a;
        short* b;
        float* c;
        double* d;
        printf("char* size : %d \n", sizeof(a));
        printf("short* size : %d \n", sizeof(b));
        printf("float* size : %d \n", sizeof(c));
        printf("double* size : %d \n", sizeof(d));
        return 0;
}
```

Pointer size varies from compiler to compiler:

Compiler	Int size	Pointer size
16 bit	2 bytes	2 bytes
32 bit	4 bytes	4 bytes
64 bit	8 bytes	8 byte

Pointer increment / decrement: When we modify the pointer, the value increase or decrease by size bytes.

Using pointer arithmetic, we can move the control from one location to another location to process the data easily.




```
int main(){
        char x = 'a';
        float y = 2.3;
        double z = 4.5;
        char* p1 = &x;
        float* p2 = &y;
        double* p3 = &z;
```

```
        printf("p1 value is : %u \n", p1);
        printf("++p1 value is : %u \n", ++p1);
        printf("p2 value is : %u \n", p2);
        printf("++p2 value is : %u \n", ++p2);
        printf("p3 value is : %u \n", p3);
        printf("++p3 value is : %u \n", ++p3);
}
```

Pointer to function: Function has address that we can assign to pointer variable by which we can invoke the function.

Syntax: <return_type> (*<ptr_name>)(args_list);

Example: int (*fptr)(int,int);

Where: "fptr" is a pointer that points to a function which is taking 2 integers and return integer.

```
int add(int,int);
int sub(int,int);
int main(){
        int r1, r2, r3, r4;
        int (*fptr)(int,int);
        r1=add(10,20);
        r2=sub(10,20);
        printf("r1 : %d\nr2 : %d\n",r1,r2);

        fptr = &add; /*pointing to add function*/
        r3=fptr(30,50);

        fptr = &sub; /*pointing to sub function*/
        r4=fptr(30,50);
        printf("r3 : %d\nr4 : %d\n",r3,r4);
        return 0;
}
int add(int x, int y){
        return x+y;
}
int sub(int x, int y){
        return x-y;
}
```

Pointer type-casting:

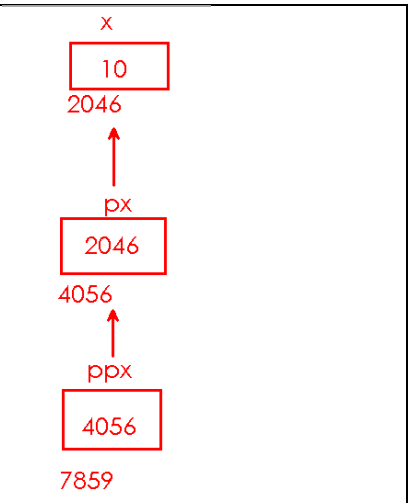
- Converting one type of pointer variable into another type to access the data.
- (type*) syntax is used to perform type casting.

```
#include<stdio.h>
void main(){
        int i = 100;
        int* ip;
        char* cp;
        ip = &i;
        cp = (char*)ip;
        printf("i value : %d\n", *ip);
        printf("i value : %d\n", *cp);
}
```

Pointer to Pointer:

- It is also called “double pointer”
- A “Pointer to Pointer” variable holds the address of another “Pointer variable”

```
#include<stdio.h>
int main()
{
        int x = 10;
        int* px = &x;
        int** ppx = &px;

        printf("x val : %d \n", x);
        printf("x val : %d \n", *px);
        printf("x val : %d \n", **px);
        return 0;
}
```

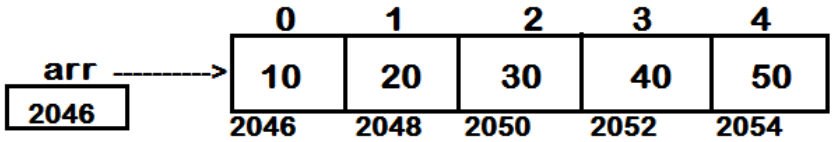


Array internal pointer representation:

- Generally, we access elements of array using their index.
- For example, arr[i]
- The expression arr[i] converts into *(arr+i)

```
#include<stdio.h>
int main()
{
        int arr[5] = {10, 20, 30, 40, 50}, i;
        printf("Array elements are : \n");
```

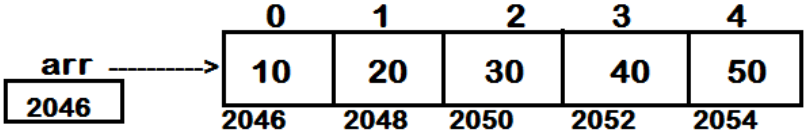
```
        for(i=0 ; i<5 ; i++)
                printf("%d\n", arr[i]);

        printf("Array elements are : \n");
        for(i=0 ; i<5 ; i++)
                printf("%d\n", *(arr+i));
        return 0;
}
```



arr[i] --> *(arr+i)
***(2046+0) --> *2046 --> 10**
***(2046+1) --> *2048 --> 20**
***(2046+2) --> *2050 --> 30**
***(2046+3) --> *2052 --> 40**
***(2046+4) --> *2054 --> 50**

In multiple ways, we can access the elements of array:

```
#include<stdio.h>
int main()
{
        int arr[5] = {10,20,30,40,50};
        int i;
        printf("Array elements are : \n");
        for(i=0 ; i<5 ; i++){
                printf("%d, %d, %d, %d \n", arr[i], *(arr+i), *(i+arr), i[arr]);
        }
        return 0;
}
```



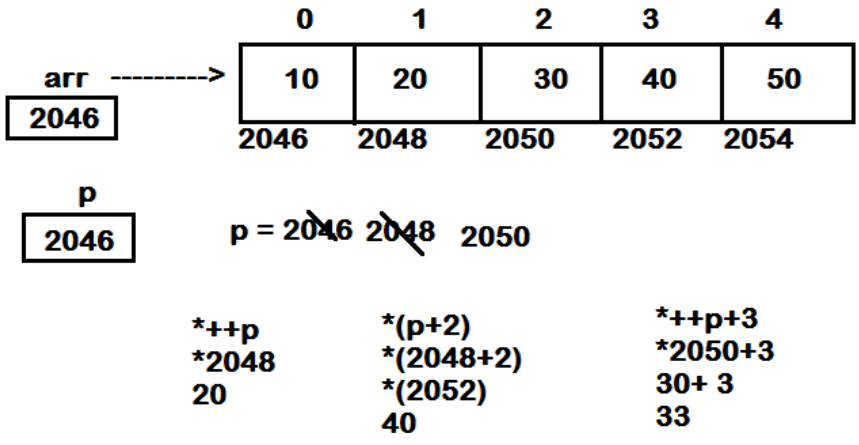
arr[i] --> *(arr+i)
***(2046+0) --> *2046 --> 10**
⇩
***(0+2046) --> *2046 --> 10**
⇩
***(i+arr) ---> i[arr]**

Pointers to Arrays:

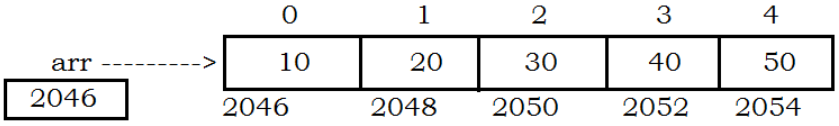
- Array variable stores address of memory block hence it is pointer type.
- We can assign array variable directly to pointer variable and access the element using that pointer variable.

```
#include<stdio.h>
void main()
{
        int a[5] = {10,20,30,40,50}, i;
        int *p = a;
        p=a;
        for(i=0; i<5; i++){
                printf("%d\t",a[i]);
                printf("%d\t",p[i]);
                printf("%d\t",*(p+i));
                printf("%d\t",*(i+p));
                printf("%d\n",i[p]);
        }
}
```

```
#include<stdio.h>
int main()
{
        int arr[5] = {10,20,30,40,50};
        int *p = arr;
        printf("%d \n", *++p);
        printf("%d \n", *(p+2));
        printf("%d \n", *++p+3);
        return 0;
}
```



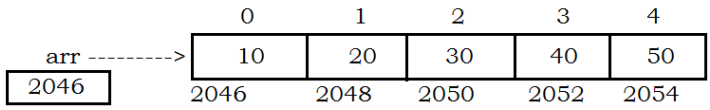
```
#include<stdio.h>
int main(){
        int arr[5] = {10,20,30,40,50};
        int *ptr = arr;
        printf("%d \n", *(++ptr+1));
        return 0;
}
```



ptr = ~~2046~~ 2048

--> *(++ptr + 1)
--> *(2048 + 1)
--> *2050
--> 30

```
#include<stdio.h>
int main(){
        int arr[5] = {10,20,30,40,50},i;
        int* ptr;
        ptr = arr;
        printf("%u\n", *++ptr + 3);
        printf("%u\n", *(ptr-- + 2) + 5);
        printf("%u\n", *(ptr+3)-10);
        return 0;
}
```

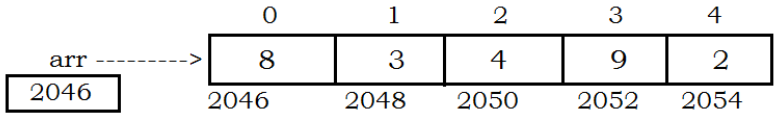


ptr = ~~2046~~ ~~2048~~ 2046

*++ptr + 3	*(ptr-- + 2) + 5	*(ptr+3)-10
*2048 + 3	*(2048 + 2) + 5	*(2046+3)-10
20 + 3	*(2052) + 5	*2052 - 10
23	40 + 5	40 - 10
	45	30

```
#include<stdio.h>
int main(){
        int arr[5] = {8, 3, 4, 9, 2},i;
```

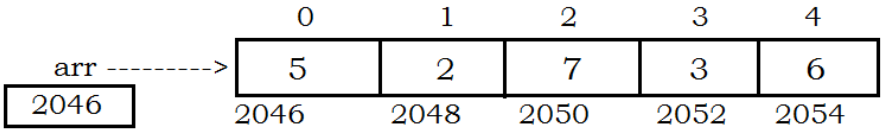
```
    int* ptr;
    ptr = arr;
    printf("%u\n", *(--ptr+2) + 3);
    printf("%u\n", *(++ptr + 2) - 4);
    printf("%u\n", *(ptr-- +1 ) + 2);
    return 0;
}
```



ptr = ~~2046~~ ~~2044~~ 2044 ~~2046~~

*(--ptr+2) + 3	*(++ptr + 2) - 4	*(ptr-- +1) + 2
*(2044+2)+3	*(2046 + 2) - 4	*(2046 + 1) + 2
*(2048)+3	*(2050) - 4	*(2048) + 2
3+3	4 - 4	3 + 2
6	0	5

```
#include<stdio.h>
int main(){
    int arr[5] = {5, 2, 7, 3, 6},i;
    int* ptr;
    ptr = arr;
    printf("%d\n", *(ptr++ + 1) - 2);
    printf("%d\n", *(ptr-- + 3) - 10);
    printf("%d\n", *(--ptr + 2 ) + 5);
    return 0;
}
```



ptr = ~~2046~~ ~~2048~~ ~~2046~~ 2044

*(ptr++ + 1) - 2	*(ptr-- + 3) - 10	*(--ptr + 2) + 5
*(2046 + 1) -2	*(2048 + 3) - 10	*(2044 + 2) + 5
*(2048) - 2	*(2054) - 10	*(2048) + 5
2 - 2	6 - 10	2 + 5
0	-4	7

Pointer to String:

- A char* can points to a single character or a String.
- Char* holds the base address.
- We can process the strings using %s
- To process character by character, we use %c

```
#include<stdio.h>
void main()
{
        char* s = "Harsha" ;
        printf("%s \n", s);
        printf("%c \n", s);
        printf("%c \n", *s);
        printf("%c \n", *(s+3));
        printf("%c \n", *s+3);
}

void main()
{
        char* s = "Sathya" ;
        printf("%c \n", *++s+2);
        printf("%c \n", *++s);
        printf("%c \n", *(s+2));
        printf("%c \n", *s--);
        printf("%c \n", *--s);
}
#include<stdio.h>
void main(){
        char* str = "learnown";
        printf("%c\n", *str++ + 3);
        printf("%s\n", ++str+2);
}

#include<stdio.h>
void main(){
        char* str = "learnown";
        printf("%c\n" , *(str++ + 2)+3);
        printf("%c\n" , *++str+2);
        printf("%s\n" , --str-1);
}
```

```
#include<stdio.h>
void main(){
        char sport[ ]= "cricket";
        int x=1 , y;
        y=x++ + ++x;
        printf("%c",sport[++y]);
}
```

Array of Pointers: If the array is Pointer type, it can store multiple references called Array of Pointers.

```
#include<stdio.h>
int main()
{
        int arr[5] = {10, 20, 30, 40, 50}, i;
        int* ptr[5];
        for(i=0 ; i<5 ; i++){
                ptr[i] = &arr[i];
        }
        printf("Array elements are : \n");
        for(i=0 ; i<5 ; i++){
                printf("%d \n", *ptr[i]);
        }
}
```

We can access elements without using index as follows:

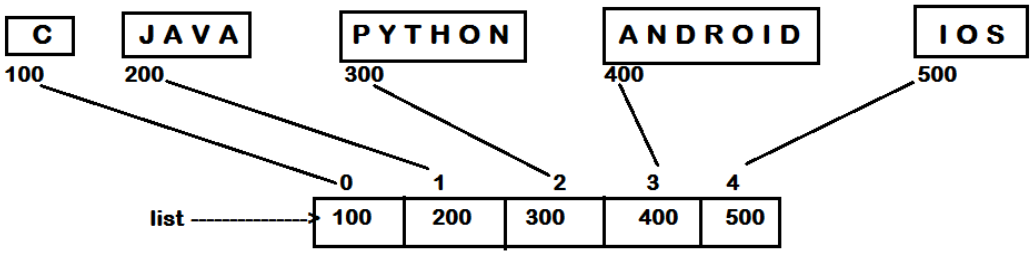
```
#include<stdio.h>
int main()
{
        int arr[5] = {10, 20, 30, 40, 50}, i;
        int* ptr[5];
        for(i=0 ; i<5 ; i++){
                *(ptr+i) = arr+i;
        }

        printf("Array elements are : \n");
        for(i=0 ; i<5 ; i++){
                printf("%d \n", **(ptr+i));
        }
        return 0;
}
```

A function can return the string using char* return type.

```
#include<stdio.h>
char* read(void);
int main(){
        char* name;
        name = read();
        printf("Name returned by read function is : %s \n", name);
        return 0;
}
char* read(void){
        char* name;
        printf("Enter your name : ");
        gets(name);
        return name;
}
```

Array of Strings:

- We can declare an array variable that stores list of pointers(strings base address).
- We can process all the strings using array variable.




```
#include<stdio.h>
int main()
{
        char* list[5] = {"C", "Java", "Python", "Android", "IOS"};
        int i;

        printf("List is : \n");
        for(i=0 ; i<5 ; i++)
        {
                printf("%s \n", list[i]);
        }
        return 0;
}
```

A function returning address:

A function can return address of aggregate data type variable or user defined data type variable to access the data.

```
#include<stdio.h>
int* add(int,int);
void main(){
        int a, b, *c;
        printf("enter two numbers : ");
        scanf("%d%d",&a,&b);
        c = add(a,b);
        printf("sum : %d\n",*c);
}
int* add(int x, int y) {
        int z;
        z=x+y;
        return &z;
}
```

Dangling pointer:

- In the above code, add function returns the address of local variable.
- Local variables will be deleted once the function execution completes.
- As we return the address of location, the value of that location may change by another function before processing the data.
- Hence Dangling pointers will not provide accurate results(values).

```
#include<stdio.h>
int *fun()
{
        int x = 5;
        return &x;
}
int main()
{
        int *p = fun();
        fflush(stdin);
        printf("%d", *p);
        return 0;
}
```

How to read complex pointers in C Programming?

Assign the priority to the pointer declaration considering precedence and associative according to following table.

Operator	Precedence	Associative
(),[]	1	Left to right
*, identifier	2	Right to left
Data type	3	

Where:

() : This operator behaves as function operator.

[] : This operator behaves as array subscription operator.

*** :** This operator behaves as pointer operator.

Identifier : it is name of pointer variable.

Data type : Data types also includes modifier (like signed int, long double etc.)

How to read following pointer?

char (* ptr)[3]

Step 1: () and [] enjoys equal precedence. So rule of associative will decide the priority. Its associative is left to right So first priority goes to ().

Step 2: Inside the bracket * and ptr enjoy equal precedence. From rule of associative (right to left) first priority goes to ptr and second priority goes to *.

Step3: Assign third priority to [].

Step4: Since data type enjoys least priority so assign fourth priority to char.

Module-4

(Functions, Recursion, DMA)

Functions in C

Variable: Stores data.
Function: Performs operation on data.

Function takes input, performs operations and returns output

Syntax	Example

int identity(arguments)
{
-> statements;
}
	int add(int a, int b)
	{
	int res = a+b ;
	return res ;
	}

C functions classified into:

1. Built In Functions
2. User Defined Functions

Built in Functions:

- C library is a set of header files
- Header file is a collection of pre-defined functions.

stdio.h	conio.h	string.h	graphics.h
printf()	getch()	strlen()	line()
scanf()	clrscr()	strrev()	circle()
feof()	cetche()	strcat()	bar()
…	….	…..	…

Custom Functions: Programmed defined functions based on application requirement

Calculator.c	Mobile.c	Account.c
add()	call()	deposit()
subtract()	message()	withdraw()
multiply()	store()	….
divide()	…..	

Every function consists

1. **Prototype:**
   a. Prototype is called function declaration. Every Custom function must be specified before main().
2. **Definition:**
   a. Definition is a block of instructions contains function logic. Function executes when we call that function.
3. **Function call:**
   a. It is a statement and it is used to execute the function logic.

No arguments & No return vals	With arguments & No return vals	With arguments & With return vals	No arguments & With return vals
void func(void) ;	void func(int, int);	float func(int);	char func(void);

void func(void)
{
logic ;
}
	void func(int a, int b)
	{
	logic ;
	}
		float func(int x)
		{
		float y = 2.3 ;
		return y ;
		}
			char func(void)
			{
			char ch = 'g' ;
			return ch ;
			}

func( ) ;	func(10 , 20);	float a = func(10);	char sym = func( ) ;

No arguments and No return values function:

```
#include<stdio.h>
void sayHi(void);
int main(void)
{
        sayHi();
        return 0 ;
}
void sayHi(void){
        printf("Hi to all \n");
}
```

With arguments and No return values function: (Even Number Program)

```
#include<stdio.h>
void isEven(int);
int main(void)
{
        int n;
        printf("Enter number : ");
        scanf("%d", &n);
        isEven(n);
        return 0 ;
}
void isEven(int num){
        if(num%2==0)
                printf("%d is Even \n", num);
        else
                printf("%d is not Even \n", num);
}
```

With arguments and No return values (Print ASCII value of given character):

```
#include<stdio.h>
void ascii_value(char);
int main(void){
        char sym ;
        printf("Enter symbol : ");
        scanf("%c", &sym);
        asciiValue(sym);
        return 0;
}
void asciiValue(char sym)
{
        printf("ASCII value is : %d \n", sym);
}
```

With arguments and With return values function: (addition of 2 numbers)

```
#include <stdio.h>
int add(int, int);
int main(){
        int a, b, res;
        printf("Enter two numbers : \n");
        scanf("%d%d", &a, &b);
        res = add(a,b);
        printf("%d + %d = %d\n", a, b, res);
        return 0;
}
int add(int x, int y){
        int z=x+y ;
        return z;
}
```

No arguments and With return values function:

```
float getPI(void);
void main(void){
        float pi;
        fpi = getPI();
        printf("PI value is : %f\n", pi);
}
float getPI(void){
        return 3.142;
}
```

Arithmetic Operations Menu Driven program – Using functions

```
#include <stdio.h>
float add(float,float);
float subtract(float,float);
float multiply(float,float);
float divide(float,float);
int main(){
        int choice;
        float a,b,c;
        while(1){
                printf("1.Add\n");
                printf("2.Subtract\n");
                printf("3.Multiply\n");
                printf("4.divide\n");
                printf("5.Exit\n");
                printf("Enter your choice : ");
                scanf("%d" , &choice);

                if(choice>=1 && choice<=4){
                        printf("Enter 2 numbers :\n");
                        scanf("%f%f", &a, &b);
                }

                switch(choice){
                        case 1 :        c = add(a,b);
                                        printf("Result : %f\n\n",c);
                                        break ;
                        case 2 :        c = subtract(a,b);
                                        printf("Result : %f\n\n",c);
                                        break ;
                        case 3 :        c = multiply(a,b);
                                        printf("Result : %f\n\n",c);
                                        break ;
                        case 4 :        c = divide(a,b);
                                        printf("Result : %f\n\n",c);
                                        break ;
                        case 5 :        exit(1);
                        default:        printf("Invalid choice \n\n");
        }
  }
  return 0;
```

```
}
float add(float x, float y){
        return x+y ;
}
float subtract(float x, float y){
        return x-y ;
}
float multiply(float x, float y){
        return x*y ;
}
float divide(float x, float y){
        return x/y ;
}
```

Recursion:

Function calling itself

or

Calling a function from the definition of same function.

```
#include<stdio.h>
void recur(void);
void main(){
        recur();
}
void recur(){
        printf("Starts \n");
        recur();
        printf("Ends \n");
}
```

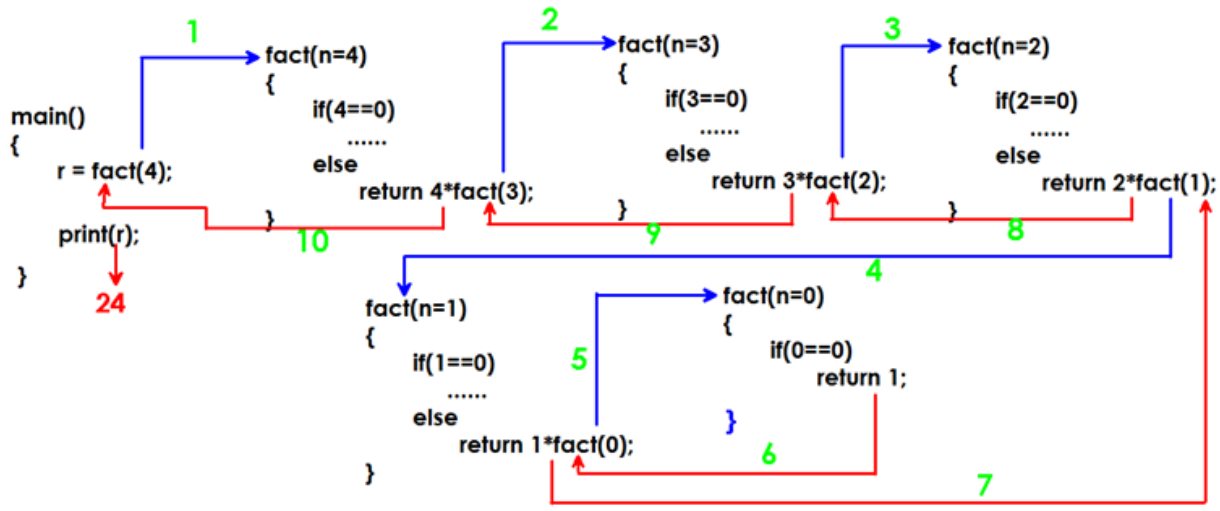
Program to print 1 to 10 numbers using recursion:

```
#include<stdio.h>
void print(int);
int main(void){
        print(1);
        return 0 ;
}
void print(int n){
        printf("%d \n", n);
        if(n<10)
                print(n+1);
}
```

Program to find the factorial of given number using recursion:

```
#include<stdio.h>
int fact(int);
void main(void)
{
        int n, factorial;
        printf("Enter one number : ");
        scanf("%d",&n);
        factorial = fact(n);
        printf("result : %d\n",factorial);
}
```

```
int fact(int n)
{
        if(n==0)
                return 1;
        else
                return n*fact(n-1);
}
```



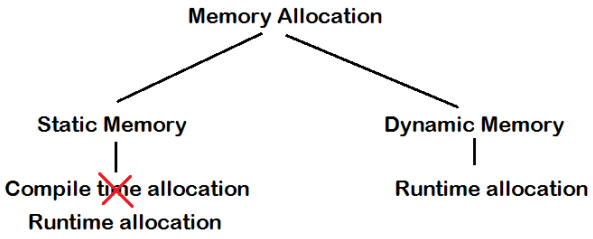
Program to find sum of first N numbers using Recursion:

```
#include<stdio.h>
int sum(int);
void main(void)
{
        int n, s;
        printf("Enter n value : ");
        scanf("%d", &n);
        s = sum(n);
        printf("Sum of %d nums : %d\n", n,
s);
}
```

```
int sum(int n)
{
        if(n==0)
                return n;
        else
                return n+sum(n-1);
}
```

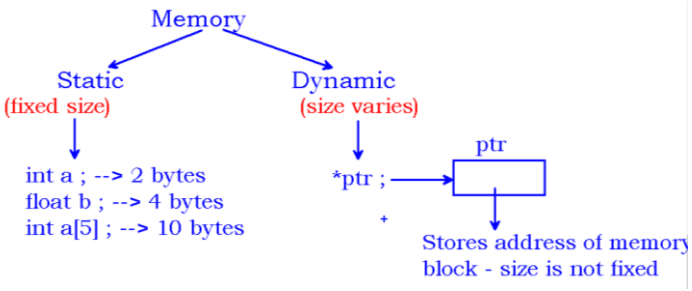
Dynamic Memory Allocation

- Applications are used to store and process information.
- We use variables to store the information.
- We allocate memory to variables in 2 ways
    - Static memory allocation
    - Dynamic memory allocation



Note : **To every program, memory allocates at runtime only.**
Compiler : is responsible for syntax checking and generate binary code.
Static memory : Just fixed in size - allocate at runtime only
Dynamic memory : Size varies - allocate at runtime only

Using pointers, we can create variables which are ready to hold addresses of memory blocks. The size of memory block specified at runtime through Dynamic memory allocation.



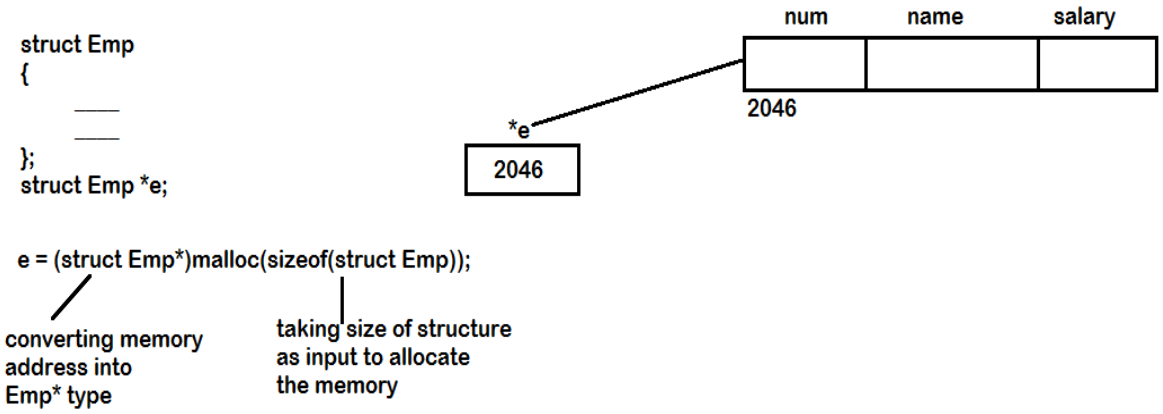
stdlib.h: stdlib.h library contains pre-defined functions to allocate and de-allocate the memory at runtime.

1. malloc(): allocates memory dynamically to structure variables
2. calloc(): allocates memory dynamically to array variables
3. realloc(): used to modify the size of Array created by calloc()
4. free(): used to release the memory which is allocated.

malloc() : Allocates numbers of bytes specified by the Size of Structure.

Prototype: void* malloc(size_t size);

- "size_t" represents "unsigned" value.
- On success, it returns base address of allocated block.
- On failure, it returns NULL.




```
#include<stdio.h>
#include<stdlib.h>
struct Emp
{
        int num;
        char name[20];
        float salary;
};
int main()
{
        struct Emp* p;
        p = (struct Emp*)malloc(sizeof(struct Emp));
        if(p==NULL)
                printf("Memory allocation failed \n");
        else
                printf("Memory allocation success \n");
        return 0;
}
```

- We access locations of structure memory using dot(.) operator.
- If a pointer is pointing to structure, we use arrow(->) operator to access the location.

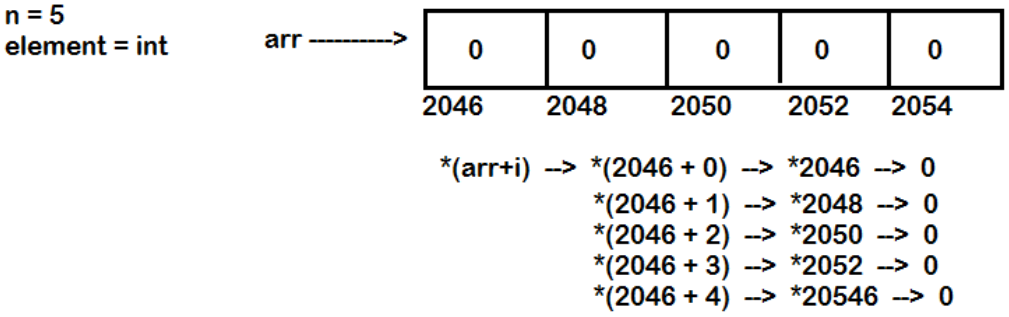
```
#include<stdio.h>
#include<stdlib.h>
struct Emp{
        int num;
        char name[20];
        float salary;
};
```

```
int main(){
        struct Emp* p;
        p = (struct Emp*)malloc(sizeof(struct Emp));
        if(p==NULL){
                printf("Memory allocation failed \n");
        }
        else{
                printf("Enter Emp details :\n");
                scanf("%d%s%f", &p->num, p->name, &p->salary);
                printf("Name is : %s \n", p->name);
        }
        return 0;
}
```

calloc() : Allocate memory to Arrays using pointers.

```
        void* calloc(size_t n , size_t size);
```

- First argument specifies number of elements in the array.
- Second argument specifies size of each element.
- If memory is available, it allocates n*size bytes and returns first byte address.
- If no memory, it returns NULL.
- After allocating the memory all the byte locations are initialized with "0" by default.




```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int n, *arr, i;

        printf("Enter size of array : ");
        scanf("%d", &n);

        arr = (int*)calloc(n, sizeof(int));
```

```
	if(arr==NULL){
		printf("Failed in memory allocation \n");
	}
	else{
		printf("Array elements are :  \n");
		for(i=0 ; i<n ; i++)
		{
			printf("%d \n", *(arr+i));
		}
	}
	return 0;
}
```

Read and find the sum of array elements :

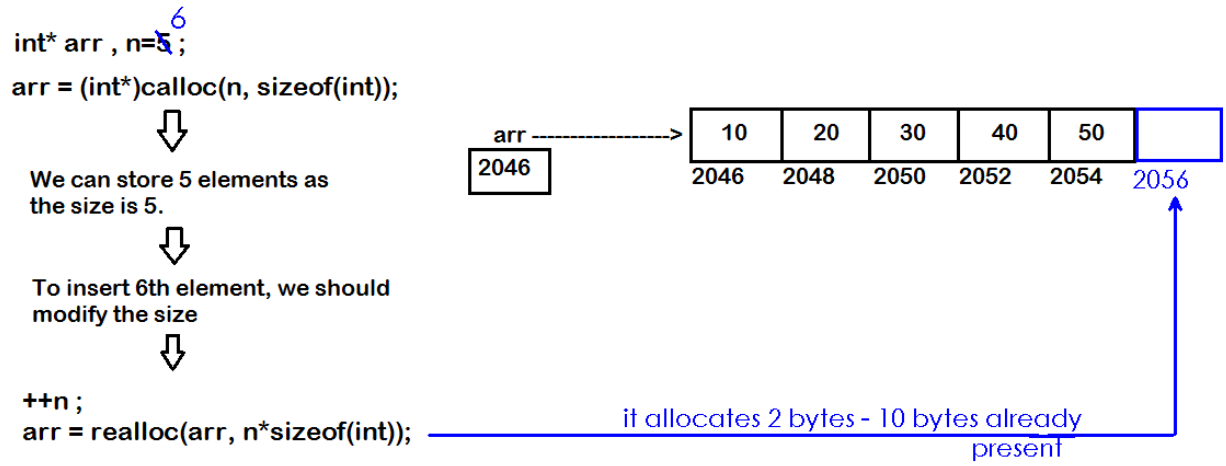
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
	int n, *arr, i, sum=0;

	printf("Enter size of array : ");
	scanf("%d", &n);

	arr = (int*)calloc(n, sizeof(int));
	if(arr==NULL){
		printf("Failed in memory allocation \n");
	}
	else{
		printf("Enter array elements :  \n");
		for(i=0 ; i<n ; i++)
			scanf("%d", arr+i);
		for(i=0 ; i<n ; i++)
			sum = sum + *(arr+i);

		printf("Sum is : %d \n", sum);
	}
	return 0;
}
```

realloc() :

- It is used to increase/decrease the size of array which is created by calloc()
- Using calloc(), we can create the size with fixed size

- Dynamic memory : Size varies depends on requirement
- We can modify the size of array using realloc() function

  void* realloc(void* ptr, size_t size);




```
#include<stdlib.h>
void main(){
        int *p1, *p2, m, n;
        printf("enter size of array : ");
        scanf("%d",&m);

        p1 = (int*)calloc(m,sizeof(int));
        if(p1 == NULL){
                printf("calloc failed\n");
                exit(1);
        }
        else
                printf("memory allocated..\n");

        printf("Enter new size of array : ");
        scanf("%d",&n);

        p2=(int*)realloc(p1 , n*sizeof(int));
        if(p2 == NULL){
                printf("realloc failed\n");
                exit(1);
        }
        else
                printf("memory re-aquired..\n");
        free(p1);
        free(p2);
}
```

Module-5

(Pre-Processor and File Handling)

File Handling in C

File: A collection of Data or Program or Records identified by a Name.

FILE* pointer:

- FILE is a pre-defined structure type variable.
- FILE* can points to any type of file.
- Once we make the pointer pointing to a file, then we can perform all FILE operations such as reading, writing, appending and so on.

Modes of Files:

Read mode("r"):

- Opens file in Read mode.
- If file is present, it opens and returns pointer to that file.
- If file is not present, it returns NULL pointer.

Write mode("w"):

- Opens file in Write mode.
- If file is present, it opens and removes the existing contents of File.
- If file is not present, it creates the new file with specified name.

Append mode("a"):

- Opens file in append mode.
- If file is present, it opens and places the cursor at the end of existing data to add the new contents.
- If file is not present, it creates the new file with specified name.

Opening a File:

- fopen() is a pre-defined function that opens a file in specified mode.
- On success, it opens and returns the pointer to file.
- On failure, it returns NULL pointer.

FILE* fopen(char* path, char* mode);

```
#include<stdio.h>
int main(){
        FILE* p;
        p = fopen("c:/users/srinivas/desktop/code.c", "r");
        if(p==NULL)
                printf("No such file to open \n");
        else
                printf("File opened in read mode \n");
        return 0;
}
```

Reading File character by character:

- fgetc() is a pre-defined function belongs to stdio.h
- fgetc() function read the input file character by character.
- On success, it reads the character and returns ASCII value of that character
- On Failure, it returns -1(EOF)

int fgetc(FILE* stream);

```
#include<stdio.h>
int main(){
	FILE* p;
	int ch;
	p = fopen("code.c", "r");
	if(p==NULL)
	{
		printf("No such file to open \n");
	}
	else
	{
		printf("File contents : \n");
		while((ch=fgetc(p)) != -1)
		{
			printf("%c", ch);
		}
	}
	return 0;
}
```

EOF (End Of File):

- It is pre-defined constant variable.
- EOF is a macro(constant)
- Constant variables represent in capital letters(in all programming languages)
- EOF is assigned with value -1
- EOF represents “End Of File” while processing file data.

```
#include<stdio.h>
int main(){
	printf("EOF value is : %d \n", EOF);
	printf("EOF character is : %c \n", EOF);
	return 0;
}
```

Closing the File: fclose() is used to close the file after use.

- It is recommended to release every resource(file) after use.
- On success, it returns 0.
- On Failure, it returns -1.

int fclose(FILE* stream);

We must release Resource (File, Database) after use. Improper shutdown causes loss of data.

```
int main(){
        FILE* p;
        int r1, r2;
        p = fopen("code.c", "r");
        if(p != NULL){
                r1 = fclose(p);
                printf("On success : %d \n", r1);
                r2 = fclose(p);
                printf("On failure : %d \n", r2);
        }
        return 0;
}
```

Writing character into File:

fputc(): A pre-defined function is used to write character into file.

void fputc(int x, FILE* stream);

```
int main(){
        FILE *src, *dest;
        int ch;
        src = fopen("code.c", "r");
        dest = fopen("output.txt", "w");
        if(src != NULL){
                while((ch = fgetc(src)) != EOF){
                        fputc(ch, dest);
                }
                printf("Data copied...\n");
                fclose(dest);
                fclose(src);
        }
        return 0;
}
```

fseek(): It is used to move the cursor position in the opened file, to perform read and write operations.

int fseek(FILE* steam, long offset, int origin);

Parameters:

- "offset" is a long type variable that represents the number of bytes to move.
  - Positive offset moves the cursor in forward direction
  - Negative offset moves the cursor in backward direction.
- Origin comes with
  - SEEK_SET : Start of the file
  - SEEK_CURR : Current cursor position
  - SEEK_END : End of the file
- On success, if specified location is present, it moves the cursor and returns 0
- On failure, it returns -1.

```
#include<stdio.h>
int main(){
	FILE* p;
	int ch;
	p = fopen("code.c", "r");
	if(p==NULL){
		printf("No such file to open \n");
	}
	else{
		fseek(p, 100, SEEK_SET);
		while((ch=fgetc(p)) != -1){
			printf("%c", ch);
		}
		fclose(p);
	}
	return 0;
}
```

rewind(): Reset the cursor position to start of the file.

void rewind(FILE* stream);

```
#include<stdio.h>
int main(){
	FILE* p;
	int ch;
	p = fopen("code.c", "r");
	if(p==NULL){
```

```
                printf("No such file to open \n");
        }
        else{
                fseek(p, -50, SEEK_END);
                while((ch=fgetc(p)) != -1){
                        printf("%c", ch);
                }

                rewind(p);
                while((ch=fgetc(p)) != -1){
                        printf("%c", ch);
                }
                fclose(p);
        }
        return 0;
}
```

Reading String from the file:

fgets() is used to read specified number of characters(String) at a time.

char* fgets(char* str, int size, FILE* stream);

- It reads “size” bytes from specified “stream” and stores into “str”.
- On Success, it returns the pointer to string that has read.
- On failure, it returns NULL pointer.
- It reads only size-1 characters into String every time. Last character of every string is null(\0) by default.

```
#include<stdio.h>
int main(){
        FILE* p;
        char str[10];
        p = fopen("code.c", "r");
        if(p==NULL){
                printf("No such file to open \n");
        }
        else{
                fgets(str, 10, p);
                printf("The string is : %s \n", str);
                fclose(p);
        }
        return 0;
}
```

Reading the complete file using fgets():

```
#include<stdio.h>
int main(){
        FILE* p;
        char str[10];
        p = fopen("code.c", "r");
        if(p==NULL){
                printf("No such file to open \n");
        }
        else{
                while(fgets(str, 10, p)){
                        printf("%s \t", str);
                }
                fclose(p);
        }
        return 0;
}
```

Writing String into File:

fputs() is used to write a string on to the file.

void fputs(char* str , FILE* stream);

```
#include<stdio.h>
int main(){
        FILE *src, *dest;
        char str[10];
        src = fopen("code.c", "r");
        if(src==NULL){
                printf("No such file to open \n");
        }
        else{
                dest = fopen("data.txt", "w");
                while(fgets(str, 10, src)){
                        fputs(str, dest);
                }
                printf("Contents copied...\n");
                fclose(src);
                fclose(dest);
        }
        return 0;
}
```

feof():

- It used to test whether End of File has reached or not
- It returns -1 when End of File has reached else returns 0

int feof(FILE* stream)

Reading File Character by Character:

```
#include<stdio.h>
int main(){
        FILE *src;
        int ch;
        src = fopen("code.c", "r");
        if(src==NULL){
                printf("No such file to open \n");
        }
        else{
                while(!feof(src)){
                        ch = fgetc(src);
                        printf("%c", ch);
                }
                fclose(src);
        }
        return 0;
}
```

Renaming the File:

- **rename()** function is used to rename the specified file.
- On success, it returns 0
- On failure, it returns -1

int rename(char* old, char* new);

```
#include<stdio.h>
int main(){
        char old[20] = "data.txt";
        char new[20] = "modified_data.txt";

        if(rename(old, new)==0)
                printf("successfully renamed...\n");
        else
                printf("Failed in renaming the file...\n");
        return 0;
}
```

Remove the file from specified location:

- **remove()** is used to rename the specified file.
- On success, it returns 0
- On failure, it returns -1

```
int remove(char* target);
```

```
#include<stdio.h>
int main()
{
	char target[20] = "data.txt";

	if(remove(target)==0)
		printf("successfully removed...\n");
	else
		printf("Failed in removing the file...\n");
	return 0;
}
```

Enumeration in C

Introduction:

- An enumeration consists of a set of named integer constants.
- Each enumeration-constant in an enumeration-list names a value of the enumeration set.

Syntax:

```
enum identifier
{
        enumerator-list
};
```

By default, the first enumeration-constant is associated with the value 0.

```
enum colors
{
        black,
        red,
        green,
        blue,
        white
};
int main()
{
        int i;
        enum colors c;
        for(i=0 ; i<5 ; i++)
        {
                printf("%d \n", c=i);
        }
        return 0;
}
```

If we don't set values explicitly, by default values set in increasing order by one:

```
#include<stdio.h>
int main()
{
        enum months {Jan=1, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};
        enum months month;
        printf("month=%d\n",month=Feb);
        return 0;
}
```

By using constant integer values, we can use the functionality of enum set elements:

```
#include<stdio.h>
enum Days
{
        Sunday,
        Monday,
        Tuesday,
        Wednesday,
        Thursday,
        Friday,
        Saturday
};
int main()
{
        enum Day today;
        int x;
        printf("Enter Day of Week(0 to 6) : ");
        scanf("%d",&x);
        today=x;
        if(today==Sunday || today==Saturday)
                printf("Enjoy! Its the weekend\n");
        else
                printf("Week day - do your work\n");
        return 0;
}
```

Typedef in C

Introduction:

- typedef is a user-defined data type specifier.
- Using typedef we can define identifiers (synonyms) for data types.
- We can use synonyms instead of original data types.

Advantage: Synonyms to data types become more readable than complex identities

Giving the name Integer to int type:

```
#include<stdio.h>
int main()
{
        typedef int Integer ;
        Integer a=10,b=20,c;
        c = a+b;
        printf("res : %d\n",c);
        return 0;
}
```

Creating Synonym to Array:

```
#include<stdio.h>
int main()
{
        typedef int Arry[5];
        int i;
        Array arr = {10,20,30,40,50};
        printf("Array elements are\n");
        for(i=0 ; i<5 ; i++)
        {
                printf("%d\n", arr[i]);
        }
        return 0;
}
```

The main advantage of typedef declaration is giving simple identity for complex types.

For example, char* can simply represent with name String.

```
#include<stdio.h>
typedef char* String;
String read(void);
int main(){
```

```
        String name;
        name = read();
        printf("welcome %s\n",name);
}
String read(){
        String name;
        printf("Enter one name : ");
        gets(name);
        return name;
}
```

Typedef declaration to Structures:

- Generally, identity of Structure is a combination of two words.
- For example, struct Emp
- We can give one word to structure type then representation become easy.

```
#include<stdio.h>
typedef struct Emp{
        int eno;
        char* ename;
        float esal;
} Employee;
int main()
{
        Employee e;
}
```

```
#include<stdio.h>
struct Emp{
        int eno;
        char* ename;
        float esal;
};
int main()
{
        typedef struct Emp Employee;
        Employee e;
}
```

By using typedef declarations, we can shorten the size of instructions:

```
#include<stdio.h>
struct Emp{
        int eno;
        char* ename;
        float esal;
};
int main()
{
        typedef struct Emp Employee;
        typedef struct Emp* Ptr;
        Ptr e;
        e = (Ptr)malloc(sizeof(Employee));
}
```

Storage classes in C

Introduction:

- We need to specify the storage class along with datatype in variable declaration.
- Storage class describes
  - Memory location of variable
  - Default value of variable
  - Scope of variable
  - Lifetime of variable
- Storage classes classified into
  - Automatic Storage classes
  - Register Storage classes
  - Static Storage classes
  - External Storage classes

Syntax:

storage_class datatype name ;

Example:

```
auto int a ;
static int b ;
```

Block scope of a variable:

- Defining a variable inside the block.
- We can access that variable from the same block only.
- We cannot access from outside of the block or from other block.

```
#include<stdio.h>
int main()
{
        {
                int a=10;
                printf("a val : %d \n", a);
        }


        {
                int b=20;
                printf("b val : %d \n", b);
                printf("a val : %d \n", a);          // Error :
        }
        return 0;
}
```

Function scope of a variable:

- Defining a variable inside the function.
- We can have number of blocks inside the function.
- Function scope variable can access throughout the function and from all block belongs to that function.

```
#include<stdio.h>
int main()
{
        int a=10;  /* function scope */

        {
                int a=20; /* block scope */
                printf("Inside First block, a val : %d \n", a);
        }


        {
                printf("Inside Second block, a val : %d \n", a);
        }

        printf("Inside function, a val : %d \n", a);
        return 0;
}
```

Note: If we don't provide any value to local variable, by default the value is "Garbage value"

```
#include<stdio.h>
int main()
{
        int a;
        {
                int a=10;
                printf("a val : %d \n", a);
        }

        {
                printf("a val : %d \n", a);
        }

        return 0;
}
```

Program scope:

- Defining a variable outside to all function
- We can also call it as Global variable
- We can access the global variable throughout the program.

```
#include<stdio.h>
void test(void);
int a=10; /*program scope*/
int main(){
	int a=20; /*function scope */
	{
		int a=30; /*block scope */
		printf("In block : %d \n", a);
	}
	printf("In main : %d \n", a);
	test();
	return 0;
}
void test(){
	printf("In test : %d \n", a);
}
```

Automatic Storage classes:
Keyword: auto
Memory location: Inside the RAM
Default value: Garbage Value
Scope: Belongs to Block in which it has defined.

Note: auto variables must be local. If we don't give storage class to local variable, it is auto default.

```
#include<stdio.h>
int main(){
	auto int a=10;
	{
		auto int a;
		printf("In block : %d \n", a);
	}
	printf("In main : %d \n", a);
	return 0;
}
```

We cannot define auto variables globally:

```
#include<stdio.h>
auto int a=30;   /* Error :  */
int main(){
        auto int a=10;
        printf("In main : %d \n", a);
        return 0;
}
```

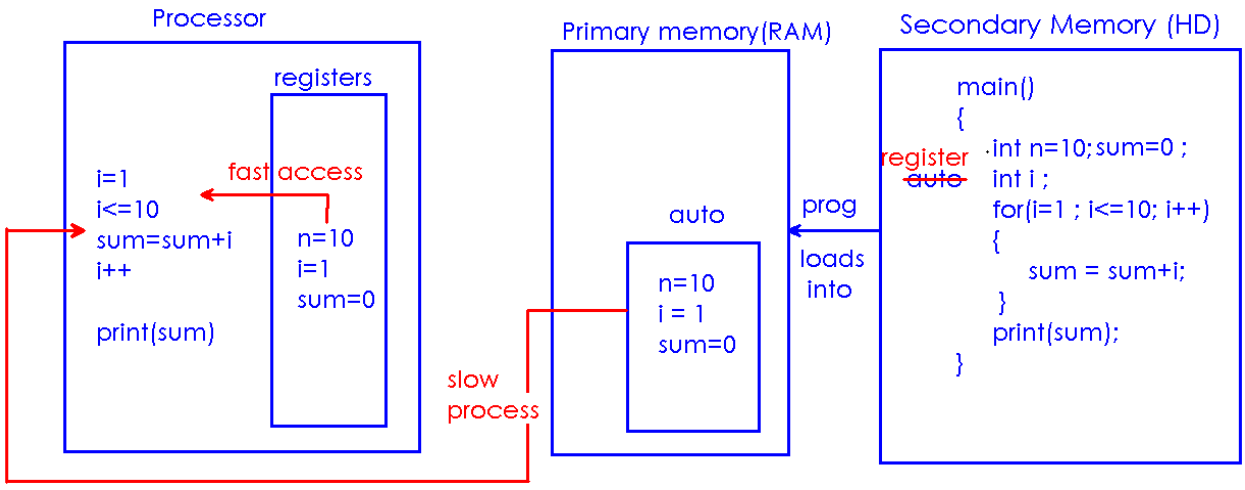
Register storage classes:
Keyword: register
Memory allocation: Inside CPU registers
Scope: Within the block or method

Note: Register variable must be defined as local variable.

Auto v/s Register variables:

- "auto" variables get memory in RAM
- "register" variables get memory inside Registers.
- "Register" variables can access much faster than "auto" variables.

Note: Repeated variables in the logic must be defined as register variables. For example "Loop counters"



Can't we declare all variables as register?

- No, Registers is a small memory area. Hence, we cannot store maximum number of variables into registers.
- Only local variables and repeated variables need to store into registers.

We cannot define register variables globally:

```
#include<stdio.h>
register int a=30;
int main()
{
        auto int a=10;
        printf("In main : %d \n", a);
        return 0;
}
```

Static storage classes:
Keyword: static
Memory: Inside RAM
Scope: Static variables can be either local or global. If the variable is local, can access within the block. If it is global, we can access throughout the program.
Default value: 0

```
#include<stdio.h>
static int x=10;
void test();
int main(){
        static int y;
        printf("y val : %d \n", y);
        test();
        return 0;
}
void test()
{
        printf("x val : %d \n", x);
}
```

Local variables memory will be deleted as soon as function execution completes.

```
#include<stdio.h>
void test();
int main()
{
        test();
        test();
        test();
        test();
        return 0;
}
```

```
void test()
{
        auto int x=5;
        x=x+2;
        printf("x val : %d \n", x);
}
```

Static variable memory will be deleted only when program execution completes.

```
#include<stdio.h>
void test();
int main()
{
        test();
        test();
        test();
        test();
        return 0;
}
void test(){
        static int x=5;
        x=x+2;
        printf("x val : %d \n", x);
}
```

External storage classes:
Keyword: extern
Memory: Inside RAM
Scope: Can access anywhere in the application

Note: External variables always global variables. These variables get memory only when we initialize with any value.

External variables cannot be global:

```
#include<stdio.h>
int main()
{
        extern int a=10;
        printf("a value is : %d \n", a);
        return 0;
}
```

Global variables must be initialized with value. Runtime error raises if we access external variable without value.

```
#include<stdio.h>
extern int a;
int main()
{
	printf("a value is : %d \n", a);
	return 0;
}
```

Memory allocates only when we initialize:

```
#include<stdio.h>
extern int a=10;
int main()
{
	printf("a value is : %d \n", a);
	return 0;
}
```

The declaration statement will not consider, memory allocates only when we assign value:

```
#include<stdio.h>
extern int a;
extern int a;
extern int a;
extern int a=10;
int main()
{
	printf("a value is : %d \n", a);
	return 0;
}
```

Command Line Arguments in C

Accepting Command Line arguments:

Every C application runs by Computer Operating System. OS runs the program from its environment. Every OS consists CUI and GUI mode. To read command line arguments, we need to run the C program from CUI OS mode.

Run application from Command prompt:

- Cmd Prompt is Ms-DOS
- It is Command User Interface.
- OS takes command as input and open specified application which is present in that machine.
- We compile and run the code from command line using tcc compiler as follows

Code program:

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

Note: save into "turboc" folder with name Program.c

Open command prompt:

Compilation:

C:/turboc> tcc Program.c

-> Generate Program.exe file

Execution:

C:/turboc>Program.exe

-> Generate output.

main() prototype: Regular main() function prototype is slightly different from the main() function taking command line arguments. We must define main() function as follows

int main(int argc , char* argv[ ])

where:

'argc' = argument count (Number of arguments passed at command line including file name)

'argv' = argument vector(list) (List of arguments - default type is String)

Why argument type is String by default?

Answer: In any programming language, only String type can hold any data.

main(int[]) --> can take only integers

main(float[]) --> can take only float values.

main(char* []) --> can hold any type of data.

```
char* a = "10";
char* b = "23.45";
char* c = "g";
char* d = "CBook";
```

How can we convert String to corresponding primitive type?

We collect the input in String format from command line. dos.h header file is providing functions for this data conversions. The two functions used for data conversions as follows

1. **int atoi(char* s) :** Converts input string to integer on success
2. **float atof(char* s) :** Converts input string to float on success

Program to display Arguments count:

```
#include<stdio.h>
int main(int argc , char* argv[]){
        printf("Count is : %d \n", argc);
        return 0;
}
```

```
cmd/> Program
Output : Count is : 1

cmd/> Program 10 20 30 40 50
Output : Count is : 6

cmd/> Program 10 23.45 g online
Output : Count is : 5
```

Program to display list of input arguments:

```
#include<stdio.h>
int main(int argc , char* argv[]){
        int i;
        if(arc==1){
                printf("No input values \n");
        }
        else{
                printf("Arguments are :\n");
```

```
                for(i=1 ; i<argc ; i++){
                        printf("%s\n", argv[i]);
                }
        }
        return 0;
}
```

cmd/> Program
Output : No input values

cmd/> Program 10 23.45 g online
Arguments are :
10
23.45
g
online

Reading input from command line and add:

```
#include<stdio.h>
int main(int argc , char* argv[]){
        int i;
        if(arc<3){
                printf("Insufficient input values \n");
        }
        else{
                char* s1 = argv[1]);
                char* s2 = argv[2]);

                int x = atoi(s1);
                int y = atoi(s2);
                printf("Sum is : %d \n" , x+y);
        }
        return 0;
}
```

cmd/> Program
Output : Insufficient input values

cmd/> Program 10 20
Output : Sum is : 30

Pre-Processor in C

Introduction:

- Pre-processor program that modifies the text of a C program before it is compiled.
- Pre-processing includes
    - Inclusion of other files
    - Definition of symbolic constants and macros
    - Conditional compilation of program code
    - Conditional execution of preprocessor directives

Preprocessing, Compiling and Linking:

one.c --> PREPROCESSOR -> tmp.c(temporary) -> COMPILER -> one.obj -> LINKER -> one.exe (final Executable)

Inclusion of header files:

- C application is a collection of Source files(.c) and Header files(.h)
- #include directive is used to connect the files in C application

We include Library file as well as Custom defined Files:

#include <filename>

- Searches standard library for file
- Use for standard library files

#include "filename"

- Searches current directory, then standard library
- Use for user-defined files

if you have a header file 'header.h' as follows,

```
int x;
int biggest(int x, int y)
{
        retrun x>y?x:y;
}
```

and a main program called 'program.c' that uses the header file, like this,

```
#include "header.h"
int main (void)
{
        x=biggest(10,20);
        printf("big :%d",x);
        return 0;
}
```

Macros(#define):

- We use #define to create Macro constants.
- Constants recommended to define in Upper case.
- At the time or pre-processing all Macros replaced with their values.

Syntax:

#define identifier replacement-text

Example:

#define PI 3.14159

Function-like Macros

- You can also define macros looks like a function call.
- For example,

  #define CIRCLE_AREA(x) (PI * (x) * (x))
- Would cause

  area = CIRCLE_AREA(4);
- To become

  area = (3.14159 * (4) * (4));

Conditional Compilation (#if, #ifdef, #ifndef, #else, #elif, #endif)

Six directives are available to control conditional compilation. They delimit blocks of program text that are compiled only if a specified condition is true.

#if directive:

It is conditional compilation directive. If the condition is valid then the code defined inside the block gets compiled.

Syntax:

```
#if <Constant_expression>
  -------------
   -------------
#endif
```

If constant expression will return 0 then condition will FALSE if it will return any non-zero number condition will TRUE.

```
#include<stdio.h>
#if 0
        int main(){
                printf("HELLO WORLD");
        return 0;
}
```

```
#endif
```

Runtime Error: undefined symbol main

#else directive: The code which is defined in else-block gets compiled if the condition fails.

```
#include<stdio.h>
#if(-4)
int main(){
        printf("WELCOME TO C-WORLD ");
        return 0;
}
#else
        int main(){
                printf("HELLO WORLD");
        return 0;
}
#endif
```

Output: WELCOME TO C-WORLD

```
int main()
{
        int var = 5;
        #if var
                printf("%d",var);
        #else
                printf("%d",var);
        #endif
}
```

Compile time Error: Directive #if will not think expression constant var as integer variable and also it will not throw an error. Then what is var for directive #if?

Directive #if will treat var as undefined macro constant.

```
#include<stdio.h>
#define var 10
int main(){
        #if var
                printf("%d",var);
        #else
                printf("%d",var);
        #endif
        return 0;
}
```

Program to display System date using pre-defined Macro constant:

```
#include<stdio.h>
int main(){
 #ifdef __DATE__
        printf("%s",__DATE__);
    #else
        printf("First define the __DATE__");
    #endif
return 0;
}
```

Output: It will print current system date.

Explanation: __DATE__ is global identifier. It has already defined in the header file stdio.h and it keeps the current system date.

Program to display System Time using pre-defined Macro

```
#include<stdio.h>
#define I 30
int main()
{
        #ifndef __TIME__
                printf("%d",I);
        #else
                printf("%s",__TIME__);
        #endif
        return 0;
}
```

Output: It will print current system time.

Explanation: __TIME__ is global identifier. It has been defined in the header file stdio.h. Compiler doesn't compile the c codes which are inside the any conditional pre-processor directive if its condition is false. So we can write anything inside it.