Date :
Page No :

Principles of Programming using C
(BPOPS203)

MODULE 1. INTRODUCTION TO C

A computer can be defined as an electronic device that is designed to accept data, perform the required mathematical and logical operations at high speed and give output result.

Characteristics of computer :

↳ Speed: Computer can perform millions of operations per second. The speed of computers is usually given in nanoseconds and picoseconds.

↳ Accuracy: A computer gives accurate results, provided the correct data and set of instructions are input to it. If the input data is wrong, then the output will also be erroneous.

↳ Automation: Besides being very fast and accurate computers are automable devices that can perform a task without any user intervention.

↳ Diligence: Computers never get tired of a repetitive task. It can continually work for hours without creating errors.

↳ Versatile: Today, computers are used in our daily life in different fields. They are versatile devices as they can perform multiple tasks of different nature at the same time.

↳ Memory: Computers also have internal or primary

memory (RAM) as well as external or secondary memory. While the internal memory of computers is very expensive and limited in size, the secondary storage is cheaper and of bigger capacity.

↳ NO IQ : Although the trend today is to make computers intelligent by inducing AI in them, they still do not have any decision-making abilities of their own.

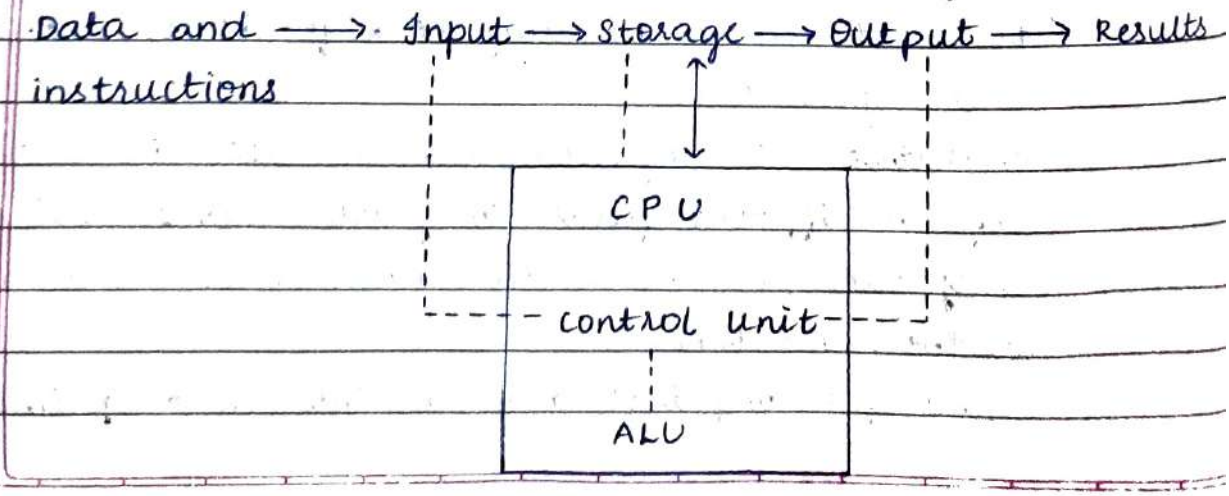
↳ Economical : Using computers reduces manpower requirements and leads to an elegant and efficient way of performing various tasks. Hence computers save time, energy and money.

Basic organization of a computer

Computer performs the five major operations

↳ Accepting data or instructions
↳ Storing data
↳ Processing data
↳ Displaying results
↳ Controlling and coordinating all operations inside a computer.

Block diagram of a computer



Date :
Page No :

↳ Input : This is the process of entering data and instructions into the computer system. The data and instructions can be entered by using different input devices, which also converts the input data to binary code.

↳ Storage : It is the process of saving data and instructions permanently in the computer so that they can be used for processing. There are 2 types of storage areas :
Primary storage - Also known as main memory is the storage area that is directly accessible by the CPU at high speeds. It is volatile and expensive.
Secondary storage - Also known as auxiliary memory. It is cheaper, non-volatile and used to permanently store data and programs permanently.

↳ Output - It is the process of giving the result of data processing to the outside world. The results are given through output devices such as monitor and printer.

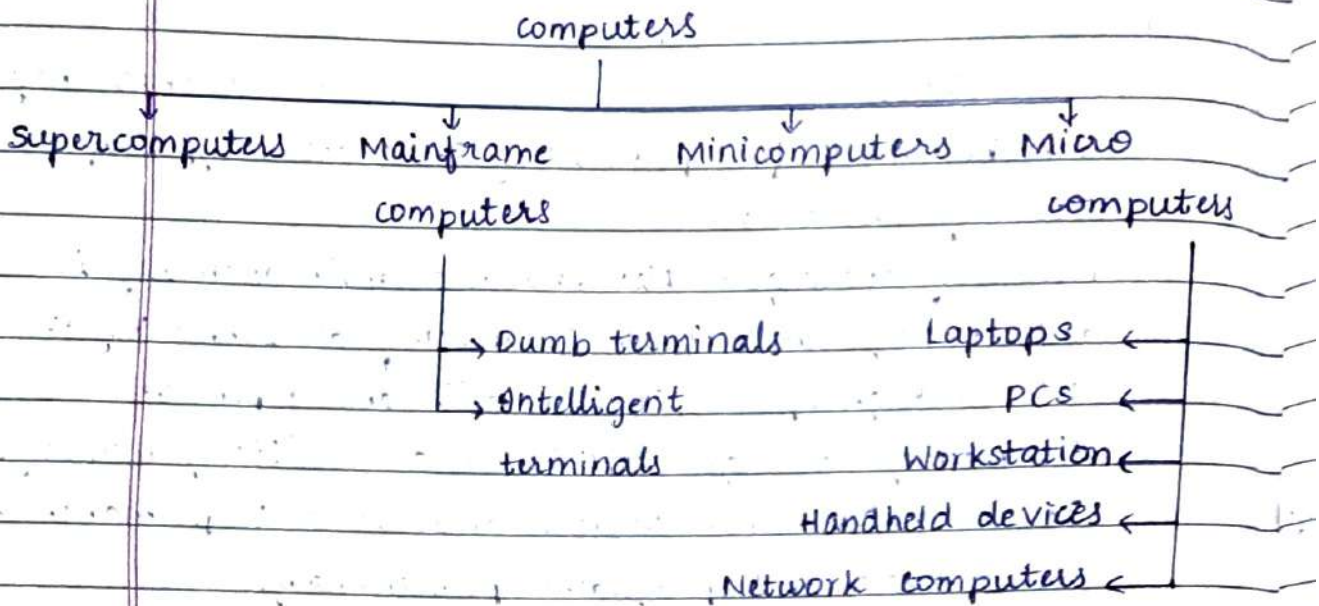
↳ Control : The control unit (CU) is the central nervous system of the entire computer system. It manages and controls all the components of the computer. CPU is a combination of ALU and CU.

↳ Processing : The process of performing operations on the data as per the instructions specified by the user (program) is called processing.

Date :
Page No :

Classifications of computers

Computers can be broadly classified into four categories based on the speed, amount of data they can process and price.



↳ Supercomputers:

They are the most fastest, expensive and powerful computers. They have parallel processing technology and can supports thousands of users at a time. They are mainly used for weather forecasting, aircraft design, automotive design etc.

Ex: CRAY-1, CRAY-2

↳ Mainframe computers

They are largescale, expensive computers. They support multi-processors. Users can access mainframes by either using terminals like dumb or intelligent terminals or via PCs.

Dumb terminals: It consists of only a monitor and a keyboard. They do not have their own CPU.

Intelligent terminals : They have their own processor and can perform some processing operations.

↳ Minicomputers :

They are smaller, cheaper and slower than mainframes. They were the smallest computer of all times. They are widely used in business, education, hospitals, govt. organisations etc.

↳ Microcomputers

Microcomputers are commonly known as PCs are very small and cheap. The first microcomputer was designed by IBM in 1981 and was named IBM-PC.

Desktop PCs -

Most popular model of PCs. It is widely used in homes and offices.

Laptops -

Laptops are small microcomputers that can easily fit inside a briefcase. They are very handy and can easily be carried from one place to another.

- Applications of computers

↳ Word processing : This software enables users to read and write documents, users can also add images, tables and graphs for illustrating a concept.

↳ Internet : It is a network of networks that connects computers all over the world. It gives the user access to an enormous amount of information.

↳ Digital video or audio composition : Computers make audio or video composition and editing very simple.

Date :
Page No :

↳ Desktop publishing : It is a software enables us to create page layouts for entire books.

↳ Electronic banking : Also known as cyber banking, supports various banking activities conducted from home, a business or on the road.

Stored Program concept

↳ Introduced by Sir John von Neumann

↳ All digital computers are based on the principle of stored program concept.

↳ Features are given below:

> First, instructions are read into memory

> Instructions are stored in the memory in binary form

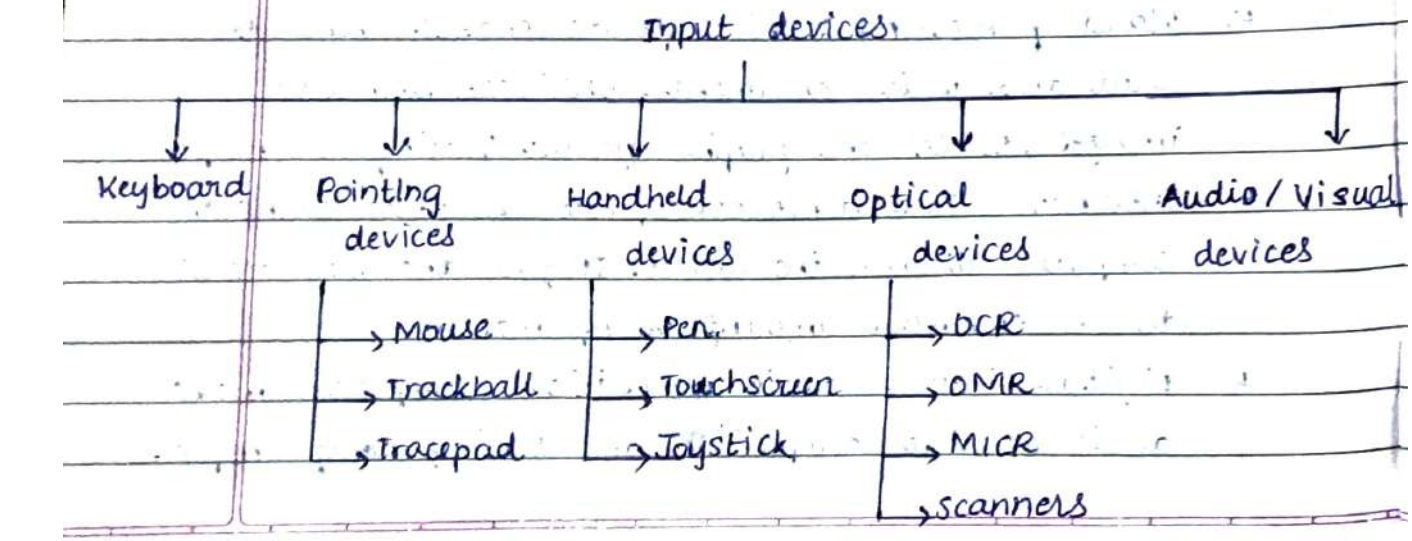
> Processing starts with the first instructions

> Instructions written by the users are done sequentially

> Input / Output and processing operations are performed simultaneously

Chapter 2

Input devices : It is used to feed data and instructions into the computer.



Date :
Page No :

↳ Keyboard :

The keyboard is the main input device for computers. Computer keyboards look very similar to the keyboards of typewriters. The standard keyboards consists of 101 keys.

↳ Pointing Devices:

It enables the users to easily control the movement of the pointer to select items on a display screen, to select commands from commands menu, to draw graphs etc. Examples are Mouse, Trackball, Light Pen.

↳ Handheld Devices :

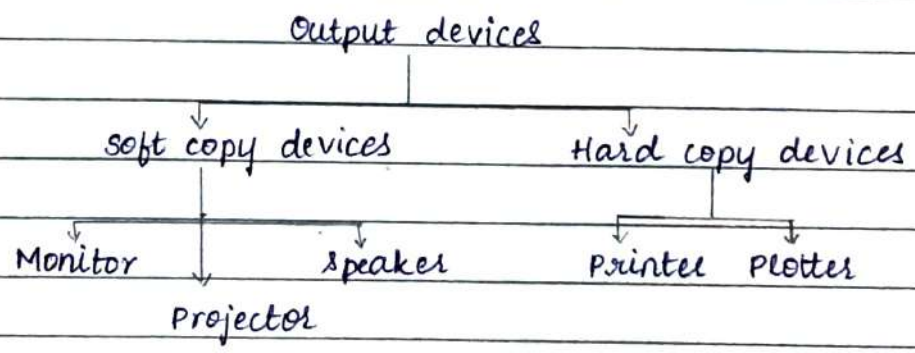
It is a pocket-sized computing device with a display screen and touch input and/or a miniature keyboard. Examples are smartphones, PDAs, iPOD.

↳ Optical Devices :

It converts these objects into digital data and send it to the computer for further processing. Some optical devices include barcode readers, image scanners, OCR devices, OMR etc.

↳ Audio-visual devices :

Today, all computers are multimedia-enabled, i.e, computers not only allow one to read or write text, but also enable the users to record songs, view animated movies etc.



Date :
Page No :

↳ Soft copy devices : These devices produce an electronic version of an output.
For example, a file that is stored on a hard disk, CD, or pen drive and is p displayed on the comp screen.

↳ Hard copy devices : These devices produce a physical form of output. For example, the content of a file printed on paper is a form of hard copy output.

↳ Monitor : It is an integral part of computer which displays both texts and graphics.

↳ Printers : It produces the hard copy of output.

Printer

- Impact printer
  - Dot matrix
  - Line
  - Daisy wheel
- Non-impact printer
  - Inkjet
  - Laser

Date :
Page No :

Chapter 4: Introduction to C.

↳ Structure of C program

```
Documentation section
Link section
Definition section
Global declaration section
main() section

{
    Declaration part
    Executable part
}
sub program section
Function 1
Function 2
    -
    -
Function n
```

- Documentation section : It consists of a set of comment line giving the name of the program, the author and other details. Compiler ignores these comments. There are two formats:
  i) Line comment // Single comment
  ii) Block comment /* Multi line comment */
- Link section : It provides instructions to the compiler to link functions from system library.

Date :
Page No :

- Definition section: It defines all symbolic constants.
- Global declaration section : The variables which are used in more than one function are called global variables and are declared in this section.
- main() section : Every C program has one main() function and it contains two parts:

i) Declaration Part : Here, all the variables are declared.

ii) Executable Part : Here, it contains at least one executable statement.

The main() function is enclosed within flower brackets and statement ends with semi-colon.

- Sub-program section : It contains all user-defined functions that are called by main() function.

Example :

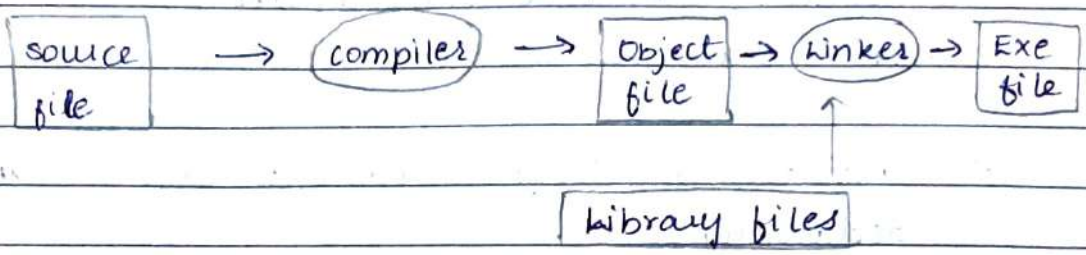
```
// Addition of two numbers
#include <stdio.h>
void main()
{
    int a,b, sum;
    printf ("Enter two numbers \n");
    scanf ("%d %d", &a, &b);
    sum = a+b;
    printf ("The sum is %d", sum);
}
```

Date :
Page No :

↳ Files used in C program:

- Source file
- Object file
- Header file
- Executable file

↳ Overview of compilation and execution of program:



↳ Types of tokens:
Keywords, strings, Operators, special symbols, constants, Indentifiers.

↳ Identifier is the name given to identify data and other objects in a program like variable, function, arrays etc..
Rules:

- It cannot include any special symbol or punctuation marks (except underscore).
- It cannot be having a space.

↳ Data types in C program:

- Basic data type / Fundamental / Built-in
- User-defined data type
- Derived data type

Basic data type : integer, float, double, character, void.

Date :
Page No :

↳ Variable : It is defined as name given to a data storage location in computer memory.

Rules to name a variable:
(Identifier rules)

Declaration of variable :
Syntax : data type variable_list ;
Ex : int a;

↳ Initialization : Assigning value to variable during declaration.
Syntax : data type var name = value
Ex : int a=10;

↳ Constant : value doesn't change

↳ Input output statements in C.
Formatted input statement :
• scanf : used to read the data that has been arranged in particular format
Syntax : scanf (" control string", arg1, arg2 ... argn);
↳ field specification

Formatted output statement :
• printf : used for printing the caption and results
Syntax : printf ("control string", arg1, arg2 ... argn);
↳ caption, field, white space characters

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

Date :
Page No :

MODULE 2. OPERATORS AND DECISION CONTROL STATEMENTS

Operators: It is a symbol that tells the compiler to perform specific mathematical and logical operations.

↳ Types of operators: Arithmetic, Relational, Logical, Assignment, Bitwise, unary, conditional operators, special.

↳ Arithmetic operators → +, -, *, /, %

↳ Relational operators → >, <, <=, >=, ==, !=

↳ Logical operators → compare more than one condition: NOT, AND, OR

↳ Assignment operators → =, +=,

↳ Bitwise operators → Bitwise NOT, Bitwise AND, Bitwise OR Bitwise left shift

X 1 0 1 1 0 0 0 1

X<<3 1 0 0 0 1 0 0 0

↳ Unary operators → ++, --

```
a = 15;          a 15
b = a++;         b 14
c = --a + 3;     c 18
d = a+b--        d 30
```

↳ conditional operators

↳ Write a program to redd 2 numbers, find out the greatest of two numbers using ternary operators

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

```
main()
{
int a,b,c;
printf("Enter two numbers");
scanf("%d%d",&a,&b);
c= (a>b)? a:b;
printf("Greatest is" = %d", c);
```

↳ special operators : comma operator , Size of operator, address operator.

↳ Type conversion: one data type → another type
Two types - Implicit and Explicit
Implicit (Automatic type conversion)
Explicit (Type casting) - converting manually from one data type to another.

↳ Expression evaluation
- Precedence
- Associative rule.

Ex: Let a=1, b= 2, c=3, d=5
Evaluate the following.

x = (a+d) / (b+c) + 15 % 3
= (1+5) / (2+3) + 15 % 3
= 6/5 + 15 % 3
= 1 + 15 % 3 = 1+0

x = 1

y = a-d + 30 * b/ (a+2)
y = 1-5 + 30 * 2/(1+2)
= 1-5 + 30 * 2/3
= 1-5 + 60/3

Date :
Page No :

= 1 - 5 + 20 = 16

~~= 1 - 25~~ = -24

↳ conditional branching system / statements

- if statement
- if-else
- if-else-if
- else-if ladder
- switch.

↳ if statement : It is one of the simplest conditional control statement.

```
if (Expression)
{
    statement 1;
}
statement 2;
```

Ex: Write a program to read the age of a person check whether he/she is eligible to vote.

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int age;
  printf ("Enter age of a person");
  scanf ("%d", &age);
  if (age >= 18)
  {
  printf("Eligible to vote");
  }
getch();
}
```

Flowchart

Date :
Page No :

↳ if - else statement :

```
if (Expression)
{
  statement 1 ; → true-block
}
else
{
  Statement 2 ; → false-block
}
Statement 3;
```

Ex: Write a program to read two numbers and find out greatest among them.

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a,b;
  scanf ("%d%d", &a,&b);
  if (a>b)
  {
   printf ("%d is greatest", a);
  }
  else
  {
   printf ("%d is greatest", b);
  }
 getch();
}
```

Date :
Page No :

Ternary operator

Write a program to read three numbers. Find out greatest among them using ternary operators.

```
#include <stdio.h>
  void main()
  {
    int a,b,c, large;
    printf("Enter three numbers");
    scanf("%d %d %d", &a, &b, &c);
    large = ((a>b) && (a>c)) ? a : (b>c) ? b : c;
    print("large");
  }
```

Write a program to read a number, check whether it is even or odd.

```
#include <stdio.h>
#include <conio.h>
 void main()
 {
    int a, b;
    printf("Enter a number:");
    scanf("%od", &a);
    b= a%2;
    if (b==0)
    {
      printf("The number is even");
    }
    else
    {
      printf("The number is odd");
    }
```

getch();
}

Write a program to enter a character. Check whether it is a vowel or constant.

```
#include <stdio.h>
#include <conio.h>
void main()
{
  char ch;
  printf("Enter a characters : ");
  scanf("%c", &ch);
  if ((ch=='A') || (ch=='a') || (ch=='e') || (ch=='E')
      (ch=='i') || (ch=='I') || (ch=='O') || (ch=='o')
      (ch=='U') || (ch=='u'))
  {
    printf("It is vowel");
  }
  else
  {
    printf("It is consonant");
  }
  getch();
  }
```

↳ Nested if...else statement: When a series of decisions are involved in a nested form.

A if within another if is nested if statement.

```
if (Expression 1)
{
  if (Expression 2)
  {
```

Date :
Page No :

```
statement 1;
}
```

Write a program to read four numbers. Find out greatest among them

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b,c,d;
printf("Enter any 4 numbers : ");
scanf("%d%d%d%d",&a,&b,&c,&d);
if (a>b)
{
  if (a>c)
   { if (a>d)
     {
      printf("A is greatest");
     }
   }
}
else if (b>c);
{
  if (b>d)
  {}
  printf("B is greatest");
  }
}
```

```
else if (c>d)
{
printf ("c is greatest");
}
else
{
printf ("D is greatest");
}
getch();
}
```

↳ Switch statement

It is a simplified version of if-else statement

It is a multi-way decision statement

```
switch (choice)
{
case label 1 : block 1;
               break;
case label 2 : block 2;
               break;
case label 3 : block 3;
               break;
default : default - block
          break;
}
Next statement.
```

Write a program to read day of a week in number display

Write a program to enter a number from 1-7 and display the corresponding day of the week using switch case statement.

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int ch;
   clrscr();
   printf("Enter a number from 1-7 \n");
   scanf("%d", &ch);
   switch (ch)
   {
     case 1:
      printf("Sunday");
      break;
      case 2:
      printf("Monday");
       break;
       case 3:
       printf("Tuesday");
       break;
       case 4:
       printf("Wednesday");
       break;
       case 5:
       printf("Thuesday");
       break;
       case 6:
        printf("Friday");
        break;
```

```
        case 7:
        printf ("saturday");
        break;
        default :
        printf ("Wrong input")
        break;
      }
      getch();
    }
```

Write an program to read an alphabet, check whether it is vowel or consonant using switch statement.

```
#include <stdio.h>
#include <conio.h>
  void main()
{
  char ch;
  clrscr();
  printf ("Enter an alphabet \n");
  scanf ("%c", &ch);
  switch (ch)
  {
   case 'a':
   case 'A':
   case 'e':
   case 'E':
   case 'i':
   case 'I':
   case 'O':
   case 'o':
   case 'U':
   case 'u':
```

```
        printf (" It is a vowel");
        break;
        default:
        printf (" It is a consonant );
        break;
        }
        getch ();
        }
```

Write a program to demonstrate the use of switch statement without break statement.

```
#include <stdio.h>
#include <conio.h>
 void main ()
 {
  int ch;
  clrscr();
  printf ("Enter your choice");
  scanf ("%d" ,&ch);
  switch (ch)
  { case 1:
   printf (" It is case 1 /n");
   case 2:
   printf (" It is case 2 /n");
   case 3:
   printf (" It is case 3 /n");
   default:
   printf (" default choice ");
  }
  getch ();
 }
```

Date :
Page No :

Output:

```
Enter your choice : 1
It is case 1
It is case 2
It is case 3
default choice
```

↳ Dangling else problem

This problem encounters with nesting if statement. This problem is created when there is no matching else for every if statement.

Example:
```
if (a>b)
    if (a>c)
        if (a>b)
        printf ("a");
    else
        printf ("d")
```

↳ Iterative statements

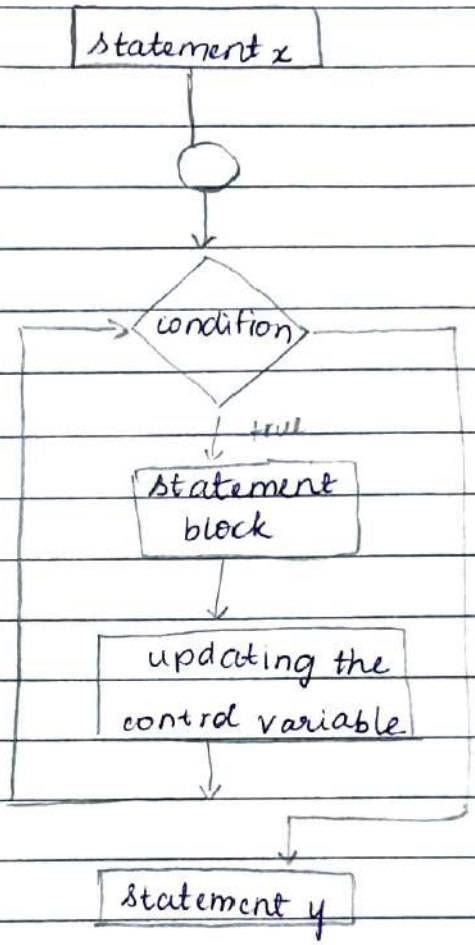
- They are used to repeat the execution of one or list of statements depending on the value of integer expression.
- C language supports three looping statements:
  - while
  - Do-while
  - for

- while loop is a pretest loop.
- It provides a mechanism to repeat one or more statements while a particular condition is true.

Date :
Page No :

Syntax :

```
statement x ;
while (condition)
{
  statement block
}
statement y;
```



Working process :

When a control enters / reach the while statement, it first test the condition only if condition is true it executes statement block, otherwise if condition is false it jumps to statement y

Write a program to print multiplication table.

```
void main()
{
  int i=1;
  while (i<=10)
  {
   printf ("7 x %d = %d \n", i, ix7);
   i++;
  }
  getch();
}
```

Write a program to calculate the sum of 1st 10 numbers

```
#include <stdio.h>
 void main()
 {
  int i=1, sum=0;
  while (i<=10)
  {
    sum=sum+i;
    i++;
  }
  printf ("%d", sum);
 }
 getch();
}
```

Write a program to read 15 numbers. Find the sum of them.

```
void main ()
{
 int i=1, n , sum=0 ;
 while ( i<=15)
 {
  printf ("Enter any variable");
  scanf ("%d", &n) ;
  sum = sum + n;
  i=i+1;
 }
  printf (" sum of 15 nos = %d", sum);

 getch ();
}
```

Write to a program to read a number until -1 is encountered. Also count the no. of negative no., positive no. and 0 entered by the user.

```
void main ()
{
 int PC=0, NC=0, ZC=0, n;
 printf (" Enter a number (or -1 to exit) \n");
 scanf ("%d", &n);
 while (n!=-1)
 {
  if (n>0)
   PC++;
  else if (n<0)
   NC++;
```

```
        else
            zc++;
        printf("Enter a number (or -1 to exit)\n");
        scanf("%d", &n);
    }
    getch(); printf("No. of positive numbers = %d \n", PC);
    printf("No of negative numbers = %d \n", NC);
    printf("No. of zeros = %d \n", zc);
}
    getch();
}
```

Write a program to calculate average of numbers entered by user.

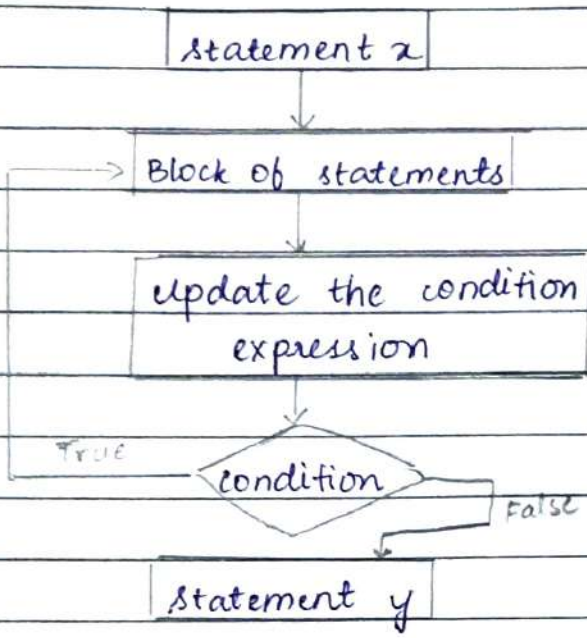
```
void main()
{
    int n, sum = 0, c = 0;
    float avg;
    printf("Enter a number or -1 to exit");
    scanf("%d", &n);
    while(n != -1)
    {
        c++;
        sum = sum + n;
        printf("Enter a number or -1 to exit: ");
    } scanf("%d", &n);
    avg = (float) sum / c;
    printf("sum of %d = %d", c, sum);
    printf("Average of %d = %f", c, avg);

getch();
}
```

Date :
Page No :

↳ DO-while loop → post-test loop

```
statement x;
do
{
  statement block;
} while (condition);
statement y;
```

Flowchart



Write a program to print natural numbers from 1 to n using do-while loop.

```
void main()
{
 int c=1, n;
 printf("Enter a value for n: ");
 scanf("%d", &n);
 do
 {
 printf("%d \t", c);
  c = c+1;
```

Date :
Page No :

```
} while (c<=n);
getch();
}
```

Write a program to calculate average of n numbers.

```
void main()
{
```

Write a program to read n and find the factorial of n.

```
void main()
{
int p=1;n, i
printf ("Enter a value for n: ");
scanf ("%d",&n);
for (i=1; i<=n ;i++)
 {
  fact = fact * i;
 }
printf (" factorial of %d = %d", n, p);
getch();
}
```

Write a program to read a number. Check whether it is prime or not.

```
void main()
{
 int i, n, flag=0;
 printf (" Enter a number for n: ");
 scanf (" %od" ,&n);
 for (i=2 ; i≤n-1 ; i++)
 {
  if ((n%i) == 0)
  {
   flag=1;
   break;
  }
 }
 if(flag=0)
 printf ("It is prime");
 else
```

```
printf (" it is not prime ");
getch ();
}
```

Nested loops

A loop inside a loop.

- WAP to print the following pattern

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

```
void main ()
{
 int i,j ;
 for (j=1 ; j<=3 ; j++)
 {
  for ( i=1; i<=5 ; i++)
  {
  printf ("*");
  }
  printf("\n");
 }
 getch();
}
```

- WAP to print the following

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

```
void main ()
{
 int i, j;
 for (j=5, j>=1 ; j=j-2 )
 {
  for (i=1 ; i<= j , i++)
  {
  printf ("* ");
  }
  printf ("\n");
 }
 getch ();
}
```

Date :
Page No :

WAP to print the following patterns

```
void main()
{ int i, j, k, n = 10
for (i=1, i<=7; i=i+2)
  {
   for (k=1; k<=n, k++)
   printf (" ");
   for (j=1; j<=i; j++)
  {
   printf ("*")
  }
   printf ("\n");
   n--;
  }
  getch();
}
```

Break statement:

↳ terminates the execution of the remaining iteration of the loop.

↳ Syntax : break;

- WAP to read numbers until -1 is encountered

```
int n;
for (;;)
{
printf("Enter a no. : ");
scanf ("%d", &n);
  if (n==-1)
   break;
}
```

```
for(i=5; i>=1; i--){
{
 for (j=i; j>=1; j--){
  } printf("%d", j);
 }
```

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

Date :
Page No :

↳ Continue statement

It ~~cont~~ terminates only the current iteration of loop.

Syntax : continue;

WAP to print natural no.s from 1 to 100 except multiple of 7.

```
int i;
{
for (i=1 ; i<=100 ; i++)
{
  if (i%7 ==0)
    continue;
  printf ("%d", i);
}
```

↳ Goto statement is used to transfer the control to the specified label

goto label;

In go to statement, the label can be anywhere in the program. either before or after go to statement.

There are two types of goto statement:

i) forward jump ii) Backward jump.

A label can be digits, alphabets or combination

WAP to print your name infinite times

```
void main ()
{
10: printf (" Varshini");
go to 10;
}
```

Date :
Page No :

MODULE 3.

FUNCTIONS AND ARRAYS

- Function is a block of code to perform a specific task
- Two types : • User defined
  • Library functions
- Library functions - in-built function
  Ex: main(), printf(), scanf()
- User defined - Defined by the user.

- Function structure (definition)

```
return_type function_name (formal parameters)
{
    Variable declaration;
    Statement 1;
    Statement 2;
    return value;
}
```

- Function declaration
  Every function in C should be declared before they are used.

```
return_type function_name (formal paramaters);
```

- Function call

```
function_name (arg1, arg 2);
```

- return type
  used inside a function. when a function return back to the calling function it carries the value along with return statement.

Date :
Page No :

1. Write a function by the name 'add' to perform sum of two numbers.

```
# include <stdio.h>
int add (int num1, int num2);          → declaration
void main()
{
  int a, b, s;
  printf ("Enter 2 numbers: ");
  scanf ("%d %d", &a, &b);
  s = add (a,b);                       → call
  printf ("sum = %d", s);
}
int add (int num1, int num2)           → definition
  {
    int sum;
    sum = num1 + num2;
    return sum;                        → return
  }
```

Working operation of a function

- Every C program begins from main() and program starts executing the codes inside main() function.
- When the control of program reaches to function_name inside main() function, the control of program jumps to void function_name and executes the codes inside it.

Date :
Page No :

2. Write a program to create a function that performs factorial of a number.

```
#include <stdio.h>
# include <conio.h>
 int factorial (int);          → declaration
void main ()
{
 int n, fact;
 printf ("Enter a number");
 scanf ("%d", &n);
 fact = factorial (n);
 printf (" factorial of %d = %d", n, fact);
 getch ();
}
 int factorial (int num)
{
 int i, prod = 1;
 for (i=1; i <= num ; i++)
 {
  prod = prod * i
 }
 return prod;
}
```

3. WAP to check whether a no is even / odd.

```
 #include <stdio.h>
void Evenodd ()
 {
  int num;
  printf (" Enter a number : ");
  scanf ("%d", &num);
```

```
if (num % 2 == 0)
    printf ("The number is even");
else
    printf ("The number is odd");
void main()
{
 Evenodd ();
 getch();
}
```

↳ Function declaration / prototype

```
main()
{
 ---
 ---
 func1 ();
 statement y;
 ---
 ---
}
```

```
func 1
{
 ---
 ---
}
```

4. WAP to create functions for finding area of a □^le, area of □^re, area of O.

```
#include <stdio.h>
# define pi 3.145
void areacircle()
{
 float r, area;
 printf ("Enter radius : ");
 scanf ("%f", &r);
 area = pi * r * r ;
 printf ("Area of a circle = %f", area);
}
```

```
void areacirc()
{
 int r, b, area;
 printf("Enter length & breadth:");
 scanf("%d%d", &l, &b);
 area = l * b;
 printf("Area of rectangle = %d", area);
}
void areasquare()
{
 int a, area;
 printf("Enter side of a square:");
 scanf("%d", &a);
 area = a * a;
 printf("Area of a square = %d", area);
}
void main()
{
 clrscr();
 areacircle();
 areacirc();
 areasquare();
 getch();
}
```

5. WAP to read a number and check whether it is prime or not using a function

```
#include <stdio.h>
#include <conio.h>
void int isprime(int num)
{
   int i;
```

```
for (i=2 ; i<num; i++)
{
  if (num % i == 0)
    {
      return 0;
    }
}
return 1;
}
void main()
{
int flag;
int n; → clrscr();
printf("Enter a numbers : ");
scanf ("%d" &n);
flag = isprime (n);
if (flag==1)
{
printf ("%t is prime");
}
else
{
 printf ("%t is not prime");
 }
 getch();
}
```

Different types of user-defined function

1. Function with argument with return value
2. " without " with " "
3. " with " without " "
4. " without " without " "

Date :
Page No :

6. same as 1. (without arguement without return)

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

7. same as 1. (without arguement with return)

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

```
    s = sum ();
    printf ("sum = %d", s);
}
```

8. Same as 1

```
#include <stdio.h>
#include <conio.h>
void sum (int num1, int num2)
{
    int s;
    s = num1 + num2;
    printf ("sum = %d", s);
}
void main ()
{
    int s;
```

9. WAP to perform swapping of two numbers with third variable using function.

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int a, b, temp;
    swap ();
    getch ();
}
```

```
void swap();
 {
  int a, b, temp;
  printf("Enter any two numbers : ");
  scanf("%d%d", &a, &b);
  a = temp;
  a = b; b = temp;
  printf("The values of a and b = %d%d", a, b);
 }
```

(or)

```
void main()
{
 int x, y;
  x = 10;
  y = 20;
 printf("Before swapping x = %d and y = %d\n"x,y);
 swap(x, y);
 getch();
}
 void swap(int a, int b);
 {
  int temp;
  temp = a;
  a = b;
  b = temp;
  printf("After swapping x = %d and y = %d\n",
          a, b);
 }
```

Date :
Page No :

10. Same as 9. (without using third variable)

```
void swap (int a, int b);
{
  a = a+b;
  b = a-b;
  a = a-b;
```

↳ Passing parameters to the function
- Call by value (passing value)
- Call by reference (passing address)

↳ Example for call by value (10.)

x	10	void main()
y	20	{
a	20	swap (x, y);
b	10	printf ("After swapping : x = %d and y = %d", x, y);

↳ Call by Reference

In call by reference the calling function sends the address of the actual parameters to the called function. (formal parameters). Here, the changes in F.P affects the actual parameters.

```
#include <stdio.h>
void swap (int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
```

```
printf (" After swapping first no. = %d and second
    no = %d \n ", *a, *b);
}
void main()
{
  int x=10, y=20; clrscr
  getch();
  printf (" Before swapping x = %d and y = %d", x, y);
  swap (&x, &y);
  printf ("After swapping x = %d and y = %d", x, y);
  getch();
}
```

In the above example. the changes affects actual paramaters also.

↳ Difference blw call by value & call by reference

11. WAP to read a number (floating point) and convert into binary. using function

```
void main()
{
  int num, rem, bin=0, i=0;
  printf ("Enter a number :");
  scanf ("%d", &num);
  temp = num;
  while (num != 0)
  {
    rem = num % 2;
    bin = rem * pow(10, i++) + bin;
    num = num / 2;
  }
  printf ("Decimal number = %d", temp);
```

```
printf(" Binary number = %d ", bin);
getch();
}
```

↳ Recursion function is defined as a function that calls itself to solve a smaller function of its task until a final is made which does not require a a call to itself

↳ Recursion refers to the process where a function call itself either directly or indirectly.

Two types:

- Direct
- Indirect

↳ Direct recursion: In this process where a function calls itself directly

```
func 1 ()
{
  - - -
  - - - -
  func1();
  - - -
  - - - -
}
```

↳ Indirect recursion: In this process where a function calls another function and that function calls back the original function.

```
func 1 ()
{
  - - -
  - - -
  func 2 ();
  - - -
  - - -
}

func 2 ()
{
  - - -
  - - -
  func3();
  - - -
  - - -
}

func 3 ();
{
  - - -
  - - -
  func 1 ();
  - - -
  - - -
}
```

Date :
Page No :

WAP to find factorial of a number using recursion

```
#include <stdio.h>
int factorial (int n)
{
   if (n==1)
     return 1;
   else
     return (n*factorial (n-1));
}
void main()
{
  int n, res;
  printf ("Enter a number : ");
  scanf ("%d", &n);
  res = factorial (n);
  printf ("The factorial of %d = %d", n, res);
  getch();
}
```

Traceout:

```
5 * factorial (4)
4 * factorial (3)
3 * factorial (2)
2 * factorial (1)
1 *
```

O/P

```
Enter a number : 5
The factorial of 5 = 120
```

WAP to find GCD of two numbers using recursion

Divisor dividend
2) 8 (4
-8
0 → rem
quotient

```
#include <stdio.h>
int GCD (int, int);  → Declaration
int main()
{
int num1, num2, res;
printf ("\n Enter the two numbers:");
```

```
scanf("%d %d", &num1, &num2);
res = GCD(num1, num2);  ⟶ calling func
printf("\n GCD of %d and %d = %d", num1, num2, res);
getch();
return 0;
}
int GCD (int x, int y)  ⟶ called func.
{
  int rem;
  rem = x % y;
  if (rem == 0)
    return y;
  else
    return (GCD (y, rem));
}
```

WAP to print fibonacci series upto n terms

```
#include <stdio.h>
int Fibonacci (int);
int main()
{
  int n, i = 0, res;
  printf("Enter the number of terms \n");
  scanf("%d", &n);
  printf("Fibonacci series \n");
  for (i = 0; i < n; i++)
  {
    res = Fibonacci (i);
    printf("%d \t", res);
  }
  return 0;
}
int Fibonacci (int n)
```

```
{
    if (n==0)
        return 0;
    else if (n==1)
        return 1;
    else
        return (Fibonacci(n-1) + Fibonacci(n-2));
}
```

- Find the exponent of a given number with the help of recursion.

↳ User-defined header files

```
int factorial (int n)
{
    int prod=1, i;
    for (i=1; i<=n ; i++)
    {
        prod = prod * i;
    }
    return prod;
}
```

⇒ under the header file name fact.h

WAP to find out the value of ${}^nC_r = \frac{n!}{r!(n-r)!}$

```
#include <stdio.h>
#include "fact.h"
void main()
{
    int n, r, res;
    printf("Enter the value for n and r : ");
```

```
scanf("%d%d", &n, &r);
res = factorial(n) / factorial(r) * factorial(n-r);
printf("NCR = %d", res);
getch();
}
```

ARRAYS

Array: Array is a fixed size sequenced collection of elements of same type. It is a desived data type

(or)

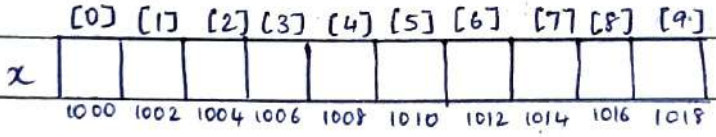
Array is collection of homogeneous elements of same data type sharing the common name.

Example: List of students in class

Declaration of an array

data type array_name [size];

Ex: int x [10];



- A data type can be any basic data. Ex: int, float, char, double
- Name of an array is used for identifying the array.
- size - max. no. of values that an array can hold

↳ Types of array

- Single-dimensional array: It is linear list of related data items of same data type where we can access the elements of an array using a single subscript

↳ Initialization of one-dimensional array.

During variable declaration, assigning a value is called initialization.

Syntax:

data-type array_name [size] = {list of values};

Example:

int marks [5] = {90, 93, 89, 88, 91};

↳ Two types of initialization

- Compile-time
  - ↳ Initialization without size
  - ↳ with size
  - ↳ with zero
  - ↳ Partial ~~itt~~ initialization
- Run-time

Ex:
```
int marks [4], i;
printf ("Enter the marks \n");
for (i=0; i<4; i++)
{
  scanf ("%d", &marks [i]);
}
```

↳ Inputting values from the keyboard

We can access the value stored in a given element of an array by specifying array name and index number inside the square brackets.

Array_name [index];

Date :
Page No :

WAP to create an array of 10 integers and print them.

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a[10], i;
  printf("Enter 10 values");
  for (i=0 ; i<10 ; i++)
    scanf ("%d", &a[i]);
  for (i=0; i<10 ; i++)
    printf ("%d\t", a[i]);
}
```

↳ Operation on arrays

- Traversing
- Inserting
- Deleting
- Merging
- searching
- sorting

° Traversing

WAP to read n students total marks. Find out average total marks.

```
#include <stdio.h>
void main()
{
  int marks[100], i, n, sum=0;
  float avg;
  printf(" Enter number of students : ");
  scanf ("%d", &n);
  printf ("Enter %d students marks", n);
  for (i= 0 ; i<n ; i++)
```

```
scanf("%d", &marks[i]);
for (i=0; i<n; i++)
  sum = sum + marks[i];        ⇒ Traversing
avg = (float) sum/n;
printf("Average of %d students marks = %f\n", n,
                                        avg);
}
```

WAP to ~~create~~ read n numbers and find out the highest.

```
#include <stdio.h>
void main()
{
  int marks[100], n, i, large;
  printf("Enter the value for n:");
  scanf("%d", &n);
  printf("Enter %d student marks", n);
  for (i=0, i<=n-1 ; i++)
    scanf("%d", &marks[i]);
  large = mark[0];
  for (i=1; i<=n-1; i++)
  {
    if (mark[i] > large)
      large = mark[i];
  }
  printf("Largest of %d numbers = %d", n,
                                    large);
}
```

Date :
Page No :

WAP to find out largest and second largest.

```
# include <stdio.h>
# include <conio.h>
void main()
{
int marks [100], n, i, large, seclarge ;
printf ("Enter no. of students : ");
scanf ("%d", &n);
for (i=0; i<n; i++)
   scanf ("%d", &marks [i]);
large = marks [0];
seclarge = marks [0];
for (i=1; i<n; i++)
{
 if (marks [i] > large)
 { seclarge = large [i];
   large = marks [i];
 }
 else
 { if (marks [i] > seclarge)
     seclar = mark [i]
 }
printf ("largest = %d", large);
printf ("second largest = %d, seclarge);
getch();
}
```

↳ Array searching :

Finding the key value in a given array. starting from 0th element till the last.

Two types of searching :

Date :
Page No :

i) Linear searching

It is one of the simplest searching method where we check from the beginning till the end with all the elements of an array.

WAP to read n elements / numbers. Find out whether a key value is existing or not using linear search.

```
#include <stdio.h>
void main()
{
int a[100], i, n, key, found=0;
printf ("Enter a value for n:");
scanf ("%d", &n);
for (i=0; i<n; i++)
  scanf ("%d", &a[i]);
printf (" Enter a key value:");
scanf (" %d", &key);
for (i=0; i<n; i++)
 {
  if(a[i]==key)
  {
   found=1;
   break;
  }
 }
  if (found==1)
printf ("Number found at %d position", i);
  else
printf ("Number is not found");
getch();
}
```

ii) Binary search : In this method, we search for a key element in a sorted array.

Example for linear search:

int A[] = { 10, 8, 2, 7, 3, 4, 9, 1, 6, 5 };

Value to be searched

Value is found and it is at position 3.

Advantages of binary search over linear search:

- No. of search will be very less in binary search.
- The time taken for searching will be less.
- Specially used with large data.

Drawbacks:

- It should always be in sorted manner

↳ Lab program:

↳ Sorting : Arranging the elements of a given array in ascending / descending order.

Bubble sort : This is one of the simplest and easiest sorting technique.

WAP to read n numbers of an array and arrange

```
# Include <stdio.h>
  void main()
  {
  int a[20], n, temp, i, j;
  printf("Enter the no. of elements \n");
  scanf("%d", &n);
  printf("Enter the unsorted array elements");
for (i=0; i<n; i++)
  scanf("%d", &a[i]);
```

```
for (i=1; i<n; i++)
 {
   for (j=0 ; j<n-1; j++)
   {
     if (a[j] > a[j+1])
     {
        temp= a[j];
        a[j] = a[j+1];
        a[j+1] = temp;
     }
   }
}
printf ("Sorted elements are \n");
for (i=0; i<n; i++)
printf ("%d \t", a[i]);
}
```

↳ 2-dimensional Arrays

In A 2-D array, elements are arranged in grid manner i.e in rows and columns.

To access 2-D array element we use two indices (say i and j) where i index indicates row no. and j indicates column no.

↳ Declaration of 2-D array

data-type array-name [row-size][column-size];

Example: int marks [3][5];

Rows/columns	colu. 0	col. 1	col. 2	col. 3	col. 4
Row 0	marks[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
Row 1	marks[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
Row 2	marks[2][0]	[2][1]	[2][2]	[2][3]	[2][4]

Date :
Page No :

↳ 2 ways of storing 2-D array in memory :
• Row Major order • column major order

↳ Initialization of 2-D array :

data-type array-name [row-size] [column_size] =
{ list of values } ;

Example : int marks [2][3] = { 80,90,95,88,97,81 };

or int marks [2][3] = { { 80,90,95 }, { 88,97,81 } }

• Partial initialisation :

int marks [2][3] = { 10,30,35 }

WAP to read and print 2-D array.

```
#include <stdio.h>
    void main()
    {
     int a[50][50], m, n, i, j;
     printf(" Enter no. of rows: ");
     scanf("%d", &m);
     printf(" Enter no. of columns : ");
     scanf("%d", &n);
     printf(" Enter the elements of matrix : ");
     for (j=0; i<m; i++)
       {
        for (j=0; j<n; j++)
         {
          scanf("%d", &a[i][j]);
         }
       }
     printf(" Matrix A : \n");
     for(i=0; i<m; i++)
       {
        for (j=0; j<n; j++)
```

```
    {
      printf ("%d\t", a[i][j]);
    }
      printf ("\n");
    }
   getch();
}
```

WAP to add two arrays

```
#include <stdio.h>
void main()
{
  int m,n,i,j, a[3][3], b[3][3], c[3][3];
  printf ("Enter ~~vari~~ no. of rows and columns\n");
  scanf ("%d%d", &m, &n);
  printf (" Enter Matrix A elements\n");
  for (i=0; i<m; i++)
    {
     for (j=0; j<n; j++)
       {
         scanf ("%d", &a[i][j]);
       }
     }
  printf (" Enter Matrix B elements\n");
   for (j=0; i<m; i++)
     {
      for (j=0; j<n; j++)
        {
         scanf ("%d", &b[i][j]);
        }
      }
```

```
for (i=0; i<m; i++)
 {
 for (j=0; j<n; j++)
  {
    c[i][j] = a[i][j] + b[i][j];
  }
 }
 printf ("Resultant matrix : \n");
  for (i=0; i<m; i++)
   {
    for (j=0; j<n ; j++)
     {
      printf ("%d\t", a[i][j]);
     }
     printf ("\n");
   }
   getch ();
  }
```

WAP to find the transpose of a matrix

```
# include <stdio.h>
# include <conio.h>
 void main ()
 {
  int a[10][10] , transpose [10][10] , i, j, m, n;
  printf (" Enter rows and columns of A matrix ");
   scanf (" %d %d ", &m, &n);
  printf ("Enter elements of matrix A : ");
  for (i=0; i<m; i++)
   {
    for (j=0; j<n; j++);
     scanf ("%d", &a[i][j]);
```

```
// Transpose of matrix
for (i=0; i<m; i++)
 {
  for(j=0; j<n ; j++)
  {
   Transpose [j][i] = a[i][j]
  }
printf (" Resultant transpose matrix \n");
 for (i=0; i<n; i++)
 {
  for (j=0; j<m ; j++)
   {
    printf("%d\t", transpose[i][j]);
   }
   printf (" \n");
 }
 getch();
 }
```

Output :

```
Enter rows and columns of A matrix  2 3
Enter elements of matrix A : 1,2,3,4,5,6
Resultant transpose matrix
   1   4
   2   5
   3   6
```

WAP to find the diagonal elements of 3x3 matrix

```
void main()
{
 int a[3][3], i, j;
 printf (" Enter the element of A matrix \n");
```

```
for (i=0; i<3; i++)
  {
   for (j=0; j<3; j++)
   scanf ("%d", &a[i][j]);
  }
printf ("Diagonal elements\n");
 for (i=0; i<3; i++)
   {
   printf ("%d\t", a[i][i]);
   }
  getch ();
  }
```

Matrix multiplication

A

m x n ~ 2 x 3

1 2 1

2 3 4

B

p x q ~ 3 x 2

1 0

2 3

4 5

C

m x q

0,0 x 0,0 + 0,0 x 1,0 +

0,1 x 1,0 + 0,1 x 1,1 +

0,2 + 2,0 0,2 x 2,1

1,0 x 0,0 + 1,0 x 0,1 +

1,1 x 1,0 + 1,1 x 1,1 +

1,2 x 2,0 1,2 x 2,1

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int a[10][10], b[10][10], c[10][10] = {0};
 int m, n, p, q, i, j, k;
printf("Enter order of A matrix");
scanf("%d%d", &m, &n);
printf("Enter order of B matrix");
scanf("%d%d", &p, &q);
 if(n != p)
  {
  printf("Matrix multiplication not possible");
  getch();
  exit(0);
  }
printf("Enter elements of A matrix:");
  for(i=0; i<m; i++)
   for(j=0; j<n; j++)
    scanf("%d", &a[i][j]);
printf("Enter elements of B matrix:");
   for(i=0; i<p; i++)
     for(j=0; j<q; j++)
       scanf("%d", &b[i][j]);
// Matrix multiplication
  for(i=0; i<m; i++)
   {
    for(j=0; j<q; j++)
    {
     for(k=0; k<n; k++)
      {
       c[i][j] = c[i][j] + a[i][k] * b[k][j];
   } }
}
```

```
printf (" c Matrix \n");
for (i=0; i<m; i++)
  {
   for j=0; j<q; j++)
    {
     printf ("%d\t", c[i][j]);
    }
    printf ("\n");
  }
  getch();
}
```

Date :
Page No :

MODULE 4.

STRINGS AND POINTERS

↳ string is a null-terminated character array.

↳ Declaration of string :

char name of [size];
string

Ex: char usn [15];

↳ Initialization of string:

char str [10] = "HELLO";

char str [10] = {'H', 'E', 'L', 'L', 'O', '\0'}

H	E	L	L	O	\0				

↳ Reading string

There are 3 types of function used to read strings

- scanf()
- gets()
- getch(), getche(), getchar()

↳ scanf()

scanf ("%s", name of string);

Ex: char name [25], USN [11];
scanf ("%s", name);
scanf ("%s", USN);

Drawback in scanf

The main drawback in scanf is that it terminates the function as soon as it encounters a blank space.

Date :
Page No :

↳ gets()

It is a simple function which is used to read a string. This function overcomes drawbacks of scanf().

gets(name of string);

EX: gets (name);

gets (USN);

↳ getch()

It helps you to read only one character. Therefore to read string we must repeat getch() until specified character encounters.

```
EX: void main()
    {
      char name[25]; i=0;
      char ch;
      ch = getch();
      while (ch != '*')
      {
       name[i] = ch;
       ch = getch();
       i++;
      }
```

→ TO store a charac.

Write a string

- printf

```
int name[15] = "GCEM Engg"
printf("%6.4s", name);
```

		G	C	E	M

#include <string.h>

Date :
Page No :

- puts

  puts(name of string);

- putchar

  putchar is a function which is used to print only at a time. so in order to print a string using putchar, we need to repeat again and again until you reach to the end of the string

WAP to assign and print a string using putchar func.

```
void main ()
{
  char name [10] = "Hello", ch;
  int i = 0;
  ch = name [i];
  while (ch != \0)
  {
    putchar (ch);
    i++;
    ch = name [i];
  }
  getch ();
}
```

↳ Operation on strings

- Length of string
- string copy
- string compare
- String concatenation
- Create a substring
- Replace with some other characters
- Convert etc.

Date :
Page No :

↳ string built-in function

• length

strlen ();

This function is used to find length of the given string.

Syntax: strlen (string);

Example: char name [20] = " zenith Mathew ";

```
int l;
l = strlen (name);
printf ("Length of string = %d", l);
```

Length of string = 13

• Copy - strcpy ()

This string function is used to copy one string to another string.

Syntax: strcpy (source string , destination string);

EX:

```
char name1 [10] = " XYZ";
char name2 [10];
strcpy (name2, name1);
puts (name1);
puts (name2);
```

• Strncpy ()

This function copies upto n characters from the source string to destination string.

Syntax: strncpy (source string , destination string , no. of characters copied from soura

EX:

```
char name [15] = "COMPUTER";
char name2 [15];
strncpy (name, name2, 4);
puts (name2);
```

OUTPUT: COMP

strncpy (str1, str2, n);
d s

In this func. upto n characters of str2 is copied to str1.

Date :
Page No :

- strcat()

This function appends the string pointed to by str2 to the end of the string pointed by str1 and terminating null character of str1.

Syntax: str (str1, str2);

```
{
    char str1 [15] = "COMP";
    char str [5] = "UTER";
    strcat (str1, str2);
    puts (str1);
}
```

O/P: COMPUTER

- strncat()

This function appends string pointed by str2 to the end of the string pointed by str1 upto n characters.

Syntax: strncat (str1, str2, n);

```
{
    char str1 [15] = "COMP";
    char str [15] = "UTER";
    strncat (str1, str2, 3);
    puts (str1);
}
```

O/P: COMPUTE

- strcmp()

same same = 0
str1 > str2 = >0
str1 < str2 = <0

This function compares the string pointed by str1 to the string pointed by str2. The function returns 0 if the strings are equal otherwise it returns less than 0 or greater than 0

Date :
Page No :

WAP to read 2 strings, check whether both are equal or not.

```
void main()
{
char str1[15], str[15];
printf("Enter two strings : ");
gets (str1);
gets (str2);
if (strcmp (str1, str2) == 0)
  printf("Both the strings are equal");
else
if (strcmp (str1, str2) > 0).
  printf ("The first string is bigger");

else
printf ("The second string is bigger");
}
```

- strncmp()

This function compares the first n characters of str1 with str2.

```
strncmp (str1, str2, n);
{
  str1 = "COMPUTER"
  str2 = "COMPARE"
x = strncmp (str1, str2, 4);
if (x == 0)
  printf ("upto 4 charaters both strings are equal");
else
  printf ("upto 4 characters both strings are
           not equal.");
```

Date :
Page No :

- strstr()

This function is used to find the first occurence of str2 in the given string str1. It returns a pointer to the first occurence of str2 in str1. If no match is found, then a null pointer is returned.

```
strstr (str1, str2);
  {
    char str1[15], str[15];
    int *ptr;
    str1 = "Happy days in college";
    str2 = "days";
    ptr = strstr (str1, str2);
    if (ptr)
        printf ("str2 is present in str1");
    else
        print ("str2 is not present in str2");
  }
```

List of built-in functions

- ↳ strlen()
- ↳ strcpy()
- ↳ strncpy()
- ↳ strcmp()
- ↳ strncmp()
- ↳ strstr()
- ↳ strchr()
- ↳ strnchr()
- ↳ strspn()
- ↳ strnspn()

↳ Operations (without built-in functions)

WAP to read a string and find out length of the string without built-in function.

```
#include <stdio.h>
void main()
{
char str[100];
int i = 0;
clrscr();
printf("Enter a string:");
gets(str);
puts(str);
while (str[i] != '\0')
i++;
printf("Length of string = %d", i);
}
```

WAP to read a sentence. Find out number of words in a given sentence.

```
#include <stdio.h>
void main()
{
char str[100];
int i=0; count=1;
clrscr();
printf("Enter a sentence:");
gets(str);
puts(str);
while (str[i] != '\0')
{
if (str[i] == ' ')
count++;
i++;
}
```

```
    printf ("No. of words in a given sentence =
                %d", count)
}
```

WAP to read a string. check whether it is palindrome or not.

```
#include <stdio.h>
void main ()
{
char str [100];
int i, j, length;
clrscr();
printf ("Enter a word:");
gets (str);
i = 0;
while (str[i] != '\0')
  i++;
length = i;
 i= 0;
 j= length-1;
while (i<= length/2)
{
 if (str[i] == str[j])
  {
     i++;
     j--;
  }
else
  break;
}
if (i> = j)
 printf ("It is a palindrome.");
```

```
    else
    printf (" It is not a palindrome. ");
}
```

(or)

```
#include <stdio.h>
int main()
{
    char string [100];
    int length=0, flag=1, i;
    printf ("Enter a string: \n");
    gets (string);
    for (i=0, string [i] != '\0', i++)
    {
        length++;
    }
    for (i=0; i < length/2 ; i++)
    {
        if (string [i] != string [length-1-i])
        {
            flag = 0;
            break;
        }
    }
    if (flag == 1)
    {
    printf ("It is a palindrome");
    else
    printf ("It is not a palindrome.");
    return 0;
}
```

WAP to read 2 strings and perform string concatenation and place it into 3rd string & display it without built-in function

```
#include <stdio.h>
void main()
{
  char str1[100], str2[100], str3[100];
  int i=0, j=0;
  printf("Enter first string");
  gets(str1);
  printf("Enter second string");
  gets(str2);
  while(str1[i] != '\0')
  {
    str3[j] = str1[i];
    i++;
    j++;
  }
  i=0;
  while(str2[i] != '\0')
  {
    str3[j] = str2[i];
    i++;
    j++;
  }
  str3[j] = '\0';
  printf("3rd string after concatenation");
  puts(str3);
  getch();
}
```

WAP to read 2 strings and compare both the string without built-in function.

```
#include <stdio.h> #include <string.h>
void main()
{
char str1[100], str2[100], value;
int i=0, k;
printf("Enter first string");
gets(str1);
printf("Enter second string");
len1=strlen(str1);
gets(str2);
len2=strlen(str2); for(k=0; str1[i]==str2[k] && str1[i]!=
'\0' && str2[k]!='\0'; i++, k++)

{
if((i==len1) && (k==len2))
{
value=0;
}
}
if (value==0)
printf("Both the strings are equal");
else
if (str1[i] > str2[k])
printf("First string is bigger");
else
printf("second string is bigger");
```

(or)

```
for();
if ((i==len1) && (k==len2))
printf("Both the strings are equal");
```

WAP to read a string and reverse the string without built-in function.

```
#include <stdio.h>
void main()
{
char str[100];
int i, j, len;
printf("Enter a string: ");
gets(str);
for(i=0; str[i]='\0'; i++)
len = i;
j = len-1;
i=0;
while (i<j)
{
temp = str[i];
str[i] = str[j];
str[j] = temp;
i++; j--;
} printf("Reverse string");
puts(str);
}
```

WAP to read a sentence convert the characters of given string into lowercase.

```
#include <stdio.h>
void main()
{
char str1[100], str2[100];
int i,j;
printf("Enter a string: ");
gets(str1);
```

```
for (i=0, j=0; str1[i] != '\0'; i++, j++)
{
  if (str1[i] >= 'A' && str1[i] <= 'Z')
      str2[j] = str1[i] + 32;
  else
      str2[j] = str1[i];
}
str2[j] = '\0';
puts(str2);
}
```

WAP to lowercase → uppercase
and uppercase → lowercase

```
else
  if (str1[i] >= 'a' && str1[i] <= 'z'
      str2[j] = str1[i] - 32;
else
  str2[j] = str1[i];
```

Character manipulation function

All the built-in function are included under ctype.h header file.

- isalnum
- isalpha
- isdigit

WAP to read a character and check whether it is alphabet or not.

```
#include <stdio.h>
#include <ctype.h>
void main()
{
 char ch;
 printf("Enter a character");
ch=getchar()
 if(isalpha(ch))
  printf("%c is an alphabet");
 else
  printf("%c is not an alphabet");
```

- isdigit(character);

```
if(isdigit(ch))
 printf("%c is a digit");
else
 printf("%c is not a digit");
```

WAP to read a character. Check whether it is alphabet or digit or special character.

```
ch= getchar()
if(isdigit(ch))
printf("%c is a digit.");
else if(isalpha(ch))
 printf("%c is an alphabet");
else
 printf("%c is special character");
```

Date :
Page No :

- Array on strings

A set of strings stored in continuous memory location.

char name of [size1] [size2];
string array

size 1 - No. of strings
size 2 - No. of characters in each string

EX: char CSEDNAME [30] [15];

WAP to read n students name and print them

```
#include <stdio.h>
void main()
{
char name [30] [15];
int n, i;
printf ("Enter no. of names: ");
scanf ("%d", &n);
printf ("Enter %d names", n);
for (i=0; i<n; i++)
    gets (name[i]);
// printing all the names
for (i=0; i<n; i++)
{
  puts(name[i]);
  printf ("\n");
}
}
```

MODULE 4

UNIT 2. POINTERS

- Pointer is a variable that holds the address of another variable (memory location)

- The pointer variable contains only the address of the reference variable not the value stored in that address.

Advantages

- Pointers are more efficient in handling arrays
- They are used with function to return multiple values.
- Pointers allow C to support dynamic memory allocation.
- They provide an efficient tool for manipulating dynamic data structures. such as stack, queue, tree, link list.
- Pointers reduce length and complexity of program.
- Pointers reduce program execution time and saves data storage in memory.

Disadvantage:

- A program may crash if sufficient memory is not available at run time to store pointers.
- Indirectly the data of another variable can be changed by pointers without the knowledge of the user.

↳ Declaration of pointer:

data type * name ;

A data type specifies type of the pointer variable that we want to declare like int, float e.t.c
* indicates or tells the compiler that we are creating a pointer variable.
name specifies the name of a pointer variable.

Ex: int * a;
float * m;
char * p;

↳ Initialization of pointer

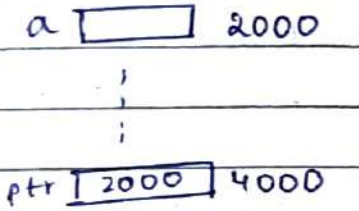
- We can initialise a pointer variable by assigning another variable (address) to it.

  data type * pointer name = & var ;

- data type can be any data type.
- var is an another variable of the same data type

Ex: int a;
int * ptr = &a;

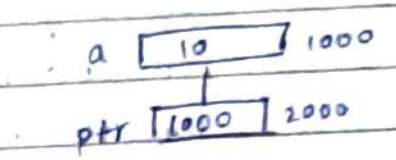
Ex 2: int a;
int * ptr;
ptr = &a;



WAP that illustrate declaration & initialisation of a pointer.

```
#include <stdio.h>
void main()
{
int *ptr;
int a=10;
```

```
    ptr = &a;
    printf("The value of a = %d\n", a);
printf("The value of a using pointer = %d\n", *ptr);
printf("The address of a = %d\n", ptr);
}
```



OUTPUT:

The value of a = 10
The value of a using pointer = 10
The address of a = 1000

↳ Pointer arithmetics

There are two operators in pointers i.e address operator (&) and indirection operator / dereference operator (*)
In direction operator gives the values stored at a particular address.
Address operator cannot be used in any constant or any expression.

Null pointer:

When we assign a constant 0 to a pointer of any type for that symbolic constant null is used.
A null pointer means that it doesn't point to any valid memory location.

Ex for pointer arithmetic:

```
{
    int a;
    int *ptr;
    ptr = &a;
```

```
printf ("Value before ptr = %d", ptr);
ptr = ptr + 1;
printf ("Value after ptr = %d", ptr);
}
```

If integer pointer ptr contains the address 1000 on incrementing +1, we get the address of ptr as 1002 instead of 1001

ptr = ptr + 1

(size of pointer x 1)

ptr = 1000 + (2 x 1)

ptr = 1002

Subtraction:

```
float i = 22.0;
float * ptr;
ptr = &i;
ptr = ptr - 3;
ptr = ptr - 2;
printf (" ptr = %d", ptr);
```

In the above example, consider i is stored in the address 2000.

Then ptr = &i (ptr value will be 2000)

ptr = ptr + 3

= 2000 - (4 x 3)

= 2000 - 12

= 1988

ptr = ptr - 2

= 1988 - (4 x 2)

= 1988 - 8

= 1980

Date :
Page No :

Generic pointer : It is a pointer variable that has void as its data type.
A generic pointer is often used when you want a pointer to point to the data of different data types at different times.

Declaration of generic pointer.

```
void *pointer name;
```

Note : In C we do not know the variable of a type void, the void pointer need to be typecast to another type of pointer.

Example:

```
int a;
char x = 'a';
void *ptr;
a = 20;
*ptr = &a;
printf ("%d", *(int*)ptr);
ptr = &x;
printf ("%c", *(char*)ptr);
```

↳ Passing arguement to function using pointer

In call by reference, we can pass the addresses of the actual parameter to a called function, where the formal parameters will be pointer variable.

When we pass addresses to pointer variable in a given function, it provides the mechasim to modify the data declared in one function code written in another function.

EX: Swapping of two no.s by call by reference

↳ Pointers & Arrays;

The concept of array is very much bond to that of pointers since array occupies consecutive memory locations. with the help of pointers we can easily traversal each element of an array using pointers.

- When we create an array, a base address ie the address of element in an given array will be stored in the array name.
- We can use a pointer variable to point the first element of an array and later the address of the successive element can be easily calculated using pointers. i.e ptr++

Write a program to create an array and print all the elements of the array using pointer.

```
void main()
{
int a[20];
int n, i;
int *ptr;
printf("Enter value for n.");
scanf("%d", &n);
printf("Enter %d elements", n);
for(i=0; i<n; i++)
{
scanf("%d", &a[i]);
}
ptr = a;   // ptr = &a[0];
```

```
printf ("All the elements of an array");
for (i=0 ; i<n; i++)
 {
  printf ("%d\n", *ptr);
  ptr++;
 }
}
```

WAP to add 2 integers by passing pointer variables

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

Pointer to pointer : When a pointer points to (holds the address of) another pointer variable is called pointer to point

Syntax:

```
data type **p;
```

Date
Page No

MODULE 5. STRUCTURE

Structure is a collection of dissimilar or heterogeneous data type grouped together under a common name.

↳ Structure declaration

```
struct tagname                struct
{                             {
  datatype member1;
  datatype member2;
  .                           } s1, s2;
  .
  .
  member n;
};
```

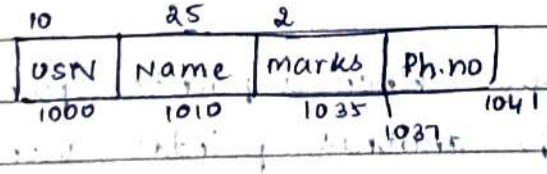
↳ Structure variable declaration

struct tagname var1, var2 ... varn;

EX: struct student s1, s2;

↳ Initialization of structure

struct student s1 = {"1GD22CS071", "zenith", 450, 720407};



↳ Accessing structure elements:

Dot operator is used to access structure member or elements.

name of structure.membername;

EX: sl.usn;
sl.name;
sl.marks;

WAP to create a structure by name of the student, age, marks, roll no, Read all membess of structure and display it.

```
#include <stdio.h>
void main()
{
  struct student
  {
   char name[25];
   int rollno;
   int age;
   int marks;
  };
  struct student sl;
  printf("Enter name, rollno, age, marks: \n");
  gets(sl.name);
  scanf("%d", &sl,rollno);
  scanf("%d", &sl.age);
  scanf("%d", &sl.marks);
  printf("All the members of student: \n");
  printf("Name = %s\n", sl.name);
  printf("Roll no = %d\n", sl.rollno);
  printf("Age = %d\n", sl.age);
  printf("Marks = %d\n", sl.marks);
}
```

↳ Array of Structure

collection of same type of structure is known as array of structure.

Example: 50 students details, n employee details

WAP to create a structure for a book with the members of book ID, author, title, price

```
#include <stdio.h>
struct book
{
 char title[50];
 char author[25];
 char ID[5];
 float price;
};
void main()
{
 int i, n;
 struct book b[100];
 printf("Enter no. of books");
 scanf("%d", &n);
 printf("Enter %d book details", n);
 for(i=0; i<n; i++)
{
 print("Enter title, author, ID, price");
 gets(b[i].title);
 gets(b[i].author);
 gets(b[i].ID);
 scanf("%f", &b[i].price);
}
```

Date :
Page No :

- Array within structure

Eg:
```
struct student
{
    char name[15];
    char USN[10];
    int marks[8];
}
```

- Structure within structure

We can have a structure within another structure

```
void main()
{
struct date
{
    int dd;
    int mm;
    int yy;
};
struct student
{
 char name[30];
 struct date dob;
 int marks;
};
struct student s;
 s.dob.dd;
 s.dob.mm;
 s.dob.yy;
}
```

Date :
Page No :

↳ UNION:

It is a derived data type that contains collection of dissimilar data type stored under a common name. A union can be declared with help of keyword called union.

Syntax:

```
union tagname
{
    datatype member1;
    " " " 2;
};
```

Union variable declaration:

```
union tagname varname;
```

EX:

```
union student
{
    char USN[10];
    char name[15];
    int age;
    int marks;
};
union student s1;
```

WAP to create an employees details such as ID, name, age, salary as members with union and print them.

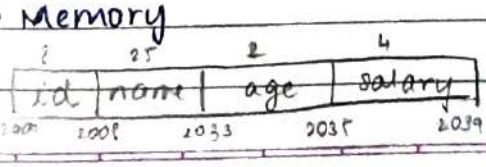
```
#include <stdio.h>
void main()
{
    union employee
    {
        char id[8];
        char name[25];
```

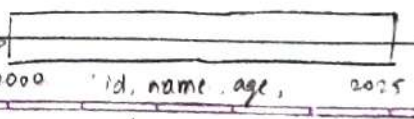
```
    int age;
    float salary;
};
union employee emp;
printf("Enter Employee id, name, age, salary");
gets(emp.id);
gets(emp.name);
scanf("%d", &emp.age);
scanf("%f", &emp.salary);
printf("Employees details \n");
printf("Employees ID = %s\n", emp.id);
printf("Employees name = %s\n", emp.id);
printf("Employee's age = %d\n", emp.age);
printf("Employee's salary = %f\n", emp.salary);
```

Note: A union is similar to structure data type except the memory allocation to the data members.
A union members share a common memory space.
The maximum size of the union will be a member which is having highest size.

↳ Difference b/w structure and union

STRUCTURE	UNION.
• Declared by keyword struct.	• Declared by the keyword union
• Each member of the structure occupies separate memory space.	• Union members occupy common memory space.
• EX:	• EX:
• Memory	





• We can't save the memory.	• We can save the memory when it is not necessary to use all the members at a time.

↳ Enumerated data type:

It is a user-defined data type based on integer type. Enumeration consists of a set of name integer constants. In other words, in enumerated type, each integer value is assigned to the identifier.

- To define a enumerated data type, we use enum keyword.

Syntax:

```
enum name of {id1, id2, ... idn} ;
        enum_var
```

- enum keyword is basically used to declare & initialize a sequence of integer constant.
- EX: enum COLOR {RED, BLUE, GREEN, WHITE, BLACK};

In the above example, RED will be assigned as 0, BLUE as 1 ... Black as 4.

Using enum, we can also assign integer value explicitily to the identifier.

EX: enum color {RED=5, BLUE, GREEN=10, WHITE, BLACK=15}

WAP to display days of week as per the choice using enumerated data type like {SUNDAY=1, MONDAY=2 ....}

```
#include <stdio.h>
#include <conio.h>
void main()
{
  enum DOW {SUNDAY=1, MONDAY=2, TUESDAY=3,
```

```
WEDNESDAY= 4, THURSDAY=5, FRIDAY=6, SATURDAY
                                        =7};
int ch;
printf ("Enter your choice");
scanf ("%d", &ch);
DOW = ch;
switch (DOW)
{
 case SUNDAY:
      printf (" Sunday");
      break;
 case MONDAY:
      printf ("Monday");
      break;
 case SATURDAY:
      "
      "
 default:
      printf ("Wrong choice");
      break;
}
  getch();
}
```

```
#include <stdio.h>
enum week { Mon, Tue, Wed .... sun};
int main()
{
  enum week day;
  day = Wed;
  printf ("%d", day);
  return 0;
}
```

Date :
Page No :

FILES

File: A file is a collection of data stored on a secondary storage device such as pendrives.

Streams : Logical interface

Three types :

↳ standard input (stdin)

↳ standard output (stdout)

↳ standard error (stderr)

Explain io stream used in files

In order to perform input / output operations, a buffer is automatically created and associated with the file.

A buffer is nothing but block of memory that is used for temporary storage of data or return to a file.

It acts as a interface blw stream & disk hardware

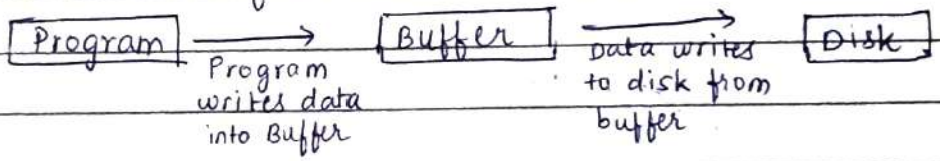
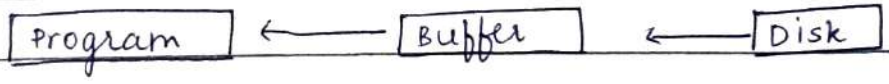


Fig. Buffer associated with storage



Similarly when reading data from the disk file, the data is read from block and written in buffer then program reads data from buffer.

Date :
Page No :

↳ Using files in C:

To use files in C, we must use the following steps:

- Declare a file pointer variable
- Open a file
- Process the file
- Close the file

↳ Discuss different modes of operation on files

File mode conveys to C, the types of processing will be done with files

"w", "r", "a", "wb", "rb", "ab",

↳ Reading data from file

We can read a data from file using 4 set of func()

- fscanf
- fgets()
- fgetc
- fread

↳ fscanf:

It is used to read formatted data from stream

fscanf (file pointer, control variable, var-name);

↳ fgets()

fgets (string-var, size, fp);

↳ fgetc()

WAP to read a student.dat file characters by characters and display it.

```
#include <stdio.h>
#include <fstream.h>
int main()
{
  FILE *fp; char ch;
  fp= fopen ("STUDENT.DAT", "r")
```

```
if( fp==NULL)
{
  printf ("File is not existing");
  exit(0)
}
while ((ch=fgetc(fp))!=EOF)
   printf ("%c", ch);
fclose(fp);
```

↳ fread → particular position
fread (string, 10, 15, fp)

↳ Writing data to file
C provides four set of func. to write data into file. ↳ fprintf ↳ fputs()

↳ fprintf()
fprintf (file pointer, control string, variable-list);

↳ fputs()
fputs(name, fp);

↳ fputc()
```
fputc( ch, fp);
}
   char ch;
   ch=getc();
   fputc (ch, fp);
```

↳ fwrite()
fwrite (str, 11, 7, fp);

Date :
Page No :

4. Detecting

When reading or writing data from files we often do not know exactly how long the file is. For ex, while reading the file, we usually start from the beginning and proceed to the end of file There are 2 ways to detect end of the file:

- while reading file in that mode, characters by characters the programmer can get with the character called EOF, which is symbolic constant depend on stdio.h which has a constant value (-1)

```
while (1)
{
  ch = fgetc (fp);
    if (ch == EOF)
      break;
    printf ("%c", ch)
}
fclose (fp);
```

- The other way is to use a std. library func. FEOF which is defined in stdio.h:

```
int feof (file pointer);

#include <stdio.h>
#include <stdlib.h>
#include <fstream.h>
  void main()
{
  FILE *fp;
  char ch;
  fp = fopen ("STUDENT.DAT", "r");
```