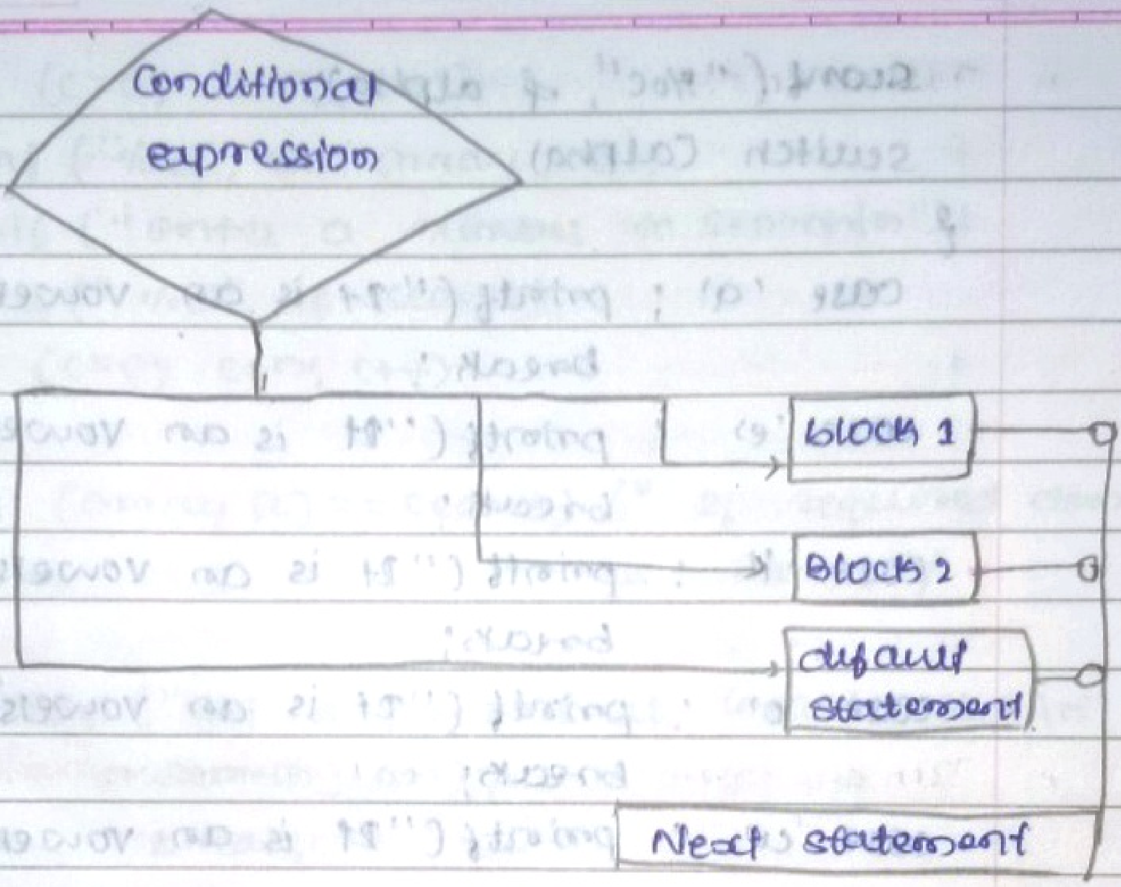


a switch statement tests the values of a given variable agaist list of case values and when a match is found a block of statement associated with that case is executed, if there is no match then default block is executed.

Ex:

⇒ 'C' program to find the vowels using switch statement

```
    # include <stdio.h>
     void main()
    {
       char alpha;
       printf ("Enter the alphabets");
```

```
scanf ("%c", & alpha);
switch (alpha)
{
 case 'a' : printf ("It is an vowels\n");
            break;
 case 'e' : printf ("It is an vowels\n");
            break;
 case 'i' : printf ("It is an vowels\n");
            break;
 case 'o' : printf ("It is an vowels\n");
            break;
 case 'u' : printf ("It is an vowels\n");
            break;
default : dprintf ("It is a consonant\n");
            break;
}
}
```

03. Write a 'c' program for linear search.

```
# include<stdio.h>
  int main ()
  {
    int array [100], search c, n;
    printf (" Enter number of element in array\n
    scanf ("%d", &n);
   printf (" Enter %d integers (s)\n", n);
for (c=0; c<n; c++)
scanf ("%d", and array [c]);
printf ("Enter a number to search\n");
scanf (" %d", & search);
for (c=0; c<n; c++)
{
   if (array [c]== search) /* If required element
                               is found */
{
  printf ("%d is present at location %d.\n",
           search, c+1);
          break;
    }
 }
  if (c==n)
   printf ("%d isn't present in the array\n",
            search);
    return 0;
}
```

04 explain basic structure of C-program

⇒ Documentation section

Link section

Definition Section

Global declaration section

.

main () function section
{
declaration part
executable part
}

subprogram section
function 1
function 2
- - - - - (user defined function)
- - - - -
function n

* documentation section : The documentation section consists of a set of comment lines giving the name of the program, the author details. which the programmer would like to use latter

* Link section: The link section provides instruction to the complier to link function from the system library such as using the
include directives

* defination section : The defination section defines all symbolic constants such using the
define, directives

• Global declaration section: These are some variables that are used in more than one function. Such variables are called global variables and are declared in global declaration section that is outside of all the functions These section also declared all the user - defined function

• main () function section.
Every c program must have one main function section. This section contain two parts declaration part and executed part.

1. Declaration part: The declaration part declared all the variables used in the executable part

2. executable part: There is at least one statement in executable part. This two parts must appear betw the opening and closing braces. The program executable begins at the opening brace and ends at the closing brace. The closing brace at the main function is the logical end of the program All statement in the declaration and executable part end with semicolon.

• Sub-program section: If the program is the program then the subprogram section contains all the function that are called in the main() function user defined function are generally placed Immediatly after the main() Function. although they may appear in any order

Q. 05. What is variable. mention the rules to form variable name Give examples for valid and Invalid variable.

⇒ def : variable is a name of memory where we store data called as variable.

Rules

01. They must begin with a latter or some system permits underscore as the first character
02. It should not be a keyword
03. Uppercase and lowercase are significant
04. White space is not allowed
05. Length should not be normally or that 8 character

examples

⇒ valid variable	Invalid variable
John	(area)
ph-value	25th
sum 12	1sh
-average	%. group s delhi

Q.06. List the classification of loop control statements explain do-while statement along with flowchart and syntax.

⇒ Types of loops
(i) While Loop
(ii) do-while Loop
(iii) For Loop

(i) while loop
Syntax:
```
while (condition is true) {
    // code
    // code
    }
```
⇒ The block keeps executing as long as the condition is true

→ an example
```
# include<stdio.h>
  int main()
{
  int i=0 //i=0
  While (i< 10)
  printf ("The Value of i is %d", i).
  For (i++)',
}
```

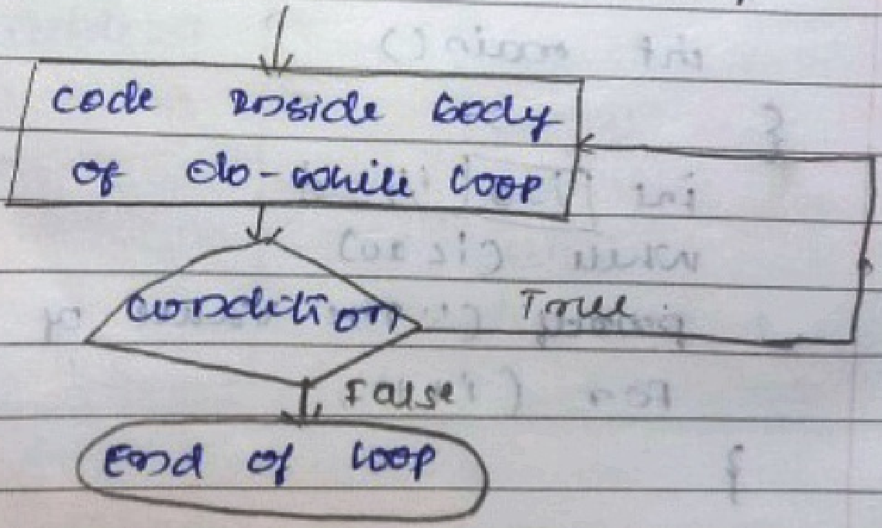
(ii) Do-while loop: A do while loop is same as while loop with one exception that it executes the statement insides the body of do-while before checking the condition, on the other hand on the while Loop. First the condition is checked and then the statements in while loop are executed so you can say that if a condition is false at the first place then the do while would run once however the while loop would ~~not~~ not run at all

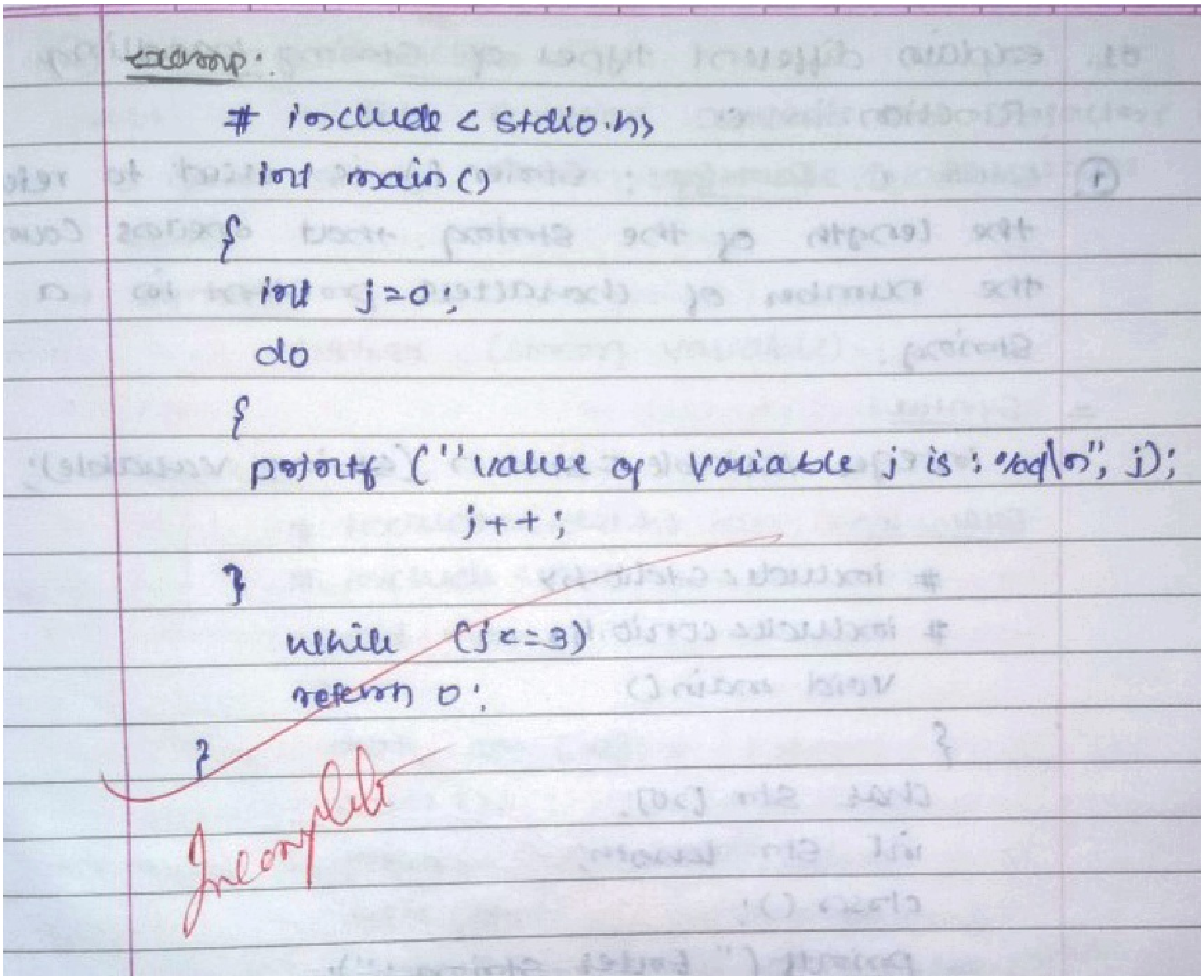
C-do-while loop

Syntax

```
do
{
  // statements
} while condition test);
```

= Flow diagram of do while loop





Example:

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

Incomplete

Q1. explain different types of string handling Function in C.

① Strlen () Function: Strlen () is used to return the length of the string. that means Counts the number of characters present in a string.

= Syntax

integer variable = strlen (string variable);

Example

```
# include<stdio.h>
# include<conio.h>
void main()
{
 char str [20];
 int str length;
 clrscr ();
 printf (" Enter string :");
 get [str);
 Strelength = str [str);
 printf ("Given string. length is :%d", strlen
 getch ();
}
```

② Strlwr () function

"This function converts all characters in a given string from uppercase to lowercase latter

⇒ Syntax:

```
strlwr (string variable)
```

⇒ Example:

```
# include<stdio.h>
# include <conio.h>
 void main()
 {
 char str [20];
 clrscr ();
 printf ("Enter String:");
 gets (Str);
 printf ("lowercase string: %s", strlwr
            (Str));
 getch ();
```

Output:

```
Enter string
welcome
lower case string: welcome.
```

3. Strrev Function

"Strrev() function is used to reverse character in a given String".

⇒ syntax:

Strrev (String variable);

⇒ Example.

```
#include<stdio.h>
# include<conio.h>
  void main ()
{
  char Str [20];
   clrscr ();
  printf (" Enter string :");
  gets (str);
  printf (" Reverse string :%s", Strrev (str));
  getch ();
}
```

Output

```
Enter String
welcome
Reverse String :
  . welcome.
```

4. Strupr ()

" Strupr () function is used to convert all characters in a given string from lower case to uppercase latter

⇒ Syntax

Strupr (String variable)

Example

```
# include<stdio.h>
# include< conio.h>
  void main ()
{
  char str [20];
   clrscr ();
  printf(" Enter String:");
  gets(str);
  printf(" Uppercase string : %s", strupr (str));
  getch ();
}
```

Output

```
        enter  String
         welcome
         uppercase  String!
                 welcome.
```

Q.02. Define pointer and write a 'c' program to swap the two numbers using datatype called as pointer.

⇒ Defi

"pointer is a variable that holds address of another variable of same datatype called as pointer

program

```
# include<stdio.h>
# include<conio.h>
int main()
{
  int x,y, *a , *b , temp';
  clrscr ();
  printf ("Enter the value of xan dy\n);
  scanf (" %d %d", &x, &y)';
  printf (" before swapping\n x= %d\n y= %d\n",
          x,y)';
  a = &x;
  b = &y;
  temp = *a;
  *b = *a;
  *a = *b;
  *a = temp;
  printf(" After swapping\n=x=%d\n, y=%d\n", x&y);
  getch ()
  return 0
}
```

Q.03. How is structure different from an array. Explain declaration and initializalisation of a structure with an example

⇒

Array	Structure
* Array is collection of Homogeneous data	structure is collection of Heterogeneous data
* Array data are acess using index	structure element are accessing using (dot). operators
* eg: a [5]	eg: s. Nam
* Array allocates static memory	structure allocates dynamic memory.
* Array is derived data type	structure is user-defined datatype
* Array elements acess takes less time than structures	structure element takes more time than array
* syntax: data type array name [size];	syntax: struct structure name { Type variable 1; Type variable 2; } variable 1, variable 2;

⇒ <u>Structure</u> <u>declaration</u>

→ To declare a structure, the following syntax is used

```
        Struct Tagname
    {
        data type variable name 1;  }  members
        data type variable name 2;  }  of the
                 1                  }  Structure.
                 1
                 1
        data type variable name n;
    }
        Structure variable
```

<u>Ex</u> :

```
        Struct student
        {
            int no;
            char name [20];
            char address [50];
            int age;
        }
            std;
```

⇒ Structure initialization

⇒ " Assigning values to the given structure members is called structure initialization for assigning values for structure members we have to follow two syntax.

syntax 1

```
Struct tagname
 {
   datatype variable name 1;
   datatype variable name 2;
        .
        .
        .
     datatype variable name n;
 }
 Struct tagname variable name={set of values};
```

= Ex

```
Struct student
 {
   int no;
   char name [50];
   char address [50];
   int age ;
 };
 struct student = { 2, "DER", "V2A", 35 };
```

Syntax 2 :

```
    struct tag name
  {
    data type variable name 1;
    datatype variable name 2;
          .
          .
    datatype variable name n;
  };
    Structure variable name;
Structure variable. member name ={value}
```

⇒ Example :

```
      Struct Student
    {
      int no;
      Char name [50];
      Char address [50];
      int age ;
    };
     Std . no =1;     Std-name = "no
     Std Address = "V2A", Std Age =30
  1 getch ()
}
```

Page No.: At run time
Date:

Q4. What is dynamic memory location in C. explain the types of dynamic memory allocation.

⇒ The process of allocating memory at run time is called "dynamic memory"

Allocation

⇒ C language support 4 dynamic memory allocation function

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

1) Malloc ()

• This function is used to allocating memory space in bytes to the variable of different datatypes

• It allocates only a single block of required memory

• This function allocates the requested size of memory in bytes and returns a pointer, it points to the first byte of the allocated space

* syntax :

ptr = (cast typed) malloc (byte size);

* example:

```
ptr=(int*) malloc (n* size of (int));
ptr = (int*) malloc (100 + size of (int a));
```

* This statement allocates 200 bytes of memory to the ptr
* All the dynamic memory Allocation function are alloc.h and stdlib.h headerfiles

2) calloc() Function:

* This function is used for allocating multiple blocks of memory. It is declared with two arguments
* syntax:

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

* Wher 'n' is the numbers of blocks.
* element size is the size of each block.
* example:

```
ptr = (int*) calloc (4, 2);
```

* The above statement of blocks allocates 4 blocks of memory, each blocks contains 2 bytes
* This function is used for allocating memory for arrays and structures
* This function allocates continuous space in memory.

3) Realloc () Function

* If the priviously allocated memory space is insufficient or more that required we can change the previously allocated memory

* Syntax :

ptr = realloc (ptr, new size)

* where ptr is reallocated with memory size of new size.

* Here accept one made to string or enlarge on previously allocated memory by malloc () or calloc()

4) free () function

* This function is used to release the memory allocates by using malloc() or callo()

* Syntax

free (ptr);

* The free () realease to memory occupied by ptr

05, What is self - refrential Structures. explain the same with example program.

⇒ " A self - refrential structure contains a pointer member that points to a structure of the same structure type. It is used to create data structures like linked lists trees etc.

* For example

```
# include<stdio.h>
# include<conio.h>
struct node
{
int data;
struct node * next;
};
void main()
{
struct node a, b, c;
clrscr ();
a * data = 10;
b, data = 20;
c, data = 30;
a, next = & b;
b. next = & c;
c. next = NULL;
printf("\n the value of b=%d", a, next data);
getch
}
```

06. What is reccursion ? W.A.P to Complete Fibonacci series

⇒ def'. A Function which calls itself, it called ~~a~~ reccussion Function. And this technique is known as reccursion.

⇒ Syntax:

```
void main()
    {
```

program

```
#include<stdio.h>
# include<conio.h>
  int fibonacci (int);
  void main ()
 {
  int Num, i;
  printf ("Enter the number of term:");
  scanf ("%d", & Num);
  For  (i=0; i<Num; i++)
  printf (" %d", Fibonacci (i)); //or.
 }
 int Fibonacci (int n)
{
  if  (n==0)
     return 0;
else if  (n==1)
     return 1;
```

else

return fibonacci (n-1) + fibonacci (n-2);

}

Q.06. Explain different types of storage classes available in 'C'

* A storage class defines the scope and life time of variable and function

⇒ Types of Storage Classes

① auto
② static
③ extern
④ Register

①②③ } RAM (memory)

1) Auto:

* By default all the variables are auto. It uses to declare automatic variable

* Automatic variable are simply local variable which are auto default

* syntax

```
auto datatype variable_name1;
auto datatype variable_name2;
        ;
{2
 3.
```

* example

```
# include< stdio.h>
# include <conio.h>
  void main ()
{
  int a;
 auto int b;
  clrscr
  printf ("%d %d", a,b);
 getch ();
}
```

2) Static:

* The "static modifier can be used with local or global variable
* It contains their values between functions call
* A static variable is Initialized at once when its block is first entered.

⇒ Ex:

```
# include<stdio.h>
# include< conio.h>
  void fun ();
  void main ()
{
   fun ();
   fun ();
   fun ();
}
void fun ()
{
  int a=1;
  static int b=1;
  printf ("%d %d", a,b);
     a++;    b++;
}
```

4) Register :

* A register storage class tells the compiler to store a variable in such a way that access to it fast as possible.

* syntax

     register datatype variable name

* example

```
      # include<stdio.h>
      # include<conio.h>
        void main ()
    {
       int  a=10;
       clrscr ();
       printf ("%d", &a);
       getch ()
}
```

3. polymorphism:

* The ability of a operator and function to take multiple forms is known as polymorphism
* the different types of polymorphism are operator overloading and function overloading

4. Inheritance:

* Inheritance is the process by which one object can acquire and use the properties of another objects
* The existing class is known as base class and new class is known as derived class
* The derived class shares some of the properties of the base class. Therefore a code from a base class can be reused by a derived class

5. Dynamic binding:

* Binding is the process of connecting one program to another
* Dynamic binding is the process of linking the procedure call to a specific sequence of code or function at run time or during the execution of the program

6. message passing

* An object oriented program consist of set

of object that communicate with each other by sending and receiving information

* message passing involves specifying the name of the object, the name of the function and the information to be sent.

22. Difference between pop and oops

⇒ procedure oriented	object oriented
* The program is divided into small part called function	program is divided into part called object small
* Importance not given to data but to functions as well as sequence of action to be done	Importance is given to the data rather than procedure or function because it works as a real world
* Follow top down approach	Follow bottom up approach
* It doesnot have any access specifier	It has access specifier public, private, protected
* data can move freely from function to function in the system	object can move and communicate with each other through member function
* To add new data & function in pops is not easy	OOP provide an easy way to add new data
* EX: C, VB, FORTRAN pascal	EX: C++, JAVA, VB, NET C#NET.

23. Define the Following

* class :

"A class is a collection of objects that have identical properties, common behavior and shared relationship.

* object.

" An object is a collection of data members and associated member functions.

* Inheritance

" Inheritance is the process by which one object can acquire and use the properties of another object.

* encapsulation :

" The wrapping of data and function into a single unit (class) is called data encapsulation.

* polymorphism :

" The ability of an operator and function to take multiple forms is known as polymor-phism.

* data abstraction

" Data abstraction refers to the process of representing essential features without including background details or explanation.

24. Write the application of oop's

⇒ * Computer graphics applications
* CAD / CAM software
* object - oriented database
* User Interface design such as windows
* Real time system
* Simulation and modeling
* Artificial intelligence and expert system
* Client - server systems

25. Explain basic structure of C++ program.

⇒ The program written in C++ language follows this basic structure the sequence of sections should be as they are in the basic structure. A C program should have one or more section but the sequence of section is to be followed.

1. Documentation section
2. Linking section
3. Definition section
4. Global declaration section
5. member function definition
6. main function

section main ()

{

Declaration section

1. Documentation section : comes first and is used to document the use of logic or reasons in your program. It can be used to write the programes objective developer and logic details The documentation is done in C language with /* and */. whatever is written inbetween these two are called comments.

2. Linking section : This section tells the compiler to link the certain occurances of keywords or functions in your program to the header files specified in this section
eg: # include <iostream>

3. Declaration section: It is used to declare some constants and assign them some value
eg: define MAX 25

4). Global declaration section ; Here the variables and class definition which are used through out the program are declared so as to make them global.

5. Sub program or function section ; This has all the sub-programs or the functions which own program needs

```
void display ()
{
Cout << " C++ is better than c";
}
```

Simple C++ program;

```
# include <iostream>
using name space std;
void display ()
{
Cout <<" C++ is better than C"
}
int main ()
{
display ()
return 0;
}
```

6. main function section : It tells the compiler where to start the execution from main()

```
{
point from execution starts
}
```

It has two section.

Declaration Section: In this the variables and their data type are declared

2. executable section: This has the part of program which actually perform the task we need

26. What is computer? Difference between primary and secondary memory? write some input and output devices

⇒ A computer is digital electronic machine that can be programmed to carry out sequence of arithmetic and logical operation automatically.

modern computer can perform generic sets of operation known as programs.

→ Input device are: mouse, keyboard, webcam, scanner, joystick, microphone,

⇒ output device are: monitor, speaker, printer, projector, Headset, plotter.

primary memory	Secondary memory
* It is the main memory where data & information store temporally.	It refers to external memory where data is stored permantly
* Data is directly accesed by processor	Data cannot be directly accesed by processor
* It is relativily faster than secondary memory because of it valatile nature	They are usually slower than primary memory.
* primary memory is also know as main memory or internal memory.	Secondary memory is also knows as external memory or auxiliary memory.
ex: RAM, ROM	ex: Hard disc floppy di

27. Explain different types of operators

ans: C operators can be classified as

1. Arithmatic operator
2. Relational operators
3. logical operators
4. Increment and decrement operators
5. Assignment operators
6. conditional operators
7. logical operators Bit wise operators
8. unary operators

9. special operators

10. Additional operators in C++

01. Arithmatic operators.

All the basic arithmatic operators are present in c

Ex +, -, =, /, % (remainder)

02. Relational operators

we often compare two quantities and depending on their relation take certain decision for that comparision we use relational operators

Ex <, >, <=, >=, ==, !=

03. Logical operators:

Logical data: A piece of data is called logical if its conveys the idea of true or false

04. Assignment operators

The assignment expression evaluate the expand on the right side of the operation and places it values in the variable on the left side

32. what is Ternary operators in C,

The programmer utilize the ternary operator in case of decision making when longer conditional like if and else exist. in simpler words when we use an operators on three variables or operands. It is known as a ternary operators in C.

33) Explain different types of computer specification

=> PC (personal computer)

It is a single user computer system having moderately powerful microprocessor

* Workstation

It is also a single user comp-uter system similar to personal computer however has a more powerful microp-rocessor.

* mini computer

It is multiuser computer system

28. Define the data type and mention its types.

Ans: A data type used to indicate the type of data value stored in a variable.

ANSI C supports three classes of data types

1 primary data type
2 Derived data type
3 user defined data type.

= primary data type.

1) Integer type : Integers are whole numbers with a range of values supported by a particular machin.

C has three classes of Integer storage namely short int, int, long int in both signed and unsigned forms

2) Floating point type. Floating point number are stored in 32 bits of 6 digit of particular

When the accuracy provided by a float number is not sufficient, the double can be used to define the number.

3) Void type

The void type has no value. This is usually used to specify the type of functions. The type of a function is said to be void when it does not return any value to the calling function.

4). Character type:

A single character can be defined as a character (char) type data. Characters are usually stored in 8 bits of internal storage.

=> Derived data type.
1) Array
2) pointer
3) Reference

= user defined data types
1) Structure
2) union
3) class
4) enumeration.

30. Why do we call C++ as superset of C

Ans: When C++ was developed by Bjarne Stroustrup he added oop (object oriented programming) feature to C without any significantly changing the 'c' component. Thus C++ is a 'relative' of C (called a superset) meaning that any valid c program is also a valid C++ program.

31. What is algorithm?

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

Algorithm are generally created independent of underlying languages i.e an algorithm can be implemented in more than one programming language

[illegible]
9/5/22

Arrays in C

* Array is derived datatype which is constructed by the help of primitive datatype

② Array is a variable which stores more than one variable of same data type in continuous memory location

③ Array can't contain dis-similar type of da

④ Array index always begin with unique identification a[10], called base index. of array.

⑤ The size of array index is always 0 to n-1

⑥ starting zero

two ways

* one-dimension Array
* multi-dimension Array

Syntax: <Datatype><variable name> [size of arr

[ ] ⇒ subscript operator

Ex: int array [10];

⇒ 1D arrays

```
#include<stdio.h>
#include<conio.h>
 void main()
{
   int a [5]; .i;
   clrscr ();
   printf (" Enter elements in array"); 25
   for (i=0; i<=4; i++)
   { scanf (" %d", & a[i]);
   }
    printf(" array elements");
    for (i=0; i<=4; i++)
   {
    printf ("%d", a [i]);
   }
    getch ();
```

1D array

Arrays:

* Arrays is a linear collection of similar element
* Array is also known as subscript variable

```
#include <stdio.h>
#include <conio.h>
 int main ()
{
int array[100], position, i, n;
printf ("Enter number of elements \n");
 scanf ("%d", &n);
printf ("Enter %d Element \n", n);
 for (i=0; i<n; i++);
 {
scanf ("%d", &array[i]);
}
printf ("Enter the location where you
          wish to delete portion \n");
 if (position > n+1)
 {
printf ("Delion is not possiable \n");
~~for~~   }else
        {
```

```
 for (i= postion-1; i<n-1 ; i++)
  array [i] = array[i+1]
 print ("Resultan array \n");
 ~~scanf~~ ("%d
  for (i=0 ; i<n-1 ; i++)
 print ("%d \n", &array[i])
    }
   getch() }
    return 0;
      }.
```