Programming for problem solving (PPS)
UNIT-1
Question Bank Solutions

PART-A(2 Marks):

1. Differences between compiler and interpreter?

The difference between an interpreter and a compiler is given below:

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

2. Define algorithm and flowchart?

Algorithm and flowchart are two types of tools to explain the process of a program. An **algorithm** in general is a sequence of steps to solve a particular problem.

A **flowchart** is a formalized graphic representation of a logic sequence, work or manufacturing process, organization **chart**, or similar formalized structure.

3. What is the importance of operator precedence and associativity?

Operator precedence determines which operator is evaluated first when an expression has more than one operator.

Associativity is used when there are two or more operators of same precedence is present in an expression.

4. List the various operators available in C.

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Conditional Operators

Increment and decrement operators
Special Operators

5. Write a C program to find largest number among three given numbers.

```
#include <stdio.h>
int main()
{
   double n1, n2, n3;

   printf("Enter three different numbers: ");
   scanf("%lf %lf %lf", &n1, &n2, &n3);

   if( n1>=n2 && n1>=n3 )
      printf("%.2f is the largest number.", n1);

   if( n2>=n1 && n2>=n3 )
      printf("%.2f is the largest number.", n2);

   if( n3>=n1 && n3>=n2 )
      printf("%.2f is the largest number.", n3);

   return 0;
}
```

6.Differentiate while and do-while statements.

The difference is in while the condition gets evaluated. while loop is a **entry control** loop. In a **do**..**while loop**, the condition is not evaluated until the end of each **loop**. That means that a **do**..**while loop will** always run at least once. While loop is a exit control loop. In a **while loop**, the condition is evaluated at the start.

PART-B (5 Marks Answers)

1a. Define computer? Explain functional units of computer.

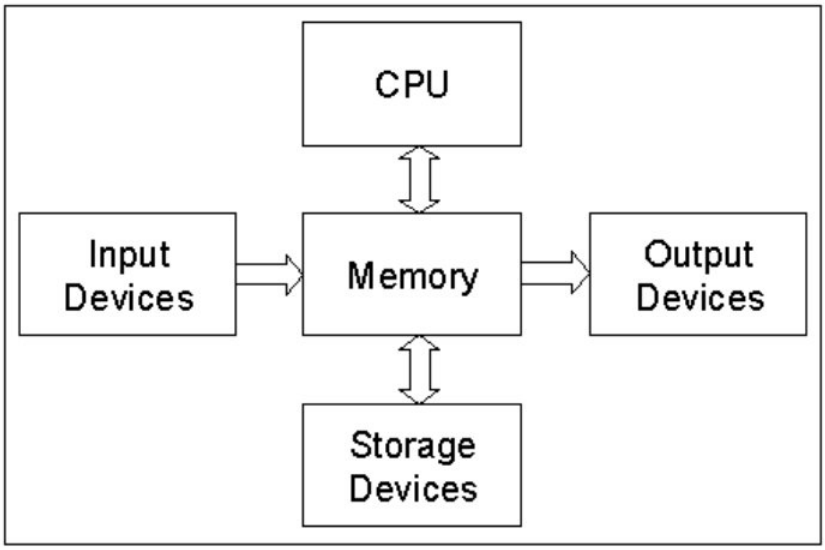
A computer is made up of multiple parts and components that facilitate user functionality. A computer has two primary categories:

Hardware: Physical structure that houses a computer's processor, memory, storage, communication ports and peripheral devices

Software: Includes operating system (OS) and software applications

A computer works with software programs that are sent to its underlying hardware architecture for reading, interpretation and execution. Computers are classified according to computing power, capacity, size, mobility and other factors, as personal computers (PC), desktop computers,

laptop computers, minicomputers, handheld computers and devices, mainframes or supercomputers.



Computer systems ranging from a controller in a microwave oven to a large supercomputer contain components providing five functions. A typical personal computer has hard, floppy and CD-ROM disks for storage, memory and CPU chips inside the system unit, a keyboard and mouse for input, and a display, printer and speakers for output. The arrows represent the direction information flows between the functional units.

A computer is a machine or device that performs processes, calculations and operations based on instructions provided by a software or hardware program. It is designed to execute applications and provides a variety of solutions by combining integrated hardware and software components.

1b. Explain bitwise operators with suitable example.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

Show Examples

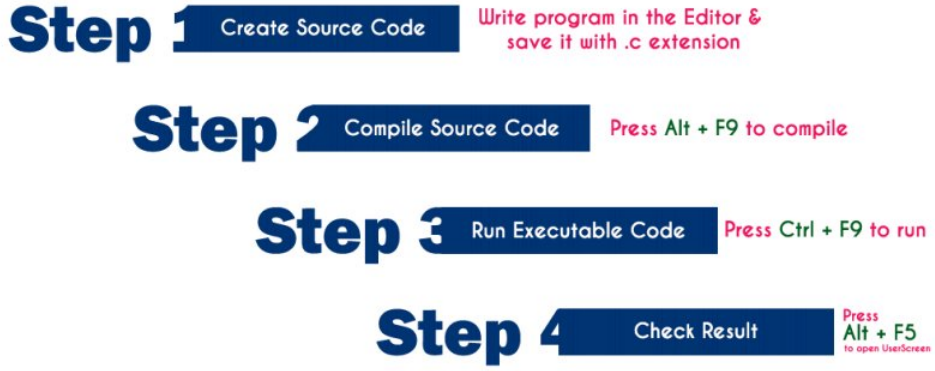
Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -60, i.e,. 1100 0100 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

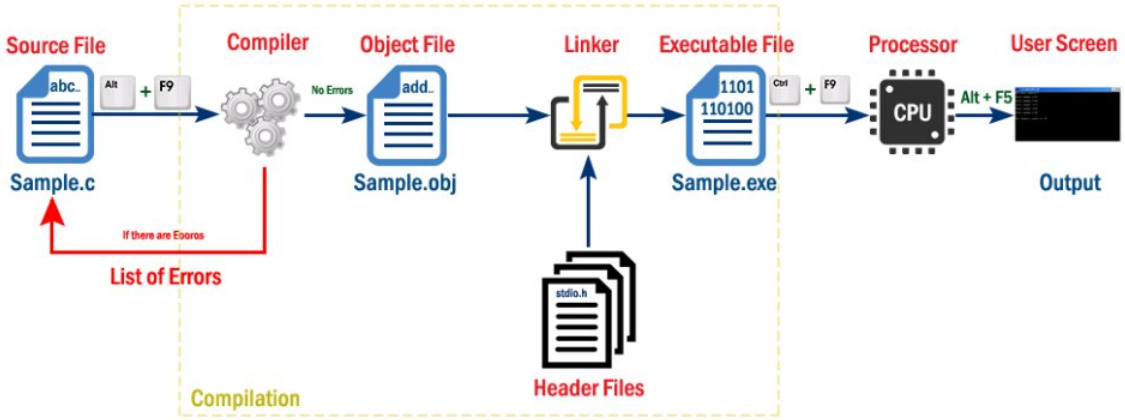
2a. What are the steps involved while creating and running a c program?

Generally, the programs created using programming languages like C, C++, Java etc., are written using high level language like English. But, computer cannot understand the high level language. It can understand only low level language. So, the program written in high level language needs to be converted into low level language to make it understandable for the computer. This conversion is performed using either Interpreter or Compiler.

Popular programming languages like C, C++, Java etc., use compiler to convert high level language instructions into low level language instructions. Compiler is a program that converts high level language instructions into low level language instructions. Generally, compiler performs two things, first it verifies the program errors, if errors are found, it returns list of errors otherwise it converts the complete code into low level language.

To create and execute C programs in Windows Operating System, we need to install Turbo C software. We use the following steps to create and execute C programs in Windows OS…





The file which contains c program instructions in high level language is said to be source code. Every c program source file is saved with .c extension, for example Sample.c.

Whenever we press Alt + F9 the source file is submitted to the compiler. Compiler checks for the errors, if there are any errors, it returns list of errors, otherwise generates object code in a file with name Sample.obj and submit it to the linker. Linker combines the code from specified header file into object file and generates executable file as Sample.exe. With this compilation process completes.

2b. Write syntax and flow diagram for the switch statement?

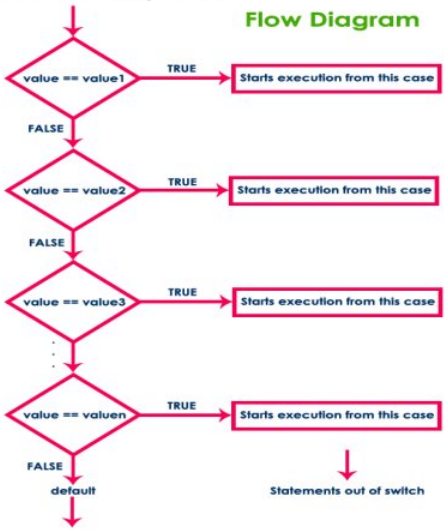
Consider a situation in which we have more number of options out of which we need to select only one option that is to be executed. Such kind of problems can be solved using nested if statement. But as the number of options increases, the complexity of the program also gets increased. This type of problems can be solved very easily using switch statement. Using switch statement, one can select only one option from more number of options very easily. In switch

statement, we provide a value that is to be compared with a value associated with each option. Whenever the given value matches with the value associated with an option, the execution starts from that option. In switch statement every option is defined as a case.

The switch statement has the following syntax and execution flow diagram...

Syntax

```
switch ( expression or value )
{
        case  value1:  set of statements;
                       ....
        case  value2:  set of statements;
                       ....
        case  value3:  set of statements;
                       ....
        case  value4:  set of statements;
                       ....
        case  value5:  set of statements;
                       ....
        .
        .
        default: set of statements;
}
```



3a. Describe different storage classes available in C.

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify. We have four different storage classes in a C program −

- auto
- register
- static
- extern

The auto Storage Class

The **auto** storage class is the default storage class for all local variables.

```
{
  int mount;
  auto int month;
}
```

The example above defines two variables within the same storage class. 'auto' can only be used within functions, i.e., local variables.

The register Storage Class

The **register** storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

```
{
```

```
  register int  miles;
}
```

The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

The static Storage Class

The **static** storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

In C programming, when **static** is used on a global variable, it causes only one copy of that member to be shared by all the objects of its class.

```
#include <stdio.h>

/* function declaration */
void func(void);

static int count = 5; /* global variable */

main() {

  while(count--) {
    func();
  }

  return 0;
}

/* function definition */
void func( void ) {

  static int i = 5; /* local static variable */
  i++;

  printf("i is %d and count is %d\n", i, count);
}
```

When the above code is compiled and executed, it produces the following result −

```
i is 6 and count is 4
i is 7 and count is 3
i is 8 and count is 2
```

```
i is 9 and count is 1
i is 10 and count is 0
```

The extern Storage Class

The **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function, which will also be used in other files, then *extern* will be used in another file to provide the reference of defined variable or function. Just for understanding, *extern* is used to declare a global variable or function in another file.

The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions as explained below.

First File: main.c

```
#include <stdio.h>

int count ;
extern void write_extern();

main() {
  count = 5;
  write_extern();
}
```

Second File: support.c

```
#include <stdio.h>

extern int count;

void write_extern(void) {
  printf("count is %d\n", count);
}
```

Here, extern is being used to declare *count* in the second file, where as it has its definition in the first file, main.c. Now, compile these two files as follows −

```
$gcc main.c support.c
```

It will produce the executable program **a.out**. When this program is executed, it produces the following result −

```
count is 5
```

3b. Define a term 'type conversion' in C. Explain different ways of type conversions with suitable examples.

Type casting is a way to convert a variable from one data type to another data type. For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'.

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

```
main() {

  int sum = 17, count = 5;
  double mean;

  mean = (double) sum / count;
  printf("Value of mean : %f\n", mean );
}
```

When the above code is compiled and executed, it produces the following result −

Value of mean : 3.400000

It should be noted here that the cast operator has precedence over division, so the value of **sum** is first converted to type **double** and finally it gets divided by count yielding a double value.

Type conversions can be implicit which is performed by the compiler automatically, or it can be specified explicitly through the use of the **cast operator**. It is considered good programming practice to use the cast operator whenever type conversions are necessary.

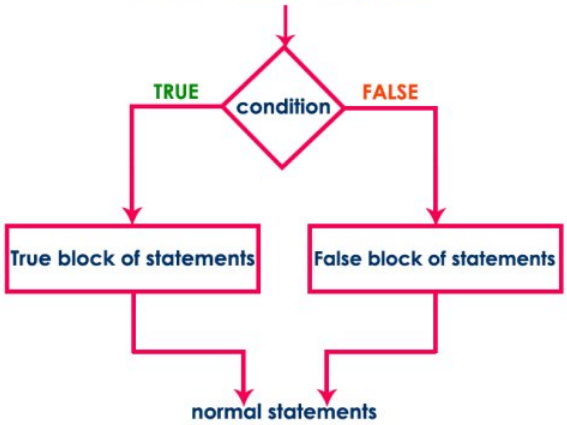
4a. How does If-else statement is executed? Explain with suitable example.

The if - else statement is used to verify the given condition and executes only one out of the two blocks of statements based on the condition result. The if-else statement evaluates the specified condition. If it is TRUE, it executes a block of statements (True block). If the condition is FALSE, it executes another block of statements (False block). The general syntax and execution flow of the if-else statement is as follows...

Syntax

```
if ( condition )
{
   ....
   True block of statements;
   ....
}
else
{
   ....
   False block of statements;
   ....
}
```

Execution flow diagram



The if-else statement is used when we have two options and only one option has to be executed based on a condition result (TRUE or FALSE).

Example Program | Test whether given number is even or odd.

```
#include <stdio.h>
#include<conio.h>
void main(){
  int n ;
  clrscr() ;
  printf("Enter any integer number: ") ;
  scanf("%d", &n) ;
  if ( n%2 == 0 )
    printf("Given number is EVEN\n") ;
  else
    printf("Given number is ODD\n") ;
}
```

Output 1:
Enter any integer number: 100
Given number is EVEN

Output 2:
Enter any integer number: 99
Given number is ODD

4b. Write a C program to print prime numbers from 1 to 100.

/* C Program to Print Prime Numbers from 1 to 100 using For Loop */

```
#include <stdio.h>

int main()
{
 int i, Number, count;

 printf(" Prime Number from 1 to 100 are: \n");
 for(Number = 1; Number <= 100; Number++)
 {
  count = 0;
  for (i = 2; i <= Number/2; i++)
  {
    if(Number%i == 0)
    {
     count++;
     break;
    }
  }
  if(count == 0 && Number != 1 )
  {
    printf(" %d ", Number);
  }
```

```
  }
 return 0;
}
```

5a. Write an algorithm and develop a C program to find whether a given number is even or odd.

START
  Step 1 → Take integer variable A
  Step 2 → Assign value to the variable
  Step 3 → Perform A modulo 2 and check result if output is 0
  Step 4 → If true print *A is even*
  Step 5 → If false print *A is odd*
  STOP

```
#include <stdio.h>

int main() {
  int even = 24;
  int odd  = 31;

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

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

  return 0;
}
```

Output

Output of the program should be −

```
24 is even
31 is odd
```

5b. Write a program to read the age of 100 persons and count the number of persons in the age group 50 to 60. Use for and continue statements.

```
/* C Program: read the age of 100 persons and count
the number of person in the age group 50 -60. */

#include<stdio.h>
#include<conio.h>
void main()
{
  int age, count =0, i,n;
  clrscr();
```

```
  printf("Enter How many person:-> ",i);
  scanf("%d",&n);

  for(i=1;i<=n;i++)
  {
   printf("Enter age of %d person :-> ",i);
   scanf("%d",&age);
   if(age>=50&&age<=60)
count++;
  }
  printf("Total person in the age group 50 to 60 = %d\n",count);
  getch();
}
```

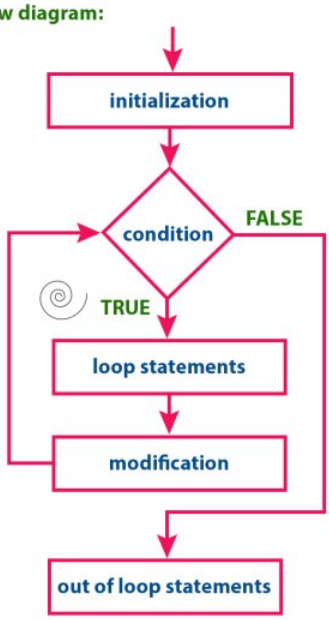
06. Demonstrate for-loop with syntax, flow diagram and write a c program to print multiplication table for a given number.

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

Syntax:

```
for( initialization ; condition ; modification )
{
    ...
    block of statements;
    ...
}
```

Execution flow diagram:



UNIT-2

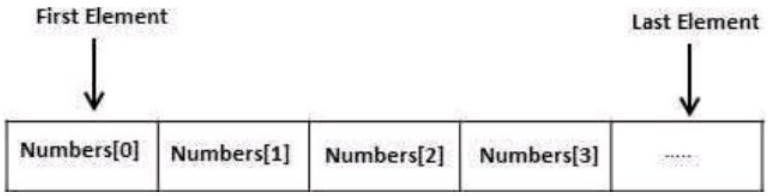
PART-A(2 Marks):

1. **What are the advantages of using arrays in C?**

Array is a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

lowest address corresponds to the first element and the highest address to he last element.




```
#include <stdio.h>
void main ()
{
    int n[ 10 ]; /* n is an array of 10 integers */
    int i,j;
    /* initialize elements of array n to 0 */
    for ( i = 0; i < 10; i++ )
    {
        n[ i ] = i + 100;   /* set element at location i to i + 100 */
    }
          /* output each array element's value */
    for (j = 0; j< 10; j++ )
      {
          printf("Element[%d] = %d\n", j, n[j] );
      }
      getch();
  }
```

When the above code is compiled and executed, it produces the following result:

Element[0] =100

Element[1] =101

Element[2] =102
Element[3] =103
Element[4] =104
Element[5] =105
Element[6] =106
Element[7] =107
Element[8] =108
Element[9] =109

2. **Differentiate strcpy() and strncpy()?**

- Strcpy( ) function copies whole content of one string into another string. Whereas, strncpy( ) function copies portion of contents of one string into another string.
- char *strcpy(char *dest, const char *src)
- char *strncpy(char *dest, const char *src, size_t n)
- If destination string length is less than source string, entire/specified source string value won't be copied into destination string in both cases.

3. What is a pointer and pointer to pointer?

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is:

type *var-name;

Here, **type** is the pointer's base type; it must be a valid C data type and **var- name** is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement, the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declaration:

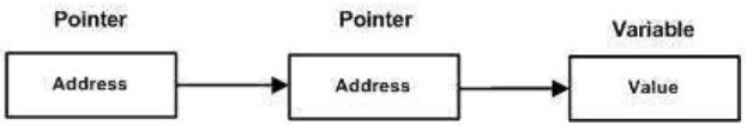
```
int *ip;    /* pointer to  an integer */
double *dp;  /* pointer to a double */
float *fp;   /* pointer to  a float */
char *ch;    /* pointer to  character */
```

The actual data type of the value of all pointers, whether integer, float, character,

or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

Pointer to Pointer:

A pointer to a pointer is a form of multiple indirections, or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, the following declaration declares a pointer to a pointer of type int:

```
int **var;
```

When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example:

When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example:

```
#include <stdio.h>

void main ()
 {
    int     var;
    int     *ptr; int
            **pptr;
    var = 3000;

    /* take the address of var */
    ptr = &var;
    /* take the address of ptr using address of operator & */
```

```
    pptr = &ptr;
    /* take the value using pptr */
     printf("Value of var = %d\n", var );
    printf("Value available at *ptr = %d\n", *ptr ); printf("Value
    available at **pptr = %d\n", **pptr);
    getch();
}
```

When the above code is compiled and executed, it produces the following result:

```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

4. Define union? Give its representation.

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

Defining a Union

To define a union, you must use the **union** statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows:

```
union [union tag]
{
    member definition;
    member definition;
    ...
    member definition;
} [one or more union variables];
```

The **union tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the union's definition, before the final semicolon, you can specify one or more union variables, but it is optional. Here is the way you would define a union type named Data having three members i, f, and str

```
union data
```

```
{
    int i;
    float f;
    char str[20];
} data;
```

5. What is a self referential structure? Where to use self referential structure.

Self Referential Structure:

A self referential data structure is essentially a structure definition which includes at least one member that is a pointer to the structure of its own kind. A chain of such structures can thus be expressed as follows.

```
struct name {
                member 1;
                member 2;
                . . .
                struct name *pointer;
            };
```

The above illustrated structure prototype describes one node that comprises of two logical segments. One of them stores data/information and the other one is a pointer indicating where the next component can be found. .Several such inter-connected nodes create a chain of structures.

Self-referential structures are very useful in applications that involve linked data structures, such as lists and trees. Unlike a *static data structure* such as array where the number of elements that can be inserted in the array is limited by the size of the array, a self-referential structure can dynamically be expanded or contracted. Operations like insertion or deletion of nodes in a self- referential structure involve simple and straight forward alteration of pointers.

Linear (Singly) Linked List

A linear linked list is a chain of structures where each node points to the next node to create a list. To keep track of the starting node's address a dedicated pointer (referred as *start pointer*) is used. The end of the list is indicated by a *NULL*pointer. In order to create a linked list of integers, we define each of its element (referred as *node*) using the following declaration.

```
struct node_type {
        int data;
        struct node_type *next;
                };
struct node_type *start = NULL;
```

Note: The second member points to a node of same type.

A linear linked list illustration:

6. Give a linked list representation with neat diagram.

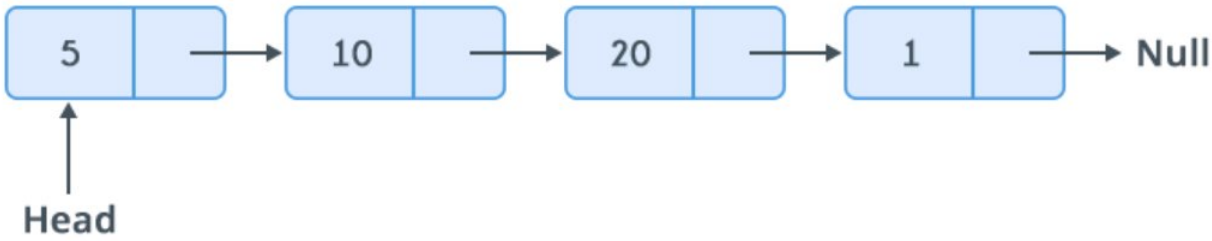
Linked List:

A **linked list** is a way to store a collection of elements. Like an array these can be character or integers. Each element in a linked list is stored in the form of a **node**.

Node:

A node is a collection of two sub-elements or parts. A **data** part that stores the element and a **next** part that stores the link to the next node.

Linked List:



A linked list is formed when many such nodes are linked together to form a chain. Each node points to the next node present in the order. The first node is always used as a reference to traverse the list and is called **HEAD**. The last node points to **NULL**.

Declaring a Linked list :

In C language, a linked list can be implemented using structure and pointers:

```
struct  LinkedList

{

        int data;
```

```
    struct LinkedList *next;
};
```

The above definition is used to create every node in the list. The **data** field stores the element and the **next** is a pointer to store the address of the next node.

UNIT-2

PART-B (5 Marks Answers)

1a. what is an array? Explain different types of arrays

Array is a reference data type used to create fixed number of multiple variables of same type to store multiple values of similar type in continuous memory locations with single variable name.

Array declaration syntax:

<datatype>[]<array variable name>;

public static int[] i;

public static Example[] e;

public static int[] i;

public static Example[] e;

The Various types of Array those are provided by c as Follows:-

1. Single Dimensional Array

2. Two Dimensional Array

3. Three Dimensional array

1. One dimensional (1-D) arrays or Linear arrays:

In it each element is represented by a single subscript. The elements are stored in consecutive memory locations. E.g. A [1], A [2], ……, A [N].

2. Two dimensional (2-D) arrays:

In it each element is represented by two subscripts. Thus a two dimensional m x n array A has m rows and n columns and contains m*n elements. It is also called matrix array because in it the elements form a matrix. E.g. A [2] [3] has 2 rows and 3 columns and 2*3 = 6 elements.

3. Three dimensional arrays:

In it each element is represented by three subscripts. Thus a three dimensional m x n x l array A contains m*n*l elements. E.g. A [2] [3] [2] has 2*3*2 = 12 elements.

1 b. Explain pointer to arrays? Can we access the array elements using pointers? Explain

Pointer to Array

Consider the following program:

```
#include<stdio.h>

int main()
{
  int arr[5] = { 1, 2, 3, 4, 5 };
  int *ptr = arr;

  printf("%p\n", ptr);
  return 0;
}
```

In this program, we have a pointer *ptr* that points to the 0^{th} element of the array. Similarly, we can also declare a pointer that can point to whole array instead of only one element of the array. This pointer is useful when talking about multidimensional arrays.
Syntax:

```
data_type (*var_name)[size_of_array];
```

Example:

```
int (*ptr)[10];
```

Here *ptr* is pointer that can point to an array of 10 integers. Since subscript have higher precedence than indirection, it is necessary to enclose the indirection operator and pointer name inside parentheses. Here the type of ptr is 'pointer to an array of 10 integers'.
Note : The pointer that points to the 0^{th} element of array and the pointer that points to the whole array are totally different. The following program shows this:

```
// C program to understand difference between
// pointer to an integer and pointer to an
// array of integers.
#include<stdio.h>

int main()
{
   // Pointer to an integer
```

```
    int *p;

    // Pointer to an array of 5 integers
    int (*ptr)[5];
    int arr[5];

    // Points to 0th element of the arr.
    p = arr;

    // Points to the whole array arr.
    ptr = &arr;

    printf("p = %p, ptr = %p\n", p, ptr);

    p++;
    ptr++;

    printf("p = %p, ptr = %p\n", p, ptr);

    return 0;
}
```

2 a. Explain string manipulation functions?

To manipulate strings in C using library functions such as gets(), puts, strlen() and more. You'll learn to get string from the user and perform operations on the string.

Manipulate strings according to the need of a problem. Most, if not all, of the time string manipulation can be done manually but, this makes programming complex and large.

Function	Work of function
strlen()	Calculates the length of string
strcpy()	Copies a string to another string
strcat()	Concatenates(joins) two strings
strcmp()	Compares two string
strlwr()	Converts string to lowercase
strupr()	Converts string to uppercase

Strings handling functions are defined under "string.h" header file.

gets () and puts()

Functions gets () and puts () are two string functions to take string input from the user and display it respectively.

```
#include<stdio.h>
void main()
{
   char name[30];
   printf ("Enter name: ");
   gets(name);     //Function to read string from user.
   printf("Name: ");
   puts(name);    //Function to display string.
   getch();
}
```

Note: Though, gets() and puts() function handle strings, both these functions are defined in "stdio.h" header file.

2b. Can we store a list of strings under one name? Justify your answer.

An array list of strings (each individual element in the array list is just a word with no white space) and to take each element and append each next word to the end of a string.

So say the array list has

```
element 0 = "hello"
element 1 = "world,"
element 2 = "how"
element 3 = "are"
element 4 = "you?"
```

To make a string called sentence that contains "hello world, how are you?"

3a. Distinguish between structure and union.

A structure is a user-defined data type available in C that allows to combining data items of different kinds. Structures are used to represent a record.
Defining a structure: To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows:

```
struct [structure name]
  {
```

```
    member definition;
    member definition;
      ...
    member definition;
};
```

Union

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

.

Defining a Union: To define a union, you must use the **union** statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows:

```
union [union name]
  {
    member definition;
    member definition;
    ...
    member definition;
  };
```

Differences

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

3b. Demonstrate array of structures with suitable example

C Structure is collection of different data types (variables) which are grouped together. Whereas, array of structures is nothing but collection of structures. This is also called as structure array in C.

Example program for array of structures in C.

This program is used to store and access "id, name and percentage" for 3 students. Structure array is used in this program to store and display records for many students. You can store "n" number of students record by declaring structure variable as 'struct student record[n]", where n can be 1000 or 5000 etc.

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

struct student
{
    int id;
    char name[30];
    float percentage;
};

void main()
{
    int i;
    struct student record[2];

    // 1st student's record
```

```
    record[0].id=1;
    strcpy(record[0].name, "Raju");
    record[0].percentage = 86.5;

    // 2nd student's record
    record[1].id=2;
    strcpy(record[1].name, "Surendren");
    record[1].percentage = 90.5;

    // 3rd student's record
    record[2].id=3;
    strcpy(record[2].name, "Thiyagu");
    record[2].percentage = 81.5;

    for(i=0; i<3; i++)
    {
        printf("     Records of STUDENT : %d \n", i+1);
        printf(" Id is: %d \n", record[i].id);
        printf(" Name is: %s \n", record[i].name);
        printf(" Percentage is: %f\n\n",record[i].percentage);
    }
    getch();
}
```

4a. what is structure pointer? How do we access structure members using pointers?

Structures can be created and accessed using pointers. A pointer variable of a structure can be created as below:

```
struct name {
   member1;
   member2;
};
int main()
{
  struct name *ptr;
}
```

Here, the pointer variable of type **struct name** is created.

Accessing structure's member through pointer:

A structure's member can be accesssed through pointer in two ways:

1. Referencing pointer to another address to access memory
2. Using dynamic memory allocation

1. Referencing pointer to another address to access the memory:

Consider an example to access structure's member through pointer.

```
#include <stdio.h>
typedef struct person
{
  int age;
  float weight;
};
int main()
{
   struct person *personPtr, person1;
   personPtr = &person1;          // Referencing pointer to memory address of person1
   printf("Enter integer: ");
   scanf("%d",&(*personPtr).age);
   printf("Enter number: ");
   scanf("%f",&(*personPtr).weight);
   printf("Displaying: ");
   printf("%d%f",(*personPtr).age,(*personPtr).weight);
   return 0;
}
```

In this example, the pointer variable of type `struct person` is referenced to the address of `person1`. Then, only the structure member through pointer can can accessed.

Using -> operator to access structure pointer member

Structure pointer member can also be accessed using `->` operator.

```
(*personPtr).age is same as personPtr->age

(*personPtr).weight is same as personPtr->weight
```

2. Accessing structure member through pointer using dynamic memory allocations:

To access structure member using pointers, memory can be allocated dynamically using malloc() function defined under "stdlib.h" header file.

Syntax to use malloc()

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

Example to use structure's member through pointer using malloc() function.

```
#include <stdio.h>
#include <stdlib.h>
struct person {
  int age;
  float weight;
  char name[30];
};

int main()
{
  struct person *ptr;
  int i, num;
  printf("Enter number of persons: ");
  scanf("%d", &num);
  ptr = (struct person*) malloc(num * sizeof(struct person));
  // Above statement allocates the memory for n structures with pointer personPtr pointing to
base address */
  for(i = 0; i < num; ++i)
  {
     printf("Enter name, age and weight of the person respectively:\n");
     scanf("%s%d%f", &(ptr+i)->name, &(ptr+i)->age, &(ptr+i)->weight);
  }
  printf("Displaying Infromation:\n");
  for(i = 0; i < num; ++i)
```

```
    printf("%s\t%d\t%.2f\n", (ptr+i)->name, (ptr+i)->age, (ptr+i)->weight);
  return 0;
}
```

4b. Write a C program that accepts list of student names and Roll numbers and display the same details in tabular format

```
#include <stdio.h>
struct student
{
   char name[50];
   int roll;
} s;
void main()
{
   printf("Enter information:\n");
   printf("Enter name: ");
   scanf("%s", s.name);
   printf("Enter roll number: ");
   scanf("%d", &s.roll);
      printf("Displaying Information:\n");
   printf("Name: ");
   puts(s.name);
   printf("Roll number: %d\n",s.roll);
   printf("Marks: %.1f\n", s.marks);
   getch();
}
```

5a. Write a C program to find largest integer in a array of elements?

C Program to Find the Largest Number in an Array

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

```
void main()
{

int array[50], size, i, largest;

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

      printf("\n Enter %d elements of  the array: ", size);

     for (i = 0; i < size; i++)
          scanf("%d", &array[i]);

     largest = array[0];

     for (i = 1; i < size; i++)
     {
          if (array[i] > largest)
                    largest = array[i];
}

      printf("\n largest element present in the given array is : %d", largest);

     getch();

}
```

5b. Write a c program to read and display 5 books information using structures.

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

#define SIZE 5

struct bookdetail

{

        char name[20];

        char author[20];

        int pages;
```

```
        float price;
};
void output(struct bookdetail v[],int n);
void main()
{
        struct bookdetail b[SIZE];
        int n,i;
        clrscr();
        printf("Enter the Numbers of Books:");
        scanf("%d",&n);
        printf("\n");
        for(i=0;i<n;i++)
        {
                printf("\t=:Book %d Detail:=\n",i+1);
                printf("\nEnter the Book Name:\n");
                scanf("%s",b[i].name);
                printf("Enter the Author of Book:\n");
                scanf("%s",b[i].author);
                printf("Enter the Pages of Book:\n");
                scanf("%d",&b[i].pages);
                printf("Enter the Price of Book:\n");
                scanf("%f",&b[i].price);
        }
        output(b,n);
```

```
        getch();
    }
    void output(struct bookdetail v[],int n)
    {
        int i,t=1;
        for(i=0;i<n;i++,t++)
        {
            printf("\n");
            printf("Book No.%d\n",t);
            printf("\t\tBook %d Name is=%s \n",t,v[i].name);
            printf("\t\tBook %d Author is=%s \n",t,v[i].author);
            printf("\t\tBook %d Pages is=%d \n",t,v[i].pages);
            printf("\t\tBook %d Price is=%f \n",t,v[i].price);
            printf("\n");
        }
    }
```

6. Write a C program to compute matrix multiplication.

```
#include <stdio.h>
 void main()
{
  int m, n, p, q, c, d, k, sum = 0;
  int first[10][10], second[10][10], multiply[10][10];
   printf("Enter number of rows and columns of first matrix\n");
  scanf("%d%d", &m, &n);
```

```
 printf("Enter elements of first matrix\n");
  for (c = 0; c < m; c++)
  for (d = 0; d < n; d++)
  scanf("%d", &first[c][d]);
 printf("Enter number of rows and columns of second matrix\n");
 scanf("%d%d", &p, &q);
 if (n != p)
  printf("The matrices can't be multiplied with each other.\n");
 else
 {
  printf("Enter elements of second matrix\n");
 for (c = 0; c < p; c++)
 for (d = 0; d < q; d++)
 scanf("%d", &second[c][d]);
for (c = 0; c < m; c++) {
for (d = 0; d < q; d++) {
for (k = 0; k < p; k++) {
sum = sum + first[c][k]*second[k][d];
}
multiply[c][d] = sum;
sum = 0;
}
}
 printf("Product of the matrices:\n");
```

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

Unit-3

PART-A(2 Marks):

1. **Define a file in C? What are the types of files?**

   A file is a place where a group of related data (records) can be stored. In general the entire data is lost when either program terminates or computer is turned off. Sometimes it may be necessary to store data permanently. This leads to introduce the concept of files to store the data permanently in the system. Files are stored in auxillary storage devices such as CD,DVD,Hard disks etc.,

   Types of Files
   Based on type of data being stored and way of accessing the content of the file, files are classified into two types:

   1. Text files
   2. Binary files

1. **Text files:** Text files are the normal .txt files that you can easily create using Notepad or any simple text editors. It consisting of collection of stream of characters that can be processed sequentially and in forward direction only.
2. **Binary files:** Binary files are mostly the .bin files in your computer. Instead of storing data in plain text, they store it in the binary form (0's and 1's).They can hold higher amount of data, are not readable easily and provides a better security than text files.

2. Distinguish between text file and binary file?

Text file	**Binary file**
Data is human readable characters. Each line ends with a newline character. EOF indicates end of the file. File reading is possible in forward direction only. They take minimum effort to maintain, are easily readable and provide least security and takes bigger storage space. Ex: C file	Data is in the form of sequence of bytes. There are no lines or newline characters. An feof() is a function to indicates end of the file. File reading is possible in any direction. They can hold higher amount of data, are not readable easily and provides a better security than text files. Ex: obj file

3. Write File processing steps?

To working with files we need to follow following steps:

a. Declare a file pointer
Ex: FILE *fp;

b. Opening an existing file using fopen() function.

Syntax: File_pointer=fopen("file_path","mode");

fp = fopen("fileName.c", "w");

c. Process the file using different file input, output(file handling)functions.
d. Closing a file using fclose() function.

4. What are the file handling functions in c?

We can read the data from file and write to the file using following file handling functions:

Function	description
fprintf()	writes a set of data to a file
fscanf()	reads a set of data from a file
fgetc()	reads a character from a file
fputc()	writes a character to a file
fgets()	reads a string from a file
fputs()	writes a string to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the beginning point

5. **What is rewind ( ) function? How it is different with fseek() function?**

   The rewind()sets the file position to the beginning of the file.

   Syntax: rewind (FILE *stream);

   fseek()is used for setting the file pointer at a specified positions.

   As the name suggests, fseek() seeks the cursor to the given record in the file.

   Syntax of fseek()

   fseek(FILE * stream, long int offset, int origin)

   The first parameter stream is the pointer to the file. The second parameter is the position of the record to be found, and the third parameter specifies the location where the offset starts.

6. **List out various file opening modes?**

mode	description
r	opens a text file in reading mode
w	opens or create a text file in writing mode.
a	opens a text file in append mode
r+	opens a text file in both reading and writing mode
w+	opens a text file in both reading and writing mode
a+	opens a text file in both reading and writing mode

Part-B (5 Marks)

1a. Give a brief explanation about file opening modes.

To open a file we need to use fopen() function. In that function we should specify file name and mode of the file. The modes can be used to know the purpose of opening a file.There are several file modes:

Fp=fopen("file name","mode");

Mode	Description
r	Opens an existing text file for reading purpose.
w	Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file.
a	Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here your program will start appending content in the existing file content.
r+	Opens a text file for both reading and writing.
w+	Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.
a+	Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

If you are going to handle binary files, then you will use following access modes instead of the above mentioned ones:

"rb", "wb", "ab", "rb+", "wb+", "ab+"

1b. Write about

a) #define b)#undef c)#ifdef d)#ifndef

Preprocessor directive	Description
#define	It defines a symbolic constant or a macro
#undef	It allows you to undefined a symbol
#ifdef	It executes statements in sequence if the macro name is defined
#ifndef	It executes statements in sequence if the macro name is not defined

#define: A macro is a name given to a block of c statements as a pre-processor directive. A macro is defined with the preprocessor directive #define. Whenever the name is used, it is replaced by the content of the macro.

#undef: This directive undefines a macro. A macro must be undefined when we want to redefine a macro or symbolic constant with different value.

#undef PI

#ifdef and #ifndef: These directives are useful for checking whether the macros are defined or not

Syntax: #ifdef macro_name
<statement sequence>
#endif
Syntax: #ifndef macro_name
<statement sequence>
#endif

2a. Explain briefly about file positioning functions.

There is no need to read each record sequentially, if we want to access a particular record.C supports these functions for random access file processing.

fseek()
ftell()
rewind()

fseek():

This function is used for seeking the pointer position in the file at the specified byte.

Syntax: fseek(file pointer, displacement, pointer position);

Where file pointer ---- It is the pointer which points to the file.

displacement ---- It is positive or negative. This is the number of bytes which are skipped backward (if negative) or forward(if positive) from the current position.

Pointer position:

This sets the pointer position in the file.

Value **pointer position**

0	Beginning of file.
1	Current position
2	End of file

Ex: fseek(p,10L,0)
0 means pointer position is on beginning of the file, from this statement pointer position is skipped 10 bytes from the beginning of the file.

Ex: fseek(p,5l,1)
1 means current position of the pointer position. From this statement pointer position is skipped 5 bytes forward from the current position.

3)fseek(p,-5,1)

From this statement pointer position is skipped 5 bytes backward from the current position.

ftell():

This function returns the value of the current pointer position in the file.The value is count from the beginning of the file.

Syntax: ftell(fptr);

Where fptr is a file pointer.

rewind():

This function is used to move the file pointer to the beginning of the given file.

Syntax: rewind(fptr);

Where fptr is a file pointer.

Example program for fseek():

Write a program to read last ‘n’ characters of the file using appropriate file functions (Here we need fseek() and fgetc()).

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
```

```
        char ch;
        clrscr();
        fp=fopen("file1.c", "r");
        if(fp==NULL)
          printf("file cannot be opened");
        else
        {
            printf("Enter value of n  to read last 'n' characters");
            scanf("%d",&n);
            fseek(fp,-n,2);
            while((ch=fgetc(fp))!=EOF)
            {
                printf("%c\t",ch);
            }
         }
        fclose(fp);
        getch();
    }
```

OUTPUT: It depends on the content in the file.

2b. Explain fseek() function with example.

declaration: int fseek(FILE *fp, long int offset, int whence)

fseek() function is used to move file pointer position to the given location.

where,

fp – file pointer

offset – Number of bytes/characters to be offset/moved from whence/the current file pointer position

whence – This is the current file pointer position from where offset is added. Below 3 constants are used to specify this.

SEEK_SET	SEEK_SET – It moves file pointer position to the beginning of the file.
SEEK_CUR	SEEK_CUR – It moves file pointer position to given location.
SEEK_END	SEEK_END – It moves file pointer position to the end of file.

EX:

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

```
{
  FILE *fp;
  char data[60];
  fp = fopen ("test.c","w");
  fputs("Fresh2refresh.com is a programming tutorial website", fp);
  fgets ( data, 60, fp );
  printf(Before fseek - %s", data);

  // To set file pointet to 21th byte/character in the file
  fseek(fp, 21, SEEK_SET);
  fflush(data);
  fgets ( data, 60, fp );
  printf("After SEEK_SET to 21 - %s", data);

  // To find backward 10 bytes from current position
  fseek(fp, -10, SEEK_CUR);
  fflush(data);
  fgets ( data, 60, fp );
  printf("After SEEK_CUR to -10 - %s", data);

  // To find 7th byte before the end of file
  fseek(fp, -7, SEEK_END);
  fflush(data);
  fgets ( data, 60, fp );
  printf("After SEEK_END to -7 - %s", data);

  // To set file pointer to the beginning of the file
  fseek(fp, 0, SEEK_SET); // We can use rewind(fp); also

  fclose(fp);
  return 0;
}
```

3a. Write a C program to merge content of two files into third file.

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

Void main()
{
  // Open two files to be merged
  FILE *fp1 = fopen("file1.txt", "r");
  FILE *fp2 = fopen("file2.txt", "r");

  // Open file to store the result
  FILE *fp3 = fopen("file3.txt", "w");
  char c;
```

```
  if (fp1 == NULL || fp2 == NULL || fp3 == NULL)
  {
      puts("Could not open files");
      exit(0);
  }

  // Copy contents of first file to file3.txt
  while ((c = fgetc(fp1)) != EOF)
    fputc(c, fp3);

  // Copy contents of second file to file3.txt
  while ((c = fgetc(fp2)) != EOF)
    fputc(c, fp3);

  printf("Merged file1.txt and file2.txt into file3.txt");

  fclose(fp1);
  fclose(fp2);
  fclose(fp3);
  return 0;
}
```

3b. Describe the file status functions?

feof():

The macro feof() is used for detecting whether the file pointer is at the end of file or not.It returns nonzero if the file pointer is at the end of the file otherwise it returns zero.
Syntax: feof(fptr);
Where fptr is a file pointer .

ferror()

The macro ferror() is used for detecting whether an error occur in the file on filepointer or not.It returns the value nonzero if an error,otherwise it returns zero.
Syntax: ferror(fptr);

perror()

perror() function displays the string you pass to it, followed by a colon, a space, and then the textual representation of the current errno value.e fptr is a file pointer.

Syntax: void perror(const char*str);

#include <stdio.h>

```
int main () {
  FILE *fp;
  int c;

  fp = fopen("file.txt","r");
  if(fp == NULL) {
    perror("Error in opening file");
    return(-1);
  }

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

  return(0);
}
```

4a. Which kind of errors can we expect while working with files? How to handle them?

While operating on files, there may be a chance of having certain errors which may cause abnormal behavior in our programs.

a. Opening a file that was not present in the system
b. Trying to read beyond the end of the mark
c. Device overflow
d. Trying to use a file that has not been opened.
e. Trying to perform an operation on a file when the file is opened for another type of operation.

We can handle these errors using error handling functions: They are

1. feof()
2. ferror()
3. perror()
4. clearerr()

1. feof():

Syntax: int feof(FILE *stream);

The feof() function indicates whether the end-of-file flag is set for the given *stream*. The end-of-file flag is set by several functions to indicate the end of the file. The end-of-file flag is cleared by calling the rewind(), fsetpos(), fseek(), or clearerr() functions for this stream.

The feof() function returns a nonzero value if and only if the EOF flag is set; otherwise, it returns 0.

2. ferror()

Syntax: int ferror(FILE *stream);

The ferror() function tests for an error in reading from or writing to the given *stream*. If an error occurs, the error indicator for the *stream* remains set until you close *stream*, call the rewind() function, or call the clearerr() function.

The ferror() function returns a nonzero value to indicate an error on the given *stream*. A return value of 0 means that no error has occurred.

3. perror()

Syntax: void perror(const char *string);n

The perror() function prints an error message to stderr. If *string* is not NULL and does not point to a null character, the string pointed to by *string* is printed to the standard error stream, followed by a colon and a space. The message associated with the value in errno is then printed followed by a new-line character.

To produce accurate results, you should ensure that the perror() function is called immediately after a library function returns with an error; otherwise, subsequent calls might alter the errno value.

4. clearerr()

Syntax: void clearerr (FILE *stream);

The clearerr() function resets the error indicator and end-of-file indicator for the specified *stream*. Once set, the indicators for a specified stream remain set until your program calls the clearerr() function or the rewind() function. The fseek() function also clears the end-of-file indicator.

There is no return value.

4b. Write a C program to copy content of one file to another file?

```
#include<stdio.h>
void  main()
{
    int ch;
    FILE *fp,*fq;
    fp=fopen("source.txt","r");
    fq=fopen("backup.txt","w");
    if(fp==NULL||fq==NULL)
        printf("File does not exist..");
    else
        while((ch=fgetc(fp))!=EOF)
        {
            fputc(ch,fq);
        }
    printf("File copied.....");
    return 0;
}
```

5a. Write a C program to reverse all characters from a file and display on to the monitor?

```
#include<stdio.h>
int main()
{
    FILE *fp;
    char ch;
    int i,pos;
    fp=fopen("input.txt","r");
    if(fp==NULL)
    {
        printf("File does not exist..");
    }
    fseek(fp,0,SEEK_END);
    pos=ftell(fp);
```

```
    //printf("Current postion is %d\n",pos);
    i=0;
    while(i<pos)
     {
        i++;
        fseek(fp,-i,SEEK_END);
        //printf("%c",fgetc(fp));
        ch=fgetc(fp);
        printf("%c",ch);
    }
    return 0;
}
```

5b. Write a C program for writing and reading structures using binary files.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// a struct to read and write
struct person
{
   int id;
   char fname[20];
   char lname[20];
};

int main ()
{
   FILE *outfile;

   // open file for writing
   outfile = fopen ("person.dat", "w");
   if (outfile == NULL)
   {
      fprintf(stderr, "\nError opend file\n");
      exit (1);
   }

   struct person input1 = {1, "rohan", "sharma"};
   struct person input2 = {2, "mahendra", "dhoni"};
```

```
   // write struct to file
   fwrite (&input1, sizeof(struct person), 1, outfile);
   fwrite (&input2, sizeof(struct person), 1, outfile);

   if(fwrite != 0)
      printf("contents to file written successfully !\n");
   else
      printf("error writing file !\n");

   // close file
   fclose (outfile);

   return 0;
}
```

06 Write a C program to count the number of times a character occurs in a text file. The file name and the characters are supplied as command line arguments.

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

int main(int argc,char *argv[])
{
Char ch,c;
FILE *fp;
Clrscr();
C=argv[2][0];
Fp=fopen("argv[1]"."r");
If(fp==NULL)
{
Printf("File opening error");
Getch();
Exit(0);
}
Printf("The contents of the file is:\n");
While((ch=fgetc(fp))!=EOF)
Printf(%c",ch);

Fclose(fp);
}
```

Unit-4

PART-A(2 Marks):

1. Define a function? Are functions required when writing a C program?

A function is a group of statements that together perform a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions. We can provide the modularity for the program by divide up our c program into separate functions known as sub modules or sub tasks. Such that each module or function performs a specific task.

2. What is function prototype? Give an example.

A function prototype is also known as function declaration, a function declaration tells the compiler about a function's name, return type, and parameters.

A function declaration has the following parts −

return_type function_name(parameter list);

For the above defined function max(), the function declaration is as follows −

int max(int num1, int num2);

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration −

int max(int, int);

3. Define recursion? What are the limitations of recursion?

When a function calls itself from its body is called Recursion.

Limitations of recursion:

1. Recursive solution is always logical and it is very difficult to trace.(debug and understand).
2. In recursive we must have an if statement somewhere to force the function to return without the recursive call being executed, otherwise the function will never return.
3. Recursion takes a lot of stack space, usually not considerable when the program is small and running on a PC.
4. Recursion uses more processor time.

4. Distinguish between malloc() and calloc() functions in C?

Differences between malloc and calloc

Malloc	Calloc
The name mallocstands for *memory*	The name callocstands for *contiguous*

allocation.	*allocation*.
void *malloc(size_t n) returns a pointer to n bytes of uninitialized storage, or NULL if the request cannot be satisfied. If the space assigned by malloc() is overrun, the results are undefined.	void *calloc(size_t n, size_t size)returns a pointer to enough free space for an array of n objects of the specified size, or NULL if the request cannot be satisfied. The storage is initialized to zero.
malloc() takes one argument that is, *number of bytes*.	calloc() take two arguments those are: *number of blocks*and *size of each block*.
syntax of malloc(): void *malloc(size_t n); Allocates n bytes of memory. If the allocation succeeds, a void pointer to the allocated memory is returned. Otherwise NULL is returned.	syntax of calloc(): void *calloc(size_t n, size_t size); Allocates a contiguous block of memory large enough to hold nelements of sizebytes each. The allocated region is initialized to zero.
malloc is faster than calloc.	calloc takes little longer than mallocbecause of the extra step of initializing the allocated memory by zero. However, in practice the difference in speed is very tiny and not recognizable.

5. Differentiate actual parameters and formal parameters in functions?

Actual Parameters

The parameters used in the function call are called actual parameters. These are the actual values that are passed to the function. The actual parameters may be in the form of constant values or variables. The data types of actual parameters must match with the corresponding data types of formal parameters (variables) in the function definition.

The parameters used in the header of function definition are called formal parameters of the function. These parameters are used to receive values from the calling function.

6. Distinguish between Library functions and User defined functions in 'C' with relevant examples

User Defined Functions	Library Function
UDF(user defined functions) are the function which are created by user as per his own	LF(library functions) are Predefined functions.

requirements.	
UDF are part of the program which compile runtime	LF are part of header file (such as math.h) which is called runtime.
In UDF the name of function id decided by user	in LF it is given by developers.
in UDF name of function can be changed any time	LF Name of function can't be changed.
Example : add(), average(), fibo()	Example : printf(), main()

PART-B (5 Marks)

1a. Explain passing parameters to functions.

Two Ways of Passing Parameters to Function in C Language :

A. Call by Value:

The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

By default, C programming uses call by value to pass arguments.

```
#include<stdio.h>
void interchange(int number1,int number2)
{
int temp;
temp = number1;
number1 = number2;
number2 = temp;
}
int main() {
int num1=50,num2=70;
interchange(num1,num2);
printf("\nNumber 1 : %d",num1);
printf("\nNumber 2 : %d",num2);
return(0);
```

```
}
```

Output :

Number 1 : 50

Number 2 : 70

B.Call by Reference:

The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. It means the changes made to the parameter affect the passed argument.

To pass a value by reference, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following function interchange(), which exchanges the values of the two integer variables pointed to, by their arguments.

```
#include<stdio.h>
void interchange(int *num1,int *num2)
{
int temp;
temp  = *num1;
*num1 = *num2;
*num2 = temp;
}
int main() {
int num1=50,num2=70;
interchange(&num1,&num2);
printf("\nNumber 1 : %d",num1);
printf("\nNumber 2 : %d",num2);
return(0);
}
```

1b. How can an array be passed to a function? Describe the procedure?

In C programming, a single array element or an entire array can be passed to a function. This can be done for both one-dimensional array or a multi-dimensional array.

Passing One-dimensional Array In Function

Single element of an array can be passed in similar manner as passing variable to a function.

C program to pass a single element of an array to function

```
#include <stdio.h>
void display(int age)
{
printf("%d", age);
}
int main()
{
int ageArray[] = { 2, 3, 4 };
display(ageArray[2]); //Passing array element ageArray[2] only.
return 0;
}
```

Passing an entire one-dimensional array to a function

While passing arrays as arguments to the function, only the name of the array is passed (,i.e, starting address of memory area is passed as argument).

C program to pass an array containing age of person to a function. This function should find average age and display the average age in main function.

```
#include <stdio.h>
float average(float age[]);
int main()
{
float avg, age[] = { 23.4, 55, 22.6, 3, 40.5, 18};
avg = average(age); // Only name of an array is passed as an argument
printf("Average age = %.2f", avg);
return 0;
}
float average(float age[])
{
int i;
float avg, sum = 0.0;
```

```
for (i = 0; i < 6; ++i) {
sum += age[i];
}
avg = (sum / 6);
return avg;
}
```

2a. Why is it sometimes desirable to pass a pointer to a function as an argument?

When we pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value. So any change made by the function using the pointer is permanently made at the address of passed variable. This technique is known as call by reference in C.

Try this same program without pointer, you would find that the bonus amount will not reflect in the salary, this is because the change made by the function would be done to the local variables of the function. When we use pointers, the value is changed at the address of variable

```
#include <stdio.h>
void salaryhike(int  *var, int b)
{
*var = *var+b;
}
int main()
{
int salary=0, bonus=0;
printf("Enter the employee current salary:");
scanf("%d", &salary);
printf("Enter bonus:");
scanf("%d", &bonus);
salaryhike(&salary, bonus);
printf("Final salary: %d", salary);
return 0;
}
```

Output:

```
Enter the employee current salary:10000
Enter bonus:2000
```

Final salary: 12000

Example 2: Swapping two numbers using Pointers

This is one of the most popular example that shows how to swap numbers using call by reference.

Try this program without pointers, you would see that the numbers are not swapped. The reason is same that we have seen above in the first example.

```
#include <stdio.h>
void swapnum(int *num1, int *num2)
{
int tempnum;

tempnum = *num1;
*num1 = *num2;
*num2 = tempnum;
}
int main( )
{
int v1 = 11, v2 = 77 ;
printf("Before swapping:");
printf("\nValue of v1 is: %d", v1);
printf("\nValue of v2 is: %d", v2);
/*calling swap function*/
swapnum( &v1, &v2 );
printf("\nAfter swapping:");
printf("\nValue of v1 is: %d", v1);
printf("\nValue of v2 is: %d", v2);
}
```

Output:

Before swapping:

Value of v1 is: 11

Value of v2 is: 77

After swapping:

Value of v1 is: 77

Value of v2 is: 11

2b. Explain the various categories of user defined functions in 'C' with examples

There can be 4 different types of user-defined functions, they are:

1. Function with no arguments and no return value
2. Function with no arguments and a return value
3. Function with arguments and no return value
4. Function with arguments and a return value

Function with no arguments and no return value

Such functions can either be used to display information or they are completely dependent on user inputs.

Below is an example of a function, which takes 2 numbers as input from user, and display which is the greater number.

```
#include<stdio.h>
void greatNum();       // function declaration
int main()
{
greatNum();        // function call
return 0;
}
void greatNum()        // function definition
{
int i, j;
printf("Enter 2 numbers that you want to compare...");
scanf("%d%d", &i, &j);
if(i > j)
{
printf("The greater number is: %d", i);
}
else
```

```
{
printf("The greater number is: %d", j);
}
}
```

Function with no arguments and a return value

We have modified the above example to make the function greatNum() return the number which is greater amongst the 2 input numbers.

```
#include<stdio.h>
int greatNum();       // function declaration
int main()
{
int result;
result = greatNum();        // function call
printf("The greater number is: %d", result);
return 0;
}
int greatNum()        // function definition
{
int i, j, greaterNum;
printf("Enter 2 numbers that you want to compare...");
scanf("%d%d", &i, &j);
if(i > j) {
greaterNum = i;
}
else {
greaterNum = j;
}
// returning the result
return greaterNum;
}
```

Function with arguments and no return value

We are using the same function as example again and again, to demonstrate that to solve a problem there can be many different ways.

This time, we have modified the above example to make the function greatNum() take two int values as arguments, but it will not be returning anything.

```
#include<stdio.h>
void greatNum(int a, int b);       // function declaration
int main()
{
int i, j;
printf("Enter 2 numbers that you want to compare...");
scanf("%d%d", &i, &j);
greatNum(i, j);        // function call
return 0;
}
void greatNum(int x, int y)        // function definition
{
if(x > y) {
printf("The greater number is: %d", x);
}
else {
printf("The greater number is: %d", y);
}
}
```

Function with arguments and a return value

This is the best type, as this makes the function completely independent of inputs and outputs, and only the logic is defined inside the function body.

```
#include<stdio.h>
int greatNum(int a, int b);       // function declaration
int main()
{
int i, j, result;
```

```
printf("Enter 2 numbers that you want to compare...");
scanf("%d%d", &i, &j);
result = greatNum(i, j); // function call
printf("The greater number is: %d", result);
return 0;
}

int greatNum(int x, int y)        // function definition
{
if(x > y) {
return x;
}
else {
return y;
}
}
```

3a. What is meant by dynamic memory allocation? What are the library functions used to allocate memory dynamically?

Dynamic Memory Allocation

Dynamic Memory Allocation refers to managing system memory at runtime. Dynamic memory management in C programming language is performed via a group four functions namely malloc(), calloc(), realloc(), and free(). These four dynamic memory allocation functions of the C programming language are defined in the C standard library header file <stdlib.h>. Dynamic memory allocation uses the heap space of the system memory. Let's examine the four dynamic memory management functions in more detail.

malloc()

malloc() is used to allocate a block of memory on the heap. Specifically, malloc() allocates the user a specified number of bytes but does not initialize. Once allocated, the program accesses this block of memory via a pointer that malloc() returns. The default pointer returned by malloc() is of the type void but can be cast into a pointer of any data type. However, if the

space is insufficient for the amount of memory requested by malloc(), then the allocation fails and a NULL pointer is returned. The function takes a single argument, which is the size of the memory chunk to be allocated. You can dynamically allocate an integer buffer using malloc() as follows:

int *buffer = (int *) malloc(SIZE_USER_NEEDS * sizeof(int));

In this statement, sizeof(int) will return 2 or 4 bytes as the default size of an integer type. Assuming a user wants a buffer to hold N integer objects, the above statement will dynamically allocate N*4 bytes on the heap. For clarity, if user input SIZE_USER_NEEDS=5, then 5*4=20 bytes will be allocated on the heap that can be accessed using the pointer *buffer.

calloc()

calloc() allocates a user-specified number of bytes and initializes them to zero. Unlike malloc(), this function takes two arguments - the number of memory chunks to be allocated and the size of each memory chunk.

int *buffer = (int *) calloc(NUMBER_OF_ELEMENTS, sizeof(int));

realloc()

realloc() is used when an allocated block on dynamic memory needs to be resized. If necessary, realloc() completely reallocates the memory block. Assume you make an initial memory allocation using malloc() as follows:

int *buffer = (int *) malloc(size);

3b. Write a C program to find Fibonacci series using recursive and non- recursive functions.

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

void fib(int n)
{
int a = 0, b = 1, c, count = 3;
if(n == 1)
printf("0");
```

```
else if(n == 2)
printf("0 1");
else
{
printf("0 1 ");
while(count <= n)
{
c = a + b;
printf("%d ", c);
a = b;
b = c;
count++;
}
}
}

int rfib(int n)
{
if(n == 1)
return 0;
else if(n == 2)
return 1;
else
{
return rfib(n - 1) + rfib(n - 2);
}
}

int main(int argc, char **argv)
{
int n, count = 3;
printf("Enter a number: ");
```

```
scanf("%d", &n);
printf("\nNon-recursive fibonacci sequence upto %d terms: \n", n);
fib(n);
printf("\nRecursive fibonacci sequence upto %d terms: \n", n);
if(n == 1)
printf("0");
else if(n == 2)
printf("0 1");
else
{
printf("0 1 ");
while(count <= n)
{
printf("%d ", rfib(count));
count++;
}
}
getch();
return 0;
}
```

Input and output of the above program

```
1 Enter a number: 10
2
3 Non-recursive fibonacci sequence upto 10 terms:
4 0 1 1 2 3 5 8 13 21 34
5 Recursive fibonacci sequence upto 10 terms:
6 0 1 1 2 3 5 8 13 21 34
```

.

4a. Write a C program two find GCD of two given integers using both recursive and non-recursive functions.

GCD without using recursion

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

```
int main(){
int x,y,m,i;
printf("Insert any two number: ");
scanf("%d%d",&x,&y);
if(x>y)
m=y;
else
m=x;
for(i=m;i>=1;i--){
if(x%i==0&&y%i==0){
printf("\nHCF of two number is : %d",i) ;
break;
}
}
return 0;
}
#include<stdio.h>
int main(){
int n1,n2;
printf("\nEnter two numbers:");
scanf("%d %d",&n1,&n2);
while(n1!=n2){
if(n1>=n2-1)
n1=n1-n2;
else
n2=n2-n1;
}
printf("\nGCD=%d",n1);
return 0;
}
```

GCD using Recursion

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

```
int main(){
int n1,n2,gcd;
printf("\nEnter two numbers: ");
scanf("%d %d",&n1,&n2);
gcd=findgcd(n1,n2);
printf("\nGCD of %d and %d is: %d",n1,n2,gcd);
return 0;
}

int findgcd(int x,int y){
while(x!=y){
if(x>y)
return findgcd(x-y,y);
else
return findgcd(x,y-x);
}
return x;
}
```

4b. How do you declare and define user defined function? Explain with a suitable example?

A function is a block of code that performs a specific task.

Function prototype

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.

A function prototype gives information to the compiler that the function may later be used in the program.

Syntax of function prototype

returnType functionName(type1 argument1, type2 argument2,...);

In the above example, int addNumbers(int a, int b); is the function prototype which provides following information to the compiler:

name of the function is addNumbers()

return type of the function is int

two arguments of type int are passed to the function

The function prototype is not needed if the user-defined function is defined before the main() function.

Calling a function

Control of the program is transferred to the user-defined function by calling it.

Syntax of function call

functionName(argument1, argument2, ...);

In the above example, function call is made using addNumbers(n1,n2); statement inside the main().

Function definition

Function definition contains the block of code to perform a specific task i.e. in this case, adding two numbers and returning it.

Syntax of function definition

```
returnType functionName(type1 argument1, type2 argument2, ...)
{
//body of the function
}
```

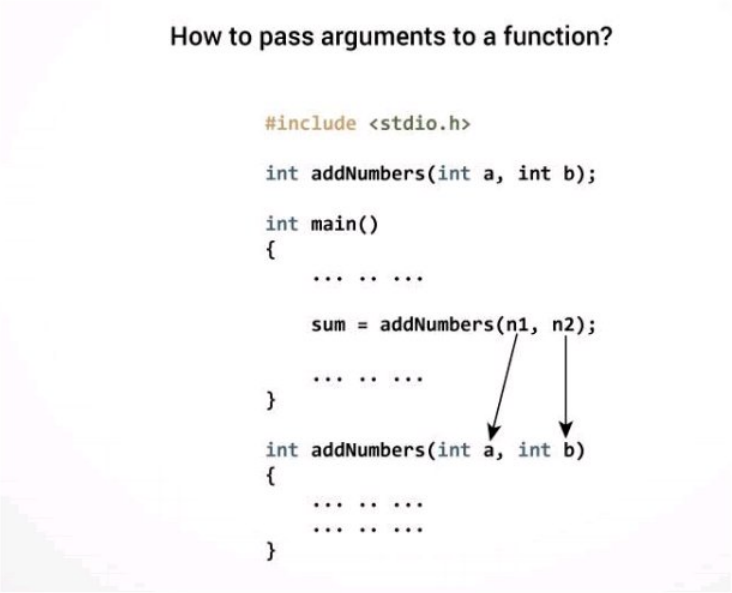
When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function.

Passing arguments to a function

In programming, argument refers to the variable passed to the function. In the above example, two variables n1 and n2 are passed during function call.

The parameters a and b accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.

How to pass arguments to a function?



The type of arguments passed to a function and the formal parameters must match, otherwise the compiler throws error.

If n1 is of char type, a also should be of char type. If n2 is of float type, variable b also should be of float type.

A function can also be called without passing an argument.

Return Statement

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after return statement.

In the above example, the value of variable result is returned to the variable sum in the main() function.

Syntax of return statement

return (expression);

For example,

return a;

return (a+b);

The type of value returned from the function and the return type specified in function prototype and function definition must match.

5a. Discuss how to allocate the memory for arrays of different data types?

We can allocate memory for array of elements in two ways: compile time memory allocation, run-time memory allocation.

Compile-time memory allocation for arrays:

We can follow following syntax to allocate the array of elements at compile time-

Datatype arrayName [SIZE];

Example, ina a[10];

float height[15] etc.

Run-time memory allocation for arrays:

We can allocate memory during run-time by using malloc() and calloc() funations. Consider the following example c program to understand how to allocate the memory for arrays dynamically.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int num, i, *ptr, sum = 0;
   printf("Enter number of elements: ");
   scanf("%d", &num);
   ptr = (int*) calloc(num, sizeof(int));
   if(ptr == NULL)
   {
      printf("Error! memory not allocated.");
      exit(0);
   }
   printf("Enter elements of array: ");
   for(i = 0; i < num; ++i)
   {
      scanf("%d", ptr + i);
      sum += *(ptr + i);
   }
  printf("Sum = %d", sum);
   free(ptr);
   return 0;
```

}

5b. Describe the idea of call by reference?

The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. It means the changes made to the parameter affect the passed argument.

To pass a value by reference, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following function swap(), which exchanges the values of the two integer variables pointed to, by their arguments.

```
/* function definition to swap the values */
void swap(int *x, int *y) {
int temp;
temp = *x;    /* save the value at address x */
*x = *y;      /* put y into x */
*y = temp;    /* put temp into y */
return;
}
```

Let us now call the function swap() by passing values by reference as in the following example −

```
#include <stdio.h>
/* function declaration */
void swap(int *x, int *y);
int main () {
/* local variable definition */
int a = 100;
int b = 200;
printf("Before swap, value of a : %d\n", a );
printf("Before swap, value of b : %d\n", b );
/* calling a function to swap the values.
* &a indicates pointer to a ie. address of variable a and
* &b indicates pointer to b ie. address of variable b.
*/
```

```
swap(&a, &b);
printf("After swap, value of a : %d\n", a );
printf("After swap, value of b : %d\n", b );
return 0;
}
```

Let us put the above code in a single C file, compile and execute it, to produce the following result −

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :200
After swap, value of b :100
```

It shows that the change has reflected outside the function as well, unlike call by value where the changes do not reflect outside the function.

06 Write C programs that use both recursive and non-recursive functions to find the factorial of a given integer.

Factorial of a number is nothing but the multiplication of numbers from a given number to 1 Ex: 5! =5*4*3*2*1= 120

```
#include <stdio.h>
#include <conio.h>
void main()
{
int n, a, b;
clrscr();
printf("Enter any number\n");
scanf("%d", &n);
a = recfactorial(n);
printf("The factorial of a given number using recursion is %d \n", a);
b = nonrecfactorial(n);
printf("The factorial of a given number using nonrecursion is %d ", b);
getch();
}
```

```
int recfactorial(int x)
{
int f;
if(x == 0)
{
return(1);
}
else
{
f = x * recfactorial(x - 1);
return(f);
}
}
int nonrecfactorial(int x)
{
int i, f = 1;
for(i = 1;i <= x; i++)
{
f = f * i;
}
return(f);
}
```

INPUT & OUTPUT:

Enter any number

5

The factorial of a given number using recursion is 120

The factorial of a given number using nonrecursion is 120

Unit-5

PART-A(2 Marks):

1. Why binary search is preferred over linear search.

Because a linear search scans only one element at a time.

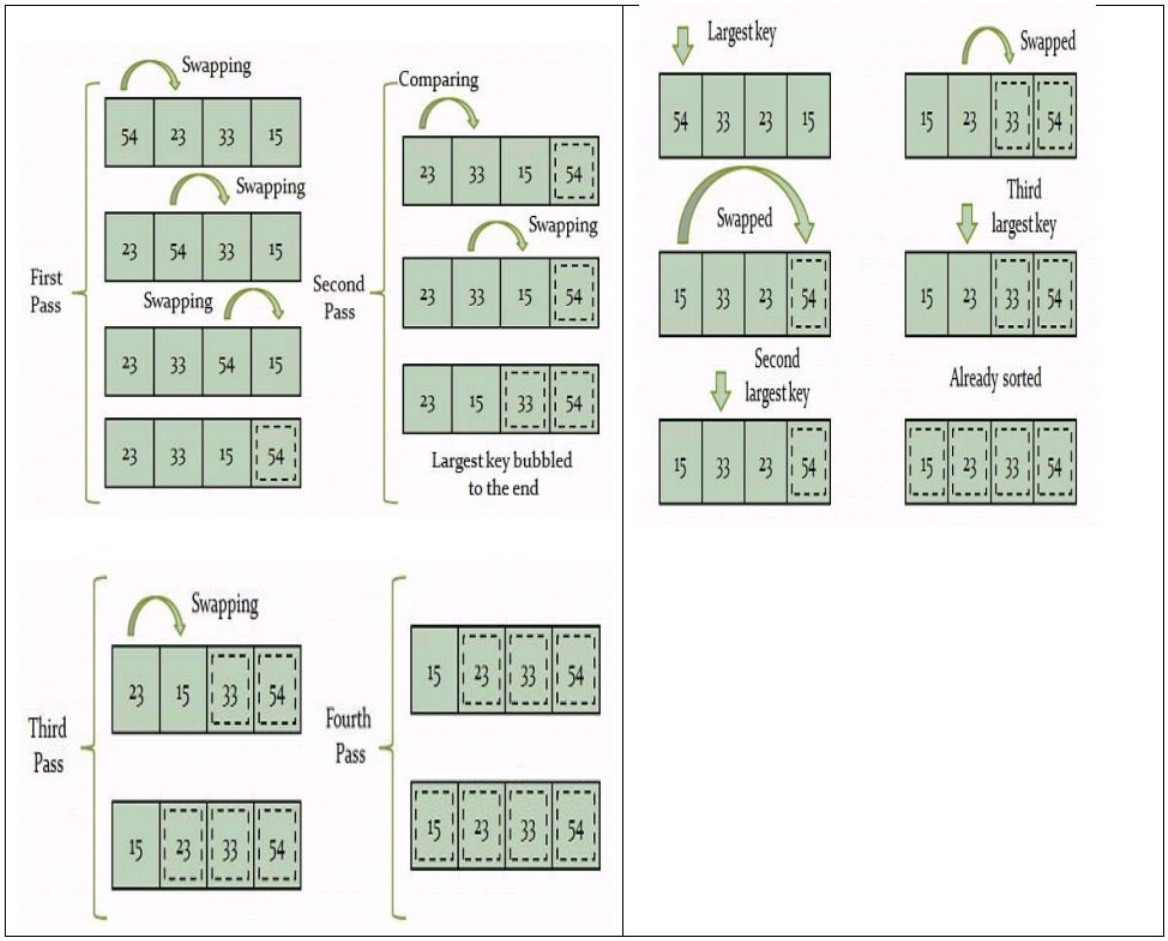
- Time taken to search elements keeps increasing as the number of elements are increased.

But a binary search however, cut down the search to half as soon as you find middle of a sorted list.

- The middle element is looked to check if it is greater than or less than the value to be searched.
- Accordingly, search is done to either half of the given list
- Linear search does the sequential access whereas Binary search access data randomly.
- Time complexity of linear search is O(n) , but Binary search time complexity is O(log n).
- Linear search performs equality comparisons and Binary search performs ordering comparisons

2. Compare and contrast any two sorting techniques?

Bubble Sort	**Selection Sort**
Bubble sort is the simplest iterative algorithm where each element and its adjacent element is compared and swapped if required.	Largest element is selected and swapped with the last element (in case of ascending order) Or Smallest element is selected and swapped with the first element (in case of descending order)
Bubble sort performs the sorting by exchanging the elements	Selection sort performs the sorting by selecting the element
Bubble sort uses exchanging of elements method	Selection sort uses selection of elements method
Bubble sort is a stable and inefficient algorithm	Selection sort is a unstable and efficient algorithm
Bubble sort algorithm is very slow and inefficient when compared with selection sort	Bubble sort algorithm is very fast and efficient when compared with bubble sort
The best case time complexity of the Bubble sort is **O(n)**, and worst case complexity is **$O(n^2)$**	The best case time complexity of the Bubble sort is **$O(n^2)$**, and worst case complexity is **$O(n^2)$**
Example: Sorting the elements (54, 23, 33, 15) using Bubble sort Technique is shown below.	Example: Sorting the elements (54, 23, 33, 15) in ascending order using Selection sort Technique is shown below.



3. Define time and space complexities?

Time complexity:

Time complexity signifies the total amount of *time* required by the algorithm or program to run till its completion. The *time complexity* of algorithms is most commonly expressed by using the big O notation. It's an asymptotic notation to represent the *time complexity*.

In other words, time complexity is to measure, how long a program function takes to process a given input.

Space complexity:

Space complexity signifies the total *amount of memory space* required by the algorithm or program to run till its completion.

or

Total amount of computer memory required by an algorithm to complete its execution is called as space complexity of a algorithm or program.

Generally, when a program is under execution it uses the computer memory for THREE reasons. They are as follows...

1. **Instruction Space:** It is the amount of memory used to store compiled version of instructions.
2. **Environmental Stack:** It is the amount of memory used to store information of partially executed functions at the time of function call.
3. **Data Space:** It is the amount of memory used to store all the variables and constants.

4. Write an algorithm for insertion sort?

Insertion sort algorithm arranges a list of elements in a particular order. In insertion sort algorithm, every iteration moves an element from unsorted portion to sorted portion until all the elements are sorted in the list.

The insertion sort algorithm is performed using following steps...

- **Step 1:** Assume that first element in the list is in sorted portion of the list and remaining all elements are in unsorted portion.
- **Step 2:** Consider first element from the unsorted list and insert that element into the sorted list in order specified.
- **Step 3:** Repeat the above process until all the elements from the unsorted list are moved into the sorted list.

5. Define sorting and write an algorithm for selection sort?

Sorting:

Sorting is the process of arranging a list of elements in a particular order (Ascending or Descending).

Selection sort:

Selection Sort algorithm is used to arrange a list of elements in a particular order (Ascending or Descending). The selection sort algorithm is performed using following steps...

- In selection sort, the first element in the list is selected and it is compared repeatedly with remaining all the elements in the list.
- If any element is smaller than the selected element (for ascending order), then both are swapped.
- Then we select the element at second position in the list and it is compared with remaining all elements in the list.
- If any element is smaller than the selected element, then both are swapped. This procedure is repeated till the entire list is sorted.

6. State the time complexities of searching techniques?

A linear search scans one item at a time, without jumping to any item . linear search or sequential search is a method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.

A binary search however, cut down your search to half as soon as you find middle of a sorted list.

- The middle element is looked to check if it is greater than or less than the value to be searched.
- Accordingly, search is done to either half of the given list

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)

Part –B: **5 Marks Questions**

1a. Write an algorithm to find roots of a quadratic equation.

ALGORITHM:

1) Start
2) Declare the required variables
3) Read the coefficients a, b, c of quadratic equation
4) Calculate d=b2-4ac
5) if d<0
    5.1 display " roots are imaginary".
6) Otherwise check the condition (d==0)
    6.1 calculate x= y= (-b/(2*a))
    6.2 Display real roots x,y
7) Otherwise
    7.1 calculate x=(-b+ sqrt(d))/2a
    7.2 y=(-b-sqrt(d))/2a
    7.3 Display real roots x,y
8) Stop.

1 b. Explain Linear Search algorithm with an example

Linear search algorithm finds given element in a list of elements with **O(n)** time complexity where **n** is total number of elements in the list.

Linear search is implemented using following steps...

- **Step 1:** Read the search element from the user
- **Step 2:** Compare, the search element with the first element in the list.
- **Step 3:** If both are matching, then display "Given element found!!!" and terminate the function
- **Step 4:** If both are not matching, then compare search element with the next element in the list.
- **Step 5:** Repeat steps 3 and 4 until the search element is compared with the last element in the list.
- **Step 6:** If the last element in the list is also doesn't match, then display "Element not found!!!" and terminate the function.

Example

Consider the following list of element and search element...

search element **12**

Step 1:

search element (12) is compared with first element (65)

Both are not matching. So move to next element

Step 2:

search element (12) is compared with next element (20)

Both are not matching. So move to next element

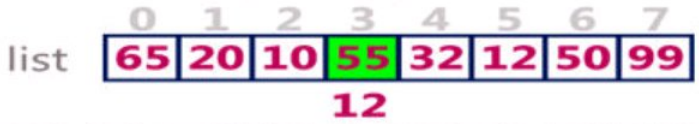
Step 3:

search element (12) is compared with next element (10)

Both are not matching. So move to next element

Step 4:

search element (12) is compared with next element (55)



Both are not matching. So move to next element

Step 5:

search element (12) is compared with next element (32)

Both are not matching. So move to next element

Step 6:

search element (12) is compared with next element (12)

Both are matching. So we stop comparing and display element found at index 5.

2a. Write a C program to implement insertion sort.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,j,item,n,a[30];
        clrscr();
        printf("Enter number of elements:");
        scanf("%d",&n);
        printf("enter %d elements:\n",n);
        for(i=0;i<=n-1;i++)
                scanf("%d",&a[i]);
        printf("Sorted elements in ascending order:\n");
        for(i=1;i<=n-1;i++)
        {
                item=a[i];
                j=i-1;
                while(item<a[j] && j>=0)
```

```
        {
                a[j+1]=a[j];
                j=j-1;
        }
        a[j+1]=item;
    }

    for(i=0;i<=n-1;i++)
        printf("%d\t",a[i]);
    getch();
}
```

Output:

```
Enter number of elements:5
Enter 5 elements
4       3       -1      2       1
Sorted elements in ascending order:
-1      1       2       3       4
```

2 b. Discuss Binary search algorithm in detail.

Binary search algorithm finds given element in a list of elements with **O(log n)** time complexity where **n** is total number of elements in the list.

- The binary search algorithm can be used with only sorted list of element. That means, binary search can be used only with list of element which are already arranged in a order.
- The binary search cannot be used for list of element which are in random order.
- This search process starts comparing of the search element with the middle element in the list.
- If both are matched, then the result is "element found". Otherwise, we check whether the search element is smaller or larger than the middle element in the list.
- If the search element is smaller, then we repeat the same process for left sub-list of the middle element.
- If the search element is larger, then we repeat the same process for right sub-list of the middle element.
- We repeat this process until we find the search element in the list or until we left with a sub-list of only one element.
- And if that element also doesn't match with the search element, then the result is "Element not found in the list".

Binary search is implemented using following steps...

- **Step 1:** Read the search element from the user
- **Step 2:** Find the middle element in the sorted list
- **Step 3:** Compare, the search element with the middle element in the sorted list.
- **Step 4:** If both are matching, then display "Given element found!!!" and terminate the function
- **Step 5:** If both are not matching, then check whether the search element is smaller or larger than middle element.
- **Step 6:** If the search element is smaller than middle element, then repeat steps 2, 3, 4 and 5 for the left sublist of the middle element.
- **Step 7:** If the search element is larger than middle element, then repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.
- **Step 8:** Repeat the same process until we find the search element in the list or until sublist contains only one element.
- **Step 9:** If that element also doesn't match with the search element, then display "Element not found in the list!!!" and terminate the function.

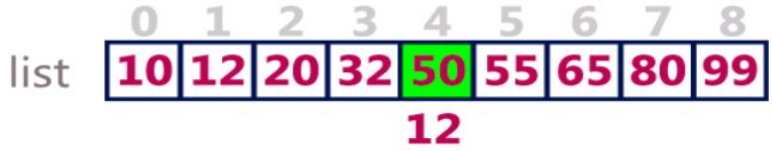
Example

Consider the following list of element and search element...

Example 1:

Step 1:

search element (12) is compared with middle element (50)



Both are not matching. And 12 is smaller than 50. So we search only in the left sublist (i.e. 10, 12, 20 & 32).

Step 2:

search element (12) is compared with middle element (12)

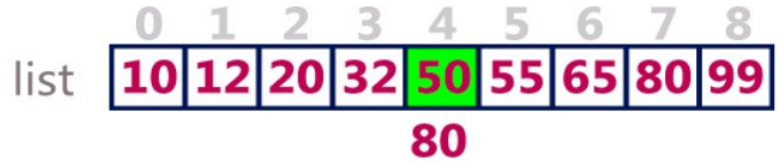
Both are matching. So the result is "Element found at index 1"

Example 2:

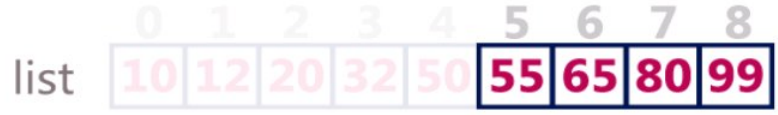
search element **80**

Step 1:

search element (80) is compared with middle element (50)



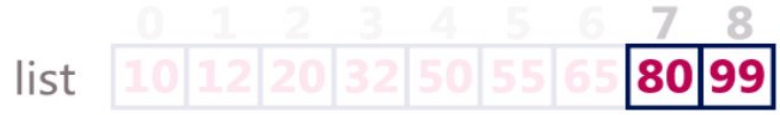
Both are not matching. And 80 is larger than 50. So we search only in the right sublist (i.e. 55, 65, 80 & 99).



Step 2:

search element (80) is compared with middle element (65)

Both are not matching. And 80 is larger than 65. So we search only in the right sublist (i.e. 80 & 99).



Step 3:

search element (80) is compared with middle element (80)

Both are not matching. So the result is "Element found at index 7"

3a Explain insertion sort with the help of suitable example. Also write an algorithm?

Insertion sort algorithm arranges a list of elements in a particular order. In insertion sort algorithm, every iteration moves an element from unsorted portion to sorted portion until all the elements are sorted in the list.

The insertion sort algorithm is performed using following steps...

- **Step 1:** Assume that first element in the list is in sorted portion of the list and remaining all elements are in unsorted portion.
- **Step 2:** Consider first element from the unsorted list and insert that element into the sorted list in order specified.
- **Step 3:** Repeat the above process until all the elements from the unsorted list are moved into the sorted list.

Consider the following unsorted list of elements...

Asume that sorted portion of the list empty and all elements in the list are in unsorted portion of the list as shown in the figure below...

Move the first element 15 from unsorted portion to sorted portion of the list.

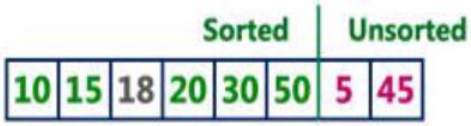
To move element 20 from unsorted to sorted portion, Compare 20 with 15 and insert it at correct position

To move element 10 from unsorted to sorted portion, Compare 10 with 20 and it is smaller so swap. Then compare 10 with 15 again smaller swap. And 10 is insert at its correct position in sorted portion of the list.

To move element 30 from unsorted to sorted portion, Compare 30 with 20, 15 and 10. And it is larger than all these so 30 is directly inserted at last position in sorted portion of the list.

To move element 50 from unsorted to sorted portion, Compare 50 with 30, 20, 15 and 10. And it is larger than all these so 50 is directly inserted at last position in sorted portion of the list.

To move element 18 from unsorted to sorted portion, Compare 18 with 30, 20 and 15. Since 18 is larger than 15, move 20, 30 and 50 one position to the right in the list and insert 18 after 15 in the sorted portion.



To move element 5 from unsorted to sorted portion, Compare 5 with 50, 30, 20, 18, 15 and 10. Since 5 is smaller than all these element, move 10, 15, 18, 20, 30 and 50 one position to the right in the list and insert 5 at first position in the sorted list.

To move element 45 from unsorted to sorted portion, Compare 45 with 50 and 30. Since 45 is larger than 30, move 50 one position to the right in the list and insert 45 after 30 in the sorted list.

Unsorted portion of the list has became empty. So we stop the process. And the final sorted list of elements is as follows...

3b. Write a C program to find the given number is prime or not

Program:

```
#include <stdio.h>
#include<conio.h>
void  main()
{
  int i, num, count = 0;
  printf("Please enter a number: \n");
  scanf("%d", &num);
  for(i=1; i<=num; i++)
  {
    if(num%i==0)
    {
      count++;
    }
  }
  if(count==2)
  {
    printf("Entered number is %d and it is a prime number.",num);
  }
  else
  {
    printf("Entered number is %d and it is not a prime number.",num);
  }
}
```

OUTPUT:

Please enter a number: 13
Entered number is 13 and it is a prime number.

4a. Sort the following 10 elements using bubble sort technique and explain with different passes?

56 78 31 45 92 10 78 23 35 66

Since we have to sort 10 elements using bubble sort, we need to perform (n-1 passes) i.e 10-1= 9 passes are required

First Pass

56	78	31	45	92	10	78	23	35	66- No Swapping
56	31	78	45	92	10	78	23	35	66- Swapped
56	31	45	78	92	10	78	23	35	66- Swapped
56	31	45	78	92	10	78	23	35	66- No Swapping
56	31	45	78	10	92	78	23	35	66- Swapped
56	31	45	78	10	78	92	23	35	66-Swapped
56	31	45	78	10	78	23	92	35	66- Swapped
56	31	45	78	10	78	23	35	92	66-Swapped
56	31	45	78	10	78	23	35	66	92- Swapped

Second pass

31	56	45	78	10	78	23	35	66	92- Swapped
31	45	56	78	10	78	23	35	66	92- Swapped
31	45	56	78	10	78	23	35	66	92- No Swapping
31	45	56	10	78	78	23	35	66	92- Swapped
31	45	56	10	23	78	78	35	66	92- Swapped
31	45	56	10	23	35	78	78	66	92- Swapped
31	45	56	10	23	35	66	78	78	92- Swapped

Third pass

31	45	56	10	23	35	66	78	78	92- No Swapping
31	45	56	10	23	35	66	78	78	92- No Swapping
31	45	10	56	23	35	66	78	78	92- Swapped
31	45	10	23	56	35	66	78	78	92- Swapped
31	45	10	23	35	56	66	78	78	92- Swapped
31	45	10	23	35	56	66	78	78	92- No Swapping

Fourth pass

31	45	10	23	35	56	66	78	78	92- No Swapping
31	10	45	23	35	56	66	78	78	92- Swapped

31	10	23	45	35	56	66	78	78	92	- Swapped
31	10	23	35	45	56	66	78	78	92	- Swapped
31	10	23	35	45	56	66	78	78	92	- No Swapping

Fifth pass

10	31	23	35	45	56	66	78	78	92	- Swapped
10	23	31	35	45	56	66	78	78	92	- Swapped
10	23	31	35	45	56	66	78	78	92	- No Swapping
10	23	31	35	45	56	66	78	78	92	- No Swapping

Sixth pass

10	23	31	35	45	56	66	78	78	92	- No Swapping
10	23	31	35	45	56	66	78	78	92	- No Swapping
10	23	31	35	45	56	66	78	78	92	- No Swapping

Seventh pass

10	23	31	35	45	56	66	78	78	92	- No Swapping
10	23	31	35	45	56	66	78	78	92	- No Swapping

Eighth pass

10	23	31	35	45	56	66	78	78	92	- No Swapping

nineth pass

10	23	31	35	45	56	66	78	78	92	- No Swapping

After Ninth pass, elements are arranged in a ascending order

10	23	31	35	45	56	66	78	78	92

4b. Write a C program to find minimum and maximum numbers of a given set?

Program:

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a[25], i, large, small, n;
```

```
  clrscr();
  printf("Enter the size of array(max 25)\n");
  scanf("%d", &n);
  printf("Enter any %d integer array elements\n",n);
  for(i = 0; i < n; i++)
  {
   scanf("%d", &a[i]);
  }
  large = a[0];
  small = a[0];
  for(i = 1; i < n ; i++)
  {
    if(a[i] > large)
    {
   large = a[i];
    }
    if(a[i] < small)
    {
   small = a[i];
    }
  }
  printf("The largest element from the given array is %d \nThe smallest element  from the given
array is %d", large, small);
  getch();
}
```

Output:

```
Enter the size of array(max 25)
5
Enter any 5 integers array elements
10 2 3 1 5
The largest element from the given array is 10
The smallest element from the given array is 1
```

5a. Write a C Program to search a key element from the list using binary search?

Program:

```
#include <stdio.h>
#include<conio.h>
void main()
{
int i,n,key,a[20];
clrscr();
printf("Enter the value of n\n");
```

```
scanf("%d",&n);
printf("Enter%d integers\n", n);
for (i=0;i<n;i++ )
scanf("%d",&a[i]);
printf("Enter value to be searched\n");
scanf("%d",&key);
binarysearch(a,n,key);
getch();
}

void binarysearch(int a[20],int n,int key)
{
int i,low,high,mid;
low=0;
high=n-1;
mid=(low+high)/2;
while(low<=high)
{
if(a[mid]==key)
{
printf("%d found at location %d.\n", key, mid);
break;
}
if(key>a[mid])
low= mid+1;
if(key<a[mid])
high=mid-1;
mid=(low+high)/2;
}
if(low>high)
printf("%d Not found, it is not present in the list.\n",key);
}
```

5b. Write a C program that sorts the given array of integers using selection sort in ascending order

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,a[20],n,temp,small;
clrscr();
printf("enter value of n:");
scanf("%d",&n);
printf("enter %d elements:\n",n);
for(i=0;i<=n-1;i++)
scanf("%d",&a[i]);
```

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

06 Write a C Program and algorithm to sort the array of element using bubble sort?

- Bubble Sort is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements. Bubble Sort compares all the element one by one and sort them based on their values.
- If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will swap both the elements, and then move on to compare the second and the third element, and so on.
- If we have total n elements, then we need to repeat this process for n-1 times.
- It is known as bubble sort, because with every complete iteration the largest element in the given array, bubbles up towards the last place or the highest index, just like a water bubble rises up to the water surface.
- Sorting takes place by stepping through all the elements one-by-one and comparing it with the adjacent element and swapping them if required.

Bubble Sort Algorithm

Following are the steps involved in bubble sort(for sorting a given array in ascending order):

- Starting with the first element(index = 0), compare the current element with the next element of the array.
- If the current element is greater than the next element of the array, swap them.
- If the current element is less than the next element, move to the next element. Repeat Step 1.

Bubble sort algorithm in programming figure.

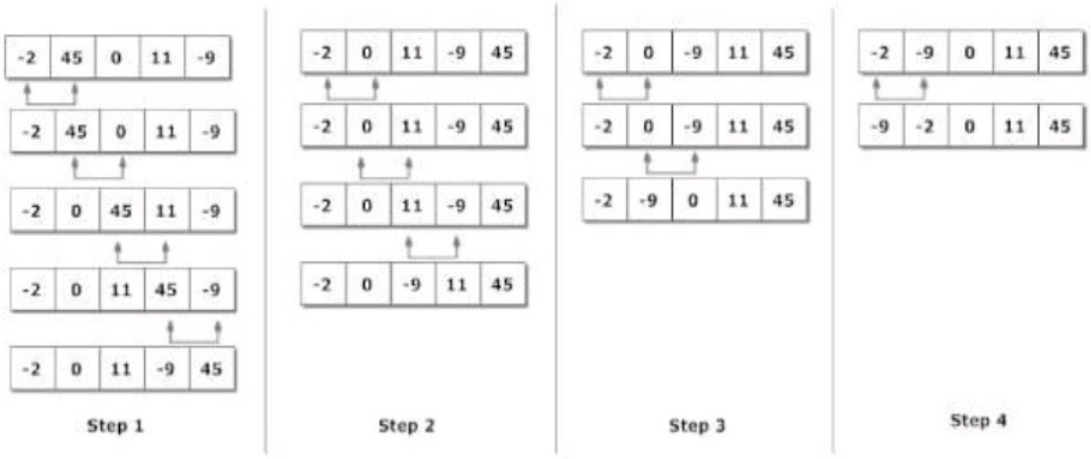


Figure: Working of Bubble sort algorithm

Program To Sort Elements using Bubble Sort Algorithm

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,a[20],n,temp,small;
clrscr();
printf("enter value of n:");
scanf("%d",&n);
printf("enter %d elements:\n",n);
for(i=0;i<=n-1;i++)
scanf("%d",&a[i]);
printf("sorted elements are:\n");
for(i=0;i<=n-1;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;
}
}
}
for(i=0;i<=n-1;i++)
printf("%d\t",a[i]);
getch();
}
```