EXPERIMENT - 01

1. a) Develop a Program in C for the following: a) Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), the second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).

```
#include<stdio.h>
#include<string.h>
struct Day
{
   char name[20];
   int date;
   char activity[100];
};
int main()
{
   struct Day calendar[7];
   strcpy(calendar[0].name,"Monday");
   calendar[0].date=1;
   strcpy(calendar[0].activity,"Work from 9 am to 5 pm");

   strcpy(calendar[1].name,"Tuesday");
   calendar[1].date=2;
   strcpy(calendar[1].activity,"Meeting at 10 pm");

   strcpy(calendar[2].name,"Wednesday");
   calendar[2].date=3;
   strcpy(calendar[2].activity,"Gym at 6 pm");

  strcpy(calendar[3].name,"Thrusday");
   calendar[3].date=4;
  strcpy(calendar[3].activity,"Dinner with friends at 7 pm");


  strcpy(calendar[4].name,"friday");
   calendar[4].date=5;
  strcpy(calendar[4].activity,"Movie night at 8 pm");

   strcpy(calendar[5].name,"saturday");
   calendar[5].date=6;
   strcpy(calendar[5].activity,"Weekend gateway");

   strcpy(calendar[6].name,"sunday");
```

```
   calendar[6].date=7;
   strcpy(calendar[6].activity,"Relax and Recharge");

   printf("calendar for the week:\n");
   for(int i=0;i<7;i++)
   {
      printf("%s(Date:%d):%s\n",calendar[i].name,calendar[i].date,calendar[i].activity);
   }
   return 0;
   }
```

Sample Output:

calendar for the week:

Monday(Date:1):Work from 9 am to 5 pm

Tuesday(Date:2):Meeting at 10 pm

Wednesday(Date:3):Gym at 6 pm

Thrusday(Date:4):Dinner with friends at 7 pm

friday(Date:5):Movie night at 8 pm

saturday(Date:6):Weekend gateway

sunday(Date:7):Relax and Recharge

b) Write functions create (), read() and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen.

```
#include <stdio.h>

#include<string.h>

struct Day

{

   char name[10];

   int date;

   char activity[50];

};

void create(struct Day calendar[7])

{
```

```
    for(int i=0;i<7;i++)
    {
    printf("Enter details for %s:\n", calendar[i].name);
    printf("date;");
    scanf("%d",&calendar[i].date);
    printf("activity");
    scanf(" %s", calendar[i].activity);
    }
}
void read(struct Day calendar[7])
{
    FILE *file = fopen("calendar.txt", "r");
    if (file == NULL) {
        printf("Error opening the file.\n");
        return;    }
        for (int i = 0; i < 7; i++) {
            fscanf(file, "%d", &calendar[i].date);
            fscanf(file, " %[^\n]", calendar[i].activity);

        }    fclose(file);}
        void display(struct Day calendar[7])
        {
            printf("Calendar for the week:\n");
            for (int i = 0; i < 7; i++)
```

```
{
   printf("%s (Date: %d): %s\n", calendar[i].name, calendar[i].date, calendar[i].activity);
   }}
   int main()
   {
      struct Day calendar[7];
       // Initialize the names of the days
       strcpy(calendar[0].name, "Monday");
       strcpy(calendar[1].name, "Tuesday");
       strcpy(calendar[2].name, "Wednesday");
       strcpy(calendar[3].name, "Thursday");
       strcpy(calendar[4].name, "Friday");
       strcpy(calendar[5].name, "Saturday");
       strcpy(calendar[6].name, "Sunday");
       int choice;
       printf("1. Create Calendar\n");
       printf("2. Read Calendar from File\n");
       printf("Enter your choice: ");
       scanf("%d", &choice);
       switch (choice) {
          case 1:          create(calendar);          break;
          case 2:          read(calendar);          break;
          default:          printf("Invalid choice.\n");
          return 1;    }
```

```
    display(calendar);
  return 0;


    }
```

INPUT:

Enter the name of day 1: Monday
Enter the date for Monday: 12/10/2023
Enter activity description for Monday: Dance
Enter the name of day 2: Tuesday
Enter the date for Tuesday: 13/10/2023
Enter activity description for Tuesday: Singing
Enter the name of day 3: Wednesday
Enter the date for Wednesday: 14/10/2023
Enter activity description for Wednesday: Quiz
……..

OUTPUT:

Week's Activity Details:

DAY DATE ACTIVITY

Monday 12/10/2023 Dance
Tuesday 13/10/2023 Singing
Wednesday 14/10/2023 Quiz
…….

EXPERIMENT - 02

Design, Develop and Implement a program in C for the following operations on Strings

a. **Read a Main String (STR), a Pattern String (PAT) and a Replace String (REP).**
b. **Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Repost suitable messages in case PAT does not exist in STR.**

Support the program with functions for each of the above operations. Don't use built-in functions.

ABOUT THE EXPERIMENT:

Strings are actually one-dimensional array of characters terminated by a **null** character '\0'.Thus a null-terminated string contains the characters that comprCSE the string followed by a **null**.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

If you follow the rule of array initialization then you can write the above statement as follows:

char greeting[] = "Hello";

C language supports a wide range of built-in functions that manipulate null-terminated strings as follows:

strcpy(s1, s2); Copies string s2 into string s1.
strcat(s1, s2); Concatenates string s2 onto the end of string s1.
strlen(s1); Returns the length of string s1.
strcmp(s1, s2); Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
strchr(s1, ch); Returns a pointer to the first occurrence of character ch in string s1.
strstr(s1, s2); Returns a pointer to the first occurrence of string s2 in string s1.

ALGORITHM:

Step 1: Start.
Step 2: Read main string STR, pattern string PAT and replace string REP.
Step 3: Search / find the pattern string PAT in the main string STR.
Step 4: if PAT is found then replace all occurrences of PAT in main string STR with REP string.
Step 5: if PAT is not found give a suitable error message.
Step 6: Stop.

PROGRAM CODE:

```
#include<stdio.h>
void main()
{
Char STR[100],PAT[100],REP[100],ans[100];
Int i,j,c,m,k,flag=0;
printf("\n Enter the MAIN string:\n");
gets(STR);
```

```
Printf("\n Enter the PATTERN string:\n");
gets(PAT);
Printf("\n Enter the REPLACE  string:\n");
gets(REP);
i=m=c=j=0;
While(STR[c]!='\0')
{
        if(STR[m]==PAT[i])
        {
                i++;
                m++;
                flag=1;
                if(PAT[i]=='\0')



                {
                For(k=0;REP[k]!='\0';k++,j++)
                Ans[j]=REP[k];
                i=0;
                c=m;
                }
        }
        else
        {
        ans[j]=STR[c];
        j++;
        c++;
        m=c;
          i=0;
        }
}
if(flag==0)

        Printf("Pattern doesnot found");
else
{
ans[j]='\0';
Printf("\n The Resultant String is:%s\n",ans);
}
}
```

SAMPLE OUTPUT:

Enter the MAIN string: good morning
Enter a PATTERN string: morning
Enter a REPLACE string: evening
The RESULTANT string is: good evening
Enter the MAIN string: hi vcet

Enter a PATTERN string: bye
Enter a REPLACE string: hello
The RESULTANT string is: pattern doesn't found!

EXPERIMENT - 03

Design, Develop and Implement a menu driven program in C for the following operations on STACK of integers (Array implementation of stack with maximum size MAX)

a. **Push an element on to stack**
b. **Pop an element from stack.**
c. **Demonstrate how stack can be used to check palindrome.**
d. **Demonstrate Overflow and Underflow situations on stack.**
e. **Display the status of stack.**
f. **Exit.**

Support the program with appropriate functions for each of the above operations.

ABOUT THE EXPERIMENT:

A stack is an abstract data type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example − deck of cards or pile of plates etc.


A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from top of the stack only. LikewCSE, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.

This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last is accessed first. In stack terminology, insertion operation is called **PUSH** operation and removal operation is called **POP** operation.

Below given diagram tries to depict a stack and its operations −

A stack can be implemented by means of Array, Structure, Pointer and Linked-List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays which makes it a fixed size stack implementation.

Basic Operations:

- **push()** - pushing (storing) an element on the stack.
- **pop()** - removing (accessing) an element from the stack.

To use a stack efficiently we need to check status of stack as well. For the same purpose, the following functionality is added to stacks;

- **peek()** − get the top data element of the stack, without removing it.
- **isFull()** − check if stack is full.
- **isEmpty()** − check if stack is empty.

ALGORITHM:

Step 1: Start.
Step 2: Initialize stack size MAX and top of stack -1.
Step 3: Push integer element on to stack and display the contents of the stack.
if stack is full give a message as 'Stack is Overflow'.
Step 3: Pop element from stack along with display the stack contents.
if stack is empty give a message as 'Stack is Underflow'.
Step 4: Check whether the stack contents are Palindrome or not.
Step 5: Stop.

PROGRAM CODE:

```
#include<stdio.h>
#define MAX 5
int stack[MAX];
int top=-1;

//a. Push an Element on to Stack
void push()
{
    int item;
    // Stack Overflow situations
    if(top==(MAX-1))
          printf("\n Stack Overflow");
    else
       {
         printf("\n Enter the element to be pushed :");
         scanf("%d",&item);
         // pushing element to the top of stack
         stack[++top]=item;
       }

}

//b. Pop an Element from Stack
void pop()
{
   // Stack Underflow situations
    if(top==-1)
         printf("\n Stack Underflow");
    else
```

```
        printf(" \nPoped element is %d ",stack[top--]);  // poping element from the top of stack
}

//e. Display the status of Stack
void display()
{
     int i;
     if(top==-1)
          printf("\n Sorry Empty Stack");
      else
        {
           printf("\nThe elements of the stack are\n");
           for(i=top;i>=0;i--)
                printf("stack[%d] = %d\n",i, stack[i]);
        }
 }

//c. Demonstrate how Stack can be used to check Palindrome
void palindrome()
{
    int i,count=0;
    for(i=0; i<=(top/2); i++)
    {
        if(stack[i] == stack[top-i])
            count++;
    }
    if((top/2+1)==count)
        printf("\n Stack contents are Palindrome");
    else
        printf("\nStack contents are not palindrome");
}


void main()
{
    int choice;
    while(1)
     {
      printf("\n STACK OPERATIONS\n");
      printf("1.Push\n 2.Pop\n 3.Display\n 4.Palindrome\n 5.Exit\n");
      printf("Enter your choice\n");
      scanf("%d",&choice);
      switch(choice)
       {
        case 1:push();
               break;
```

```
        case 2:pop();
               break;
        case 3:display();
               break;
        case 4:palindrome();
               break;
        case 5:return;
        default: printf("Invalid choice\n");
        }
      }
}
```

SAMPLE OUTPUT:

```
----MAIN MENU----
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice:        1
Enter an element to be pushed:              1

The stack contents are:
 1

----MAIN MENU----
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice:        1
Enter an element to be pushed:              2

The stack contents are:
 2
 1



    AFTER THE 4 TIMES PUSH OPERATION
----MAIN MENU----
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice:        1
Enter an element to be pushed:              9
Stack is Overflow
The stack contents
are:
```

1

2
2
1

----MAIN MENU----
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice: 2
Popped element is 1

The stack contents are:
2
2
1

(..... AFTER THE 4 TIMES POP OPERATION -------)
----MAIN MENU----
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice: 2
Stack is Underflow
The stack contents are:
Stack is Empty

----MAIN MENU----
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice: 1
Enter an element to be pushed: 1

The stack contents are:
1

----MAIN MENU----
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice: 1
Enter an element to be pushed: 2

The stack contents are:
2
1

----MAIN MENU----
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice: 1
Enter a element to be pushed: 1

The stack contents are:
1
2
1

----MAIN MENU----
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice: 3
Stack content is Palindrome

(AFTER 3 TIMES PUSH)
----MAIN MENU----
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice: 1
Enter an element to be pushed: 3

The stack contents are:
3
2
1

----MAIN MENU----
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice: 3
Stack contents are not Palindrome

EXPERIMENT - 04

Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, %(Remainder), ^ (Power) and alphanumeric operands.

ABOUT THE EXPERIMENT:

Infix: Operators are written in-between their operands. Ex:
X + Y **Prefix:** Operators are written before their operands.
Ex: +X Y **postfix:** Operators are written after their operands.
Ex: XY+

Examples of Infix, Prefix, and Postfix

Infix Expression	Prefix Expression	Postfix Expression
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +

ALGORITHM:
Step 1: Start.
Step 2: Read an infix expression with parenthesis and without parenthesis.
Step 3: convert the infix expression to postfix expression.
Step 4: Stop

PROGRAM CODE:

```
#include<stdio.h>
void infix_to_postfix();
void push(char);
char pop();
int priority(char);
char infix[30], postfix[30],stack[30];
int top=-1;

void main()
{
        printf("Enter the valid Infix expression \n");
        scanf("%s",infix);
        infix_to_postfix();
        printf("\n Infix expression : %s",infix);
        printf("\n Postfix expression : %s\n",postfix);
}

void push(char item)
{
        stack[++top]=item;
}
char pop()
{
        return stack[top--];
}
```

```
// check the priority of operator
int priority(char symb)
{
        int p;
        switch(symb)
        {
                case '+':
                case '-': p=1;
                          break;
                case '*':
                case '/':
                case '%':         p=2;
                                  break;
                case '^':
                case '$': p=3;   break;
                case '(':
                case ')': p=0;break;
                case '#': p=-1;
                          break;
        }
        return p;
}

void infix_to_postfix()
{
        int i=0,j=0;
        char symb,temp;
        push('#');
        for(i=0;infix[i]!='\0';i++)
        {
                symb=infix[i];
                switch(symb)
                {
                case '(': push(symb);
                          break;
                case ')': temp=pop();
//pop all symbols from top of stack and store in postfix until (
                                  while(temp!='(')
                {
                  postfix[j++]=temp;
```

```
                    temp=pop();
                }
                break;
                        case'+':
                        case'-':
                        case'*':
                        case'/':
                        case'%':
                        case'^':
                        case'$': while(priority(stack[top])>=priority(symb))
// check for priority of operator
                {
                    temp=pop();
                    postfix[j++]=temp;
                }
                  push(symb);
                  break;
default: postfix[j++]=symb;
          } // end of switch
                    } // end of for
while(top>0) // pop remaining all symbols form top of stack and store to postfix
                    {
                            temp=pop();
                            postfix[j++]=temp;
       } // end of while
                    postfix[j]='\0'; // end string postfix
           } // end of function infix_to_postfix
```

SAMPLE OUTPUT:

Enter a valid infix expression (a+(b-c) *d)
The infix expression is: (a+(b-c) *d)
The postfix expression is: abc-d*+

EXPERIMENT - 05

Design, Develop and Implement a Program in C for the following Stack Applications

a. **Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^**
b. **Solving Tower of Hanoi problem with n disks.**

ABOUT THE EXPERIMENT:

a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^

Postfix/Suffix Expression: Operators are written after their operands. Ex: XY+

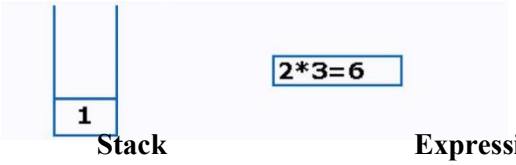
In normal algebra we use the infix notation like **a+b*c**. The corresponding postfix notation is **abc*+**

Example: **Postfix String:** **123*+4-**

Initially the Stack is empty. Now, the first three characters scanned are 1,2 and 3, which are operands. Thus they will be pushed into the stack in that order.

Stack **Expression**

Next character scanned is "*", which is an operator. Thus, we pop the top two elements from the stack and perform the "*" operation with the two operands. The second operand will be the first element that is popped.

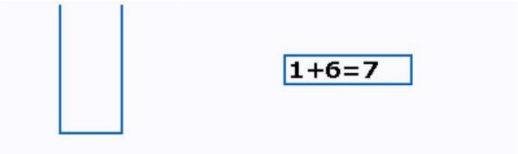


Stack **Expression**

The value of the expression(2*3) that has been evaluated(6) is pushed into the stack.

Stack **Expression**

Next character scanned is "+", which is an operator. Thus, we pop the top two elements from the stack and perform the "+" operation with the two operands. The second operand will be the first element that is popped.

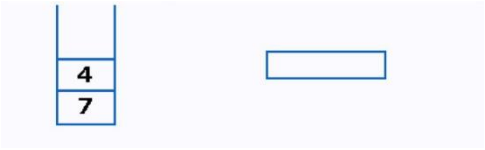


Stack **Expression**

The value of the expression(1+6) that has been evaluated(7) is pushed into the stack.

Stack **Expression**

Next character scanned is "4", which is added to the stack.



Stack **Expression**

Next character scanned is "-", which is an operator. Thus, we pop the top two elements from the stack and perform the "-" operation with the two operands. The second operand will be the first element that is popped.

Stack **Expression**

The value of the expression(7-4) that has been evaluated(3) is pushed into the stack.

Stack **Expression**

Now, since all the characters are scanned, the remaining element in the stack (there will be only one element in the stack) will be returned.

End result:

Postfix String: **123*+4-**

Result: 3

ALGORITHM:

Step 1: Start.
Step 2: Read the postfix/suffix expression.
Step 3: Evaluate the postfix expression based on the precedence of the operator.
Step 4: Stop.

PROGRAM CODE:

```
#include<stdio.h>
#include<math.h>
#include<string.h>
double compute(char symbol, double op1, double op2)
{
        switch(symbol)
        {
                case '+':       return op1 + op2;
                case' -':       return op1 - op2;
                case '*':       return op1 * op2;
                case '/':       return op1 / op2;
                case '$':
                case '^':       return pow(op1,op2);
                default:        return 0;
        }
}
void main()
{
        double s[20], res, op1, op2;
        int top, i;
        char postfix[20], symbol;
        printf("\nEnter the postfix expression:\n");
        gets(postfix);
        top=-1;
        for(i=0; <strlen(postfix); i++)
        {
                symbol = postfix[i];
                if(isdigit(symbol))
                        s[++top] = symbol - '0';
                else
                {
                        op2 = s[top--];
                        op1 = s[top--];
                        res = compute(symbol, op1, op2);
                        s[++top] = res;
                }
        }
        res = s[top--];
        printf("\nThe result is : %f\n", res);
}
```

SAMPLE OUTPUT:

RUN1:

Enter the postfix expression: 23+

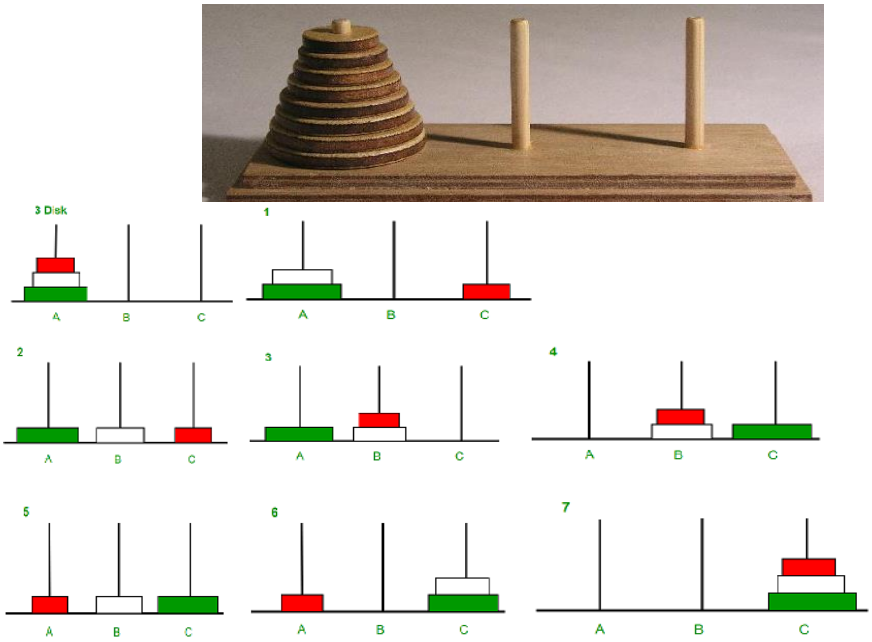
The result is: 5.000000

RUN2:
Enter the postfix expression: 23+7*
The result is: 35.000000

b. Solving Tower of Hanoi problem with n disks.

The **Tower of Hanoi** is a mathematical gameorpuzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conicalshape. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.
  With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is 2^n - 1, where *n* is the number of disks.



ALGORITHM:
Step 1: Start.
Step 2: Read N number of discs.
Step 3: Move all the discs from source to destination by using temp rod.
Step 4: Stop.

PROGRAM 5B:

```
#include<stdio.h>

void tower(int n, int source, int temp, int destination)
{
```

```
        if(n == 0)
                return;
        tower(n-1, source, destination, temp);
        printf("\nMove disc %d from %c to %c", n, source, destination);
        tower(n-1, temp, source, destination);
}

void main()
{
        int n;
        printf("\nEnter the number of discs: \n");
        scanf("%d", &n);
        tower(n, 'A', 'B', 'C');
        printf("\n\nTotal Number of moves are: %d", (int)pow(2,n)-1);
}
```

SAMPLE OUTPUT:

```
Enter the number of
discs: 3
Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C

Total Number of moves are:  7
```

EXPERIMENT - 06

Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)

a. **Insert an Element on to Circular QUEUE**
b. **Delete an Element from Circular QUEUE**
c. **Demonstrate *Overflow* and *Underflow* situations on Circular QUEUE**
d. **Display the status of Circular QUEUE**
e. **Exit**

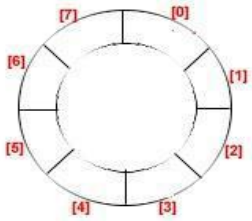
Support the program with appropriate functions for each of the above operations

ABOUT THE EXPERIMENT:

Circular queue is a linear data structure. It follows FIFO principle. In **circular queue** the last nodeis connected back to the first node to make a **circle**. **Circular** linked list fallow the First In

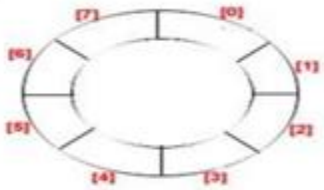
First Out principle. Elements are added at the rear end and the elements are deleted at front end of the **queue**. The queue is considered as a circular queue when the positions 0 and MAX-1 are adjacent. Any position before front is also after rear.

A circular queue looks like

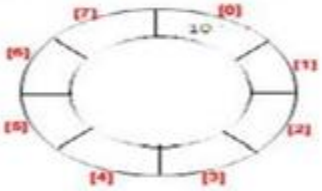


Consider the example with Circular Queue implementation:

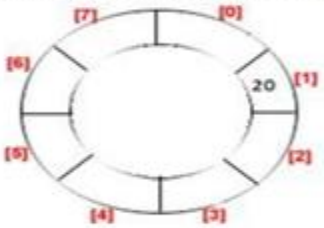
1) Initially: **Front = 0** and **rear = -1**



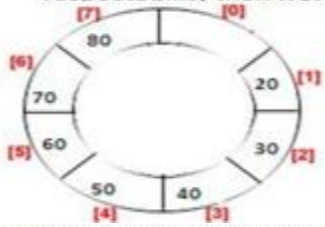
2) Add item 10 then **front = 0** and **rear =0.**



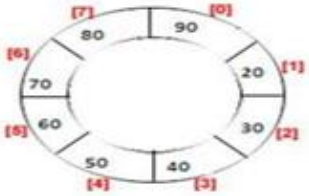
3) Now delete one item then **front = 1** and **rear =1.**



4) Like this now insert 30, 40, and 50,50,70,80 respectability then **front =1** and **rear = 7.**



5) Now in case of linear queue, we can not access 0 block for insertion but in circular queue next item will be inserted of 0 block then **front =0** and **rear = 0.**



ALGORITHM:

Step 1: Start.

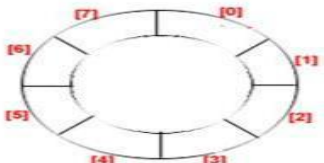
Step 2: Initialize queue size to MAX.

Step3: Insert the elements into circular queue. If queue is full give a message as 'queue is overflow"

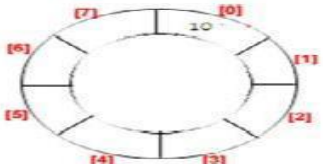
Step 4: Delete an element from the circular queue. If queue is empty give a message as'queue is underflow'.

Step 5: Display the contents of the queue.

1) Initially: **Front = 0** and **rear = -1**



2) Add item 10 then **front = 0** and **rear =0.**



Step 6: Stop.

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define max 5
int q[max],f=-1,r=-1;
void ins()
{
   if(f==(r+1)%max)
      printf("\nQueue overflow");
   else
   {
      if(f==-1)
         f++;
      r=(r+1)%max;
      printf("\nEnter element to be inserted:");
      scanf("%d",&q[r]);
   }
}
void del()
{
   if(r==-1)
      printf("\nQueue underflow");
   else
   {
      printf("\nElemnt deleted is:%d",q[f]);
      if(f==r)
         f=r=-1;
      else
         f=(f+1)%max;
   }
}
void disp()
{
   if(f==-1)
      printf("\nQueue empty");
   else
   {
      int i;
      printf("\nQueue elements are:\n");
      for(i=f;i!=r;i=(i+1)%max)
         printf("%d\t",q[i]);
      printf("%d",q[i]);
      printf("\nFront is at:%d\nRear is at:%d",q[f],q[r]);
   }
```

```
}
int main()
{
   printf("\nCircular Queue operations");
   printf("\n1.Insert");
   printf("\n2.Delete");
   printf("\n3.Display");
   printf("\n4.Exit");
   int ch;
   do{
      printf("\nEnter choice:");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1:ins();break;
         case 2:del();break;
         case 3:disp();break;
         case 4:exit(0);
         default:printf("\nInvalid choice...!");
      }
  }while(1);
   return 0;
}
```

SAMPLE OUTPUT:

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character / item to be inserted: 12
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character / item to be inserted: 13
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character / item to be inserted: 14
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character / item to be inserted: 15
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 3

Contents of Queue

12 13 14 15

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character / item to be inserted: 16

Queue is Full
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 2
Deleted item is: **12**
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 2
Deleted item is: **13**
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 3
Contents of Queue
14 15
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character / item to be inserted: 17
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 3
Contents of Queue
14 15 17
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 4

EXPERIMENT - 07

Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Branch, Sem, PhNo*

a. Create a SLL of N Students Data by using *front insertion*.
b. Display the status of SLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of SLL
d. Perform Insertion and Deletion at Front of SLL
e. Demonstrate how this SLL can be used as STACK and QUEUE
f. Exit

ABOUT THE EXPERIMENT:

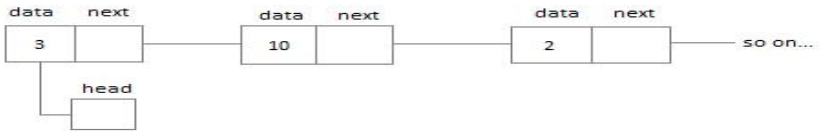
Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create trees and graphs.

- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.

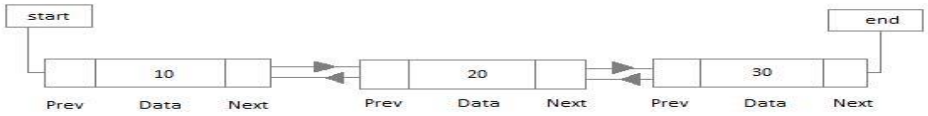
- Linked List reduces the access time.
- Linked lists are used to implement stacks, queues, graphs, etc.
- Linked lists let you insert elements at the beginning and end of the list.
- In Linked Lists we don't need to know the size in advance.

Types of Linked List:

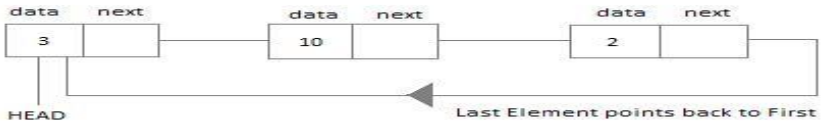
Singly Linked List: Singly linked lists contain nodes which have a data part as well as an addresspart i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.



Doubly Linked List: In a doubly linked list, each node contains two links the first link points to theprevious node and the next link points to the next node in the sequence.



Circular Linked List: In the circular linked list the last node of the list contains the address ofthe first node and forms a circular chain.



ALGORITHM:

Step 1: Start.
Step 2: Read the value of N. (N student's information)
Step 2: Create a singly linked list. (SLL)
Step 3: Display the status of SLL.
Step 4: Count the number of nodes.
Step 5: Perform insertion at front of list.
Step 6: Perform deletion at the front of the list.
Step 7: Perform insertion at end of the list.
Step 8: Perform deletion at the end of the list.

Step 9: Demonstrate how singly linked list can be used as stack.
Step 10: Demonstrate how singly linked list can be used as queue.
Step 11: Stop.

PROGRAM CODE:

```
#include<stdio.h>
struct student
{
        char usn[10], name[30], branch[5];
        int sem;
        char phno[10];
        struct student *next;    //Self referential pointer.
};
typedef struct student NODE;

NODE *first=NULL,*temp=NULL,*newnode=NULL,*cur,*prev;
int count;

NODE* getnode( )
{
        newnode=(NODE*)malloc(sizeof(NODE));  //Create first NODE
        printf("\nEnter USN, Name, Branch, Sem, Ph.No\n");
        scanf("%s",newnode->usn);
        scanf("%s",newnode->name);
        scanf("%s",newnode->branch);
        scanf("%d",&newnode->sem);
        scanf("%s",newnode->phno);
        newnode->next=NULL;         //Set next to NULL...
        return newnode;
}

void insert_front()
{
        newnode=getnode();
        newnode->next=first;
        first=newnode;
}

void delete_front()
{
        if(first==NULL)
        {
```

```
                printf("Linked list is empty\n");
                return;
        }
        if(first->next==NULL)
        {
                free(first);
                first=NULL;
                return;
        }
        temp=first;
        first=first->next;
        free(temp);
        }
}

void insert_rear()
{
        newnode=getnode();
        if(first==NULL)
                first=newnode;
        else
        {
                temp=first;
                while(temp->next!=NULL)
                {
                        temp=temp->next;
                }
                temp->next=newnode;
        }
}

void delete_rear()
{
        if(first==NULL)
        {
                printf("Linked list is empty\n");
                return;
        }
        if(first->next==NULL)
        {
                free(first);
                first=NULL;
                return;
```

```
        }
        cur=first;
        while(cur->next!=NULL)
        {
                prev=cur;
                cur=cur->next;
        }
        prev->next=NULL;
        free(cur);
}

void display()
{
        count=0;
        if(first == NULL)
        {
                printf("List empty to display \n");
                return;
        }
        temp=first;
        printf("\n");
        while (temp!= NULL)
        {
                printf("%s %s %s %d %s\n", temp->usn, temp->name, temp->branch, temp-
                        >sem,temp->phno );
                temp = temp->next;
                count++;
        }
        printf(" No of students = %d ", count);
}

void main()
{
        intch,n,i;
        printf("-----------------MENU----------------------\n");
        printf("\n 1 – Create \n 2 – Display \n 3 - Insert at the rear \n 4 – delete from rear");
        printf("\n 5 - Insert at front \n 6 - delete from front \n 7 - exit\n");
        printf("-----------------------------------------------\n");
        while (1)
        {
                printf("\n Enter choice : ");
                scanf("%d", &ch);
                switch (ch)
```

```
                {
                        case 1: printf("\n Enter no of students : ");
                                scanf("%d", &n);
                                for(i=0;i<n;i++)
                                insert_front();
                                break;
                        case 2:  display();break;
                        case 3:  insert_rear(); break;
                        case 4:  delete_rear(); break;
                        case 5:  insert_front();break;
                        case 6:  delete_front(); break;
                        case 7:  exit(0);
                        default: printf("Invalid choice\n");
                }
        }
}
```

SAMPLE OUTPUT:

```
–----------MENU----------
1– Create
2 - Display
3 - Insert at the rear
4 - deletefrom rear
5 - Insert at front
6 - deletefrom front
7 -exit
-----------------------------------------------
Enter choice : 1
Enter no of students : 2
Enter USN, Name, Branch, Sem, Ph.No : 007 vijay CSE 3 121
Enter USN, Name, Branch, Sem, Ph.No : 100 yashas CSE 3 911

Enter choice : 2
100 yashas CSE 3 911
007 vijay CSE 3 121
No of students = 2

Enter choice : 3
Enter USN, Name, Branch, Sem, Ph.No : 001 raj CSE 3 111
Enter choice : 2
100 yashas CSE 3 911
007 vijay CSE 3 121
001 raj CSE 3 111
No of students = 3
```

Enter choice : 4

Enter choice : 2
100 yashas CSE 3 911
007 vijay CSE 3 121
No of students = 2
Enter choice : 5
Enter USN, Name, Branch, Sem, Ph.No : 003 harsh cse 3 111

Enter choice : 2
003 harsh cse 3 111
100 yashas CSE 3 911
007 vijay CSE 3 121
No of students = 3

Enter choice : 6
Enterchoice : 2
100 yashas CSE 3 911
007 vijay CSE 3 121
No of students = 2
Enter choice : 7

EXPERIMENT - 08

Design, Develop and Implement a menu driven Program in C for the following operations on **Doubly Linked List (DLL)** of Employee Data with the fields: ***SSN, Name, Dept, Designation, Sal, PhNo.***

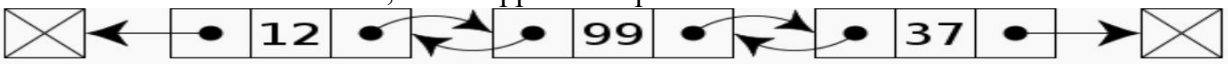
a. Create a **DLL** of **N** Employees Data by using ***end insertion***.
b. Display the status of **DLL** and count the number of nodes in it
c. Perform Insertion and Deletion at End of **DLL**
d. Perform Insertion and Deletion at Front of **DLL**
e. Demonstrate how this **DLL** can be used as **Double Ended Queue**
f. Exit

ABOUT THE EXPERIMENT:

Doubly Linked List: In a doubly linked list, each node contains two links the first link points to theprevious node and the next link points to the next node in the sequence.

In computer science, a **doubly linked list** is a linked data structure that consists of a set of sequentially linkedrecords called nodes. Each node contains two fields, called *links*, that

arereferences to the previous and to the next node in the sequence of nodes. The beginning and ending nodes' **previous** and **next** links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list. If there is only one sentinel node, then the list is circularly linked via the sentinel node. It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequential orders.



A doubly linked list whose nodes contain three fields: an integer value, the link to the next node, and the link to the previous node.The two node links allow traversal of the list in either direction. While adding or removing a node in a doubly linked list requires changing more links than the same operations on a singly linked list, the operations are simpler and potentially more efficient (for nodes other than first nodes) because there is no need to keep track of the previous node during traversal or no need to traverse the list to find the previous node, so that its link can be modified.

ALGORITHM:

Step 1: Start.
Step 2: Read the value of N. (N student's information)
Step 3: Create a doubly linked list. DLL)
Step 4: Display the status of DLL.
Step 5: Count the number of nodes.
Step 6: Perform insertion at front of list.
Step 7: Perform deletion at the front of the list.
Step 8: Perform insertion at end of the list.
Step 9: Perform deletion at the end of the list.
Step 10: Demonstrate how doubly linked list can be used as double ended queue.
Step 11: Stop.

PROGRAM CODE:

```
#include<stdio.h>
struct emp
{
int ssn;
char name[20], dept[10], desig[15], phno[10];
int sal;
struct emp *left;
struct emp *right;
};
typedef struct emp NODE;
NODE * first=NULL,*temp=NULL,*newnode=NULL;

NODE* getnode()
{
        newnode=(NODE*)malloc(sizeof(NODE)); //Create first NODE
        printf("\nEnter SSN, Name, Dept. , Desig, Ph.No and salary\n");
        scanf("%d",&(newnode->ssn));
        scanf("%s",newnode->name);
        scanf("%s",newnode->dept);
        scanf("%s",newnode->desig);
```

```
        scanf("%s",newnode->phno);
        scanf("%d",&(newnode->sal));
        newnode->left=NULL;
        newnode->right=NULL;
        return newnode;
}

void insert_front()
{
        NODE * newnode=getnode();
        newnode->right=first;
        if(first!=NULL)
                first->left=newnode;
        first=newnode;
}

void delete_front()
{
        if(first==NULL)
        {
                printf("Linked list is empty\n");
                return;
        }
        temp=first;
        if(first!=NULL)
                first=first->right;
        free(temp);
        first->left=NULL;
}

void insert_rear()
{
        newnode=getnode();
        if(first==NULL)
                first=newnode;
        else
        {
                temp=first;
                while(temp->right!=NULL)
                {
                        temp=temp->right;
                }
                temp->right=newnode;
                newnode->left=temp;
        }
}
```

```
void delete_rear()
{
        if(first==NULL)
        {
                printf("Linked list is empty\n");
                return;
        }
        if(first->right==NULL)
        {
                free(first);
                first=NULL;
                return;
        }
        temp=first;
        while(temp->right!=NULL)
        {
                temp=temp->right;
        }
        temp->left->right=NULL;
        free(temp);
}

void display()
{
        int count=0;
        if(first == NULL)
        {
                printf("List is empty..! \n");
                return;
        }
        temp=first;
        printf("\n----EMPLOYEE DATA----\n");
        printf("\nSSN\tNAME\t\Dept\tDesig\tPh.NO.\tSalary\n");
        while (temp!= NULL)
        {
                printf("%d\t%s\t%s\t%s\t%s\t%d\t%s\n", temp->ssn, temp->name, temp->dept,
                temp->desig,temp->phno, temp->sal );
                temp = temp->right;
                count++;
        }
        printf(" No of employees = %d ", count);
}

void main()
{
```

```
        intch,n,i;
        while (1)
        {
                printf("-----------------MENU----------------------\n");
                printf("\n 1–Create \n 2–Display \n 3–Insert at the rear \n 4–delete from rear");
                printf("\n 5–Insert at front \n 6–delete from front \n 7–exit\n");
                printf("-------------------------------------------\n");
                printf("\n Enter choice : ");
                scanf("%d", &ch);
                switch (ch)
                {
                        case 1: printf("\n Enter no of employees : ");
                                scanf("%d", &n);
                                for(i=0;i<n;i++)
                                insert_rear();
                                break;
                        case 2: display(); break;
                        case 3: insert_rear(); break;
                        case 4: delete_rear(); break;
                        case 5: insert_front(); break;
                        case 6: delete_front(); break;
                        case 7:  exit(0);
                        default: printf("Invalid choice\n");
                }
        }
}
```

SAMPLE OUTPUT:

```
–----------------MENU--------
--------------
1– Create
2 - Display
3 - Insert at the rear
4 - deletefrom rear
5 - Insert at front
6 - deletefrom front
7 -exit
--------------------------------
---------------
Enter choice : 1
Enter no of employees : 2
Enter SSN, Name, Dept.
,Desig, Ph.No and salary
1 RAJ SALES MANAGER
15000 911
Enter SSN, Name, Dept. ,Desig, Ph.No and
salary
2 RAVI HR ASST 10000 123

Enter choice : 2
1 RAJ SALES MANAGER 15000911
2 RAVI HR ASST
10000123
No of employees = 2

Enter choice : 3
Enter SSN, Name, Dept. ,Desig, Ph.No and
salary
3 RAM MARKET MANAGER 50000 111

Enter choice : 2
1 RAJ SALES MANAGER
```

15000911
2 RAVI HR ASST 10000123
3 RAM MARKET MANAGER 50000 111
No of employees = 3

Enter choice : 4
3 RAM MARKET MANAGER 50000111

Enter choice : 2
1 RAJ SALES MANAGER 15000911
2 RAVI HR ASST 10000 123
No of employees = 2

Enter choice : 5
Enter SSN, Name, Dept. ,Desig, Ph.No and salary
0 ALEX EXE TRAINEE 2000 133

Enter choice : 2
0 ALEX EXE TRAINEE 2000 133
1 RAJ SALES MANAGER 15000 911
2 RAVI HR ASST 10000 123
No of employees = 3

Enter choice : 6
0 ALEX EXE TRAINEE 2000 133

Enter choice : 2
1 RAJ SALES MANAGER 15000 911
2 RAVI HR ASST 10000 123

No of employees = 2
Enter choice : 7
Exit

EXPERIMENT - 09

Design, Develop and Implement a Program in C for the following operations on **Singly CircularLinked List (SCLL)** with header nodes

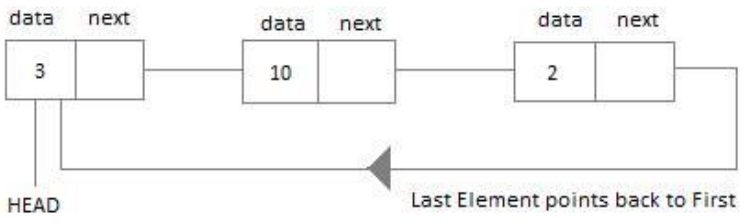
a. Represent and Evaluate a Polynomial **P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3**
b. Find the sum of two polynomials **POLY1(x,y,z)** and **POLY2(x,y,z)** and store the result in **POLYSUM(x,y,z)**

Support the program with appropriate functions for each of the above operations

ABOUT THE EXPERIMENT:

Circular Linked List:

In the circular linked list the last node of the list contains the address of the first node and forms a circular chain.

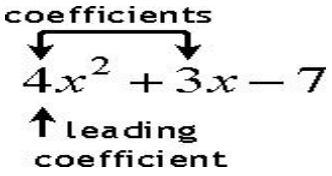


Polynomial:

A **polynomial equation** is an **equation** that can be written in the form. $ax^n + bx^{n-1} + \ldots + rx + s = 0$, where a, b, . . . , r and s are constants. We call the largest exponent of x appearing in a non-zero term of a **polynomial** the degree of that **polynomial**.

As with **polynomials** with one variable, you must pay attention to the rules of exponents and the order of operations so that you correctly **evaluate** an expression with two or more variables. **Evaluate** $x^2 + 3y^3$ for x = 7 and y = −2. Substitute the given values for x and y. **Evaluate** $4x^2y - 2xy^2 + x - 7$ for x = 3 and y = −1.

When a term contains both a number and a variable part, the number part is called the "coefficient". The coefficient on the leading term is called the "leading" coefficient.



In the above example, the coefficient of the leading term is 4; the coefficient of the second term is 3; the constant term doesn't have a coefficient.

Here are the steps required for Evaluating Polynomial Functions:

Step 1: Replace each x in the expression with the given value.

Step 2: Use the order of operation to simplify the expression.

Example 1 – Given $f(x) = -2x^2 + 5x - 7$, find f(3).

Step 1: Replace each x in the expression withthe given value. In this case, we replace each x with 3.	$f(3) = -2(3)^2 + 5(3) - 7$
Step 2: Use the order of operation to simplifythe expression.	$f(3) = -2(9) + 5(3) - 7$ $f(3) = -18 + 15 - 7$ $f(3) = -10$

Here are the steps required for addition of two polynomials.

Step 1

- Arrange the Polynomial in standard form
- Standard form of a polynomialjust means that the term with highest degree is first and each of the following terms

Step 2

- Arrange the like terms in columns and add the like terms

Example 1:*Let's find the sum of the following two polynomials*
$(3y^5 - 2y + y^4 + 2y^3 + 5)$ and $(2y^5 + 3y^3 + 2 + 7)$

1) Write in Standard form $(3y^5 + y^4 + 2y^3 - 2y + 5) + (2y^5 + 3y^3 + 7y + 2)$

2) Arrange in columns of like terms and then add $3y^5 + y^4 + 2y^3 - 2y + 5$

ALGORITHM:
Step 1: Start.
Step 2: Read a polynomial.
Step 3: Represent the polynomial using singly circular linked list.
Step 3: Evaluate the given polynomial
Step 4: Read two polynomials and find the sum of the polynomials.
Step 5: Stop

```
#include<stdio.h>
#include<math.h>
#include<alloc.h>
struct node
{
        int co,ex,ey,ez;
        int flag;
        struct node *link;
};
typedef struct node* poly;
poly attach(int c, int x, int y, int z, poly head)
{
        poly temp, cur;
        temp = (struct node *)malloc(sizeof(struct node));
        temp->co = c;
        temp->ex = x;
        temp->ey = y;
        temp->ez = z;
        cur = head;
        while(cur->link!= head)
                cur = cur->link;
        cur->link = temp;
```

```
        temp->link = head;
        return head;
}
poly read(poly head)
{
        int ch=1;
        int cf,x,y,z;
        while(ch!=0)
        {
                printf("Enter Coeff and 3 exponents: ");
                scanf("%d%d%d%d",&cf,&x,&y,&z);
                head=attach(cf,x,y,z,head);
                printf("\nIf you wish to continue press 1 otherwise press 0: ");
                scanf("%d", &ch);
        }
        return head;
}
void display(poly head)
{
        poly temp;
        if(head->link == head)
        {
                printf("polynomial does not exists\n");
                return;
        }
        temp =head->link;
        while(temp!=head)
        {
                if(temp->co < 0)
                        printf("- %d x^%d y^%d z^%d",temp->co,temp->ex,temp->ey,temp->ez);
                else
                        printf("+%d x^%d y^%d z^%d",temp->co,temp->ex,temp->ey,temp->ez);
                temp = temp->link;
        }
}

poly add(poly first, poly sec, poly res)
{
        poly a, b;
        int x1,x2,y1,y2,z1,z2,cf1,cf2,cf;
        a = first->link;
        while(a!=first)
        {
                x1=a->ex;
                y1=a->ey;
                z1=a->ez;
```

```
                cf1=a->co;
                b = sec->link;
                while(b!=sec)
                {
                        x2=b->ex;
                        y2=b->ey;
                        z2=b->ez;
                        cf2=b->co;
                        if(x1==x2 && y1==y2 && z1==z2)  // compare the exponent component
                                break;
                        b=b->link;
                }
                if(b!=sec)
                {
                        cf=cf1+cf2;
                        b->flag=1;
                        if(cf!=0)
                                res=attach(cf,x1,y1,z1,res);
                        a=a->link;
                }
                else
                {
                        res=attach(cf1,x1,y1,z1,res);
                        a=a->link;
                }
        }
        b=sec->link;
        while(b!=sec)
        {
                if(b->flag==0)
                        res=attach(b->co,b->ex,b->ey,b->ez,res);
                b=b->link;
        }
        return res;
}
void evaluate(poly head)
{
        poly h1;
        int x, y, z;
        int result=0;
        h1=head->link;
        printf("\nEnter values of x, y and z to evaluate:\n");
        scanf("%d%d%d", &x, &y, &z);
        while(h1!= head)
        {
                result = result +h1->co*pow(x,h1->ex)*pow(y,h1->ey)*pow(z,h1->ez);
```

```
                h1=h1->link;
        }
        printf("\nPolynomial result is: %d", result);
}
void main()
{
        int ch;
        poly eval,first,sec,res;
        first = (struct node *)malloc(sizeof(struct node));
        sec = (struct node *)malloc(sizeof(struct node));
        res = (struct node *)malloc(sizeof(struct node));
        eval=(struct node *)malloc(sizeof(struct node));

        first->link = first;
        sec->link = sec;
        res->link = res;
        eval->link = eval;
        printf("\n1.Evaluate polynomial\n2.Add 2 polynomials\n3.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: printf("Enter the polynomial\n");
                        eval = read(eval);
                        evaluate(eval);
                        break;
                case 2: printf("Enter the first polynomial\n");
                        first = read(first);
                        printf("Enter the second polynomial\n");
                        sec = read(sec);
                        res = add(first, sec, res);
                        printf("\nFirst Polynomial equation is\n");
                        display(first);
                        printf("\nSecond Polynomial equation is\n");
                        display(sec);
                        printf("\nResultant polynomial equation is \n");
                        display(res);
                        break;
                case 3: exit(0);
                default:printf("\ninvalid choice!!!");
        }
}
```

SAMPLE OUTPUT:

1. Evaluate polynomial
2. Add 2 polynomials
3. Exit

```
Enter your choice:1
Enter the polynomial
Enter Coeff and 3 exponents: 6 2 2 1
If you wish to continue press 1 otherwise press 0:1
Enter Coeff and 3 exponents: -4 0 1 5
If you wish to continue press 1 otherwise press 0:1
Enter Coeff and 3 exponents: 3 3 1 1
If you wish to continue press 1 otherwise press 0:1
Enter Coeff and 3 exponents: 2 1 5 1
If you wish to continue press 1 otherwise press 0:1
Enter Coeff and 3 exponents: -2 1 1 3
If you wish to continue press 1 otherwise press 0:0
Enter values of x, y and z to evaluate: 1 1 1
Polynomial result is:5

Enter your choice:2
Enter the first polynomial
Enter Coeff and 3 exponents: 4 2 2 2
If you wish to continue press 1 otherwise press 0:1
Enter Coeff and 3 exponents: 3 1 1 2
If you wish to continue press 1 otherwise press 0:0
Enter the second polynomial
Enter Coeff and 3 exponents:6 2 2 2
If you wish to continue press 1 otherwise press 0:0
First Polynomial equation is
4 x^2 y^2 z^2 + 3 x^1 y^1 z^2
Second Polynomial equation is
6 x^2 y^2 z^2
Resultant polynomial equation is
10 x^2 y^2 z^2 + 3 x^1 y^1 z^2
```

EXPERIMENT - 10

Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers

a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (KEY) and report the appropriate message
d. Delete an element (ELEM) from BST
e. Exit

ABOUT THE EXPERIMENT:

A **binary search tree** (BST) is a **tree** in which all nodes follows the below mentioned properties

- The left sub-**tree** of a node has key less than or equal to its parent node's key.
- The right sub-**tree** of a node has key greater than or equal to its parent node's key.

Thus, a binary search tree (BST) divides all its sub-trees into two segments; *left* sub-tree and *right* sub-tree and can be defined as

left_subtree (keys) ≤ node (key) ≤ right_subtree (keys)

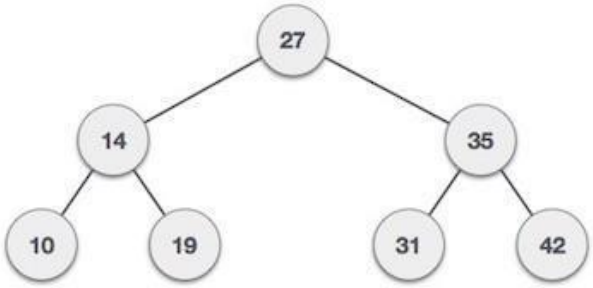


Fig: An example of BST

Following are basic primary operations of a tree which are following.

- **Search** − search an element in a tree.
- **Insert** − insert an element in a tree.
- **Preorder Traversal** − traverse a tree in a preorder manner.
- **Inorder Traversal** − traverse a tree in an inorder manner.
- **Postorder Traversal** − traverse a tree in a postorder manner.

ALGORITHM:

Step 1: Start.
Step 2: Create a Binary Search Tree for N elements.
Step 3: Traverse the tree in inorder.
Step 4: Traverse the tree in preorder
Step 6: Traverse the tree in postorder.
Step 7: Search the given key element in the BST.
Step 8: Delete an element from BST.
Step 9: Stop

PROGRAM CODE:

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

struct BST
{
        int data;
        struct BST *left;
        struct BST *right;
};

typedef struct BST NODE;

NODE *root=NULL,*cur,*prev;

NODE* insert(int item)
{
        NODE *temp;
        temp= (NODE*)malloc(sizeof(NODE));
        temp->data = item;
        temp->left = temp->right = NULL;
        if (root == NULL)
        {
                root=temp;
                return;
        }
        cur=root;
        while(cur!=NULL)
        {
                prev=cur;
                if (item < (cur->data))
                        cur=cur->left;
                else if(item>(cur->data))
                        cur=cur->right;
                else
                        return;
        }
        if (item <prev->data)
                prev->left=temp;
        Else
                prev->right=temp;
}

void search(NODE * node, int key)
{
```

```
        if(node == NULL)
                printf("\nElement not found");
        else if(key< node->data)
        {
                search(node->left,key);
        }
        else if(key> node->data)
        {
                search(node->right,key);
        }
        else
                printf("\nElement found”);
}

void inorder(NODE *node)
{
        if(node != NULL)
        {
                inorder(node->left);
                printf("%d\t", node->data);
                inorder(node->right);
        }
}

void preorder(NODE *node)
{
        if(node != NULL)
        {
                printf("%d\t", node->data);
                preorder(node->left);
                preorder(node->right);
        }
}

void postorder(NODE *node)
{
        if(node != NULL)
        {
                postorder(node->left);
                postorder(node->right);
                printf("%d\t", node->data);
        }
}

void del(int item)
{
        NODE *parent=NULL,*suc,*q,*cur;
        if(root == NULL)
```

```
        {
                printf("\n Tree empty");
                return;
        }
        cur=root;
        while(cur!=NULL)
        {
                if(item==cur->data)
                        break;
                parent=cur;
                cur=(item<cur->data)?cur->left:cur->right;
        }
        if(cur==NULL)
        {
                printf("Item not found\n");
                return;
        }
        if(cur->left==NULL)
                q=cur->right;
        else if(cur->right==NULL)
                q=cur->left;
        else
        {
                suc=cur->right;
                while(suc->left!=NULL)
                        suc=suc->left;
                suc->left=cur->left;
                q=cur-right;
        }
        if(parent==NULL)
        {
                root=q;
                return;
        }
        if(cur==parent->left)
                parent->left=q;
        else
                parent->right=q;
        free(cur);
}
void main()
{
        int item, ch, i, n;
        while (1)
        {
                printf("\n1.Create\n2.Search\n3.Delete\n4.Inorder\n5.Preorder\n6.Postorder\n7.Exit");
                printf("\n Enter your choice: ");
```

```
                scanf("%d", &ch);
                switch (ch)
                {
                        case 1: printf("\nEnteritems to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n");
                                for(i=0;i<12;i++)
                                {
                                        scanf("%d", &item);
                                        insert(item);
                                }
                                break;
                        case 2: printf("\n Enter the element to search: ");
                                scanf("%d", &item);
                                search(root, item); break;
                        case 3: printf("\nEnter the element to delete: ");
                                scanf("%d", &item);
                                del(item);
                                break;
                        case 4: printf("\n Inorder Traversal: \n");
                                inorder(root);
                                break;
                        case 5: printf("\nPreorder Traversal: \n");
                                preorder(root);
                                break;
                        case 6: printf("\nPostorder Traversal: \n");
                                postorder(root);
                                break;
                        case 7: exit(0);
                        default: printf("\n Wrong option"); break;
                }
        }
}
```

SAMPLE OUTPUT:

1. Create
2. Search
3. Delete
4. Inorder
5. Preorder
6. Postorder
7. Exit

Enter your choice: 1

Enter items to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)

6 9 5 2 8 15 24 14 7 8 5 2

Enter your choice: 4

Inorder Traversal:

2 5 6 7 8 9 14 15 24

Enter your choice: 5

Preorder Traversal:
6 5 2 9 8 7 15 14 24

Enter your choice: 6

Postorder Traversal:
2 5 7 8 14 24 15 9 6

Enter your choice: 2
Enter the element to search: 24
Element found

Enter your choice: 2
Enter the element to search: 50
Element not found

Enter your choice: 3
Enter the element to delete: 15

Enter your choice: 4

Inorder Traversal:
2 5 6 7 8 9 14 24

Enter your choice: 5

Preorder Traversal:
6 5 2 9 8 7 24 14

Enter your choice: 7

EXPERIMENT - 11

Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities

a. **Create a Graph of N cities using Adjacency Matrix.**
b. **Print all the nodes reachable from a given starting node in a digraph using BFS method**
c. **Check whether a given graph is connected or not using DFS method.**

ABOUT THE EXPERIMENT:

Adjacency Matrix

In **graph** theory, computer science, an **adjacency matrix** is a square **matrix** used to **represent** a finite **graph**. The elements of the **matrix** indicate whether pairs of vertices are adjacent or not in the

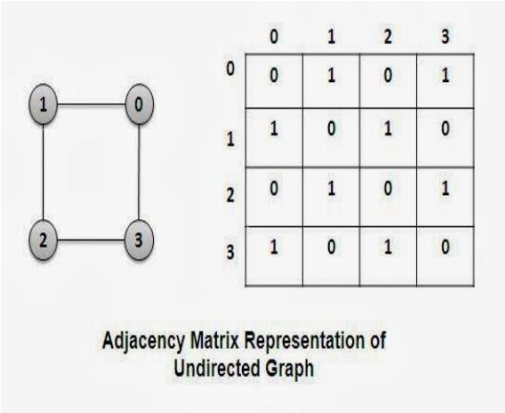
graph. In the special case of a finite simple **graph**, the **adjacency matrix** is a (0, 1)-**matrix** with zeros on its diagonal.

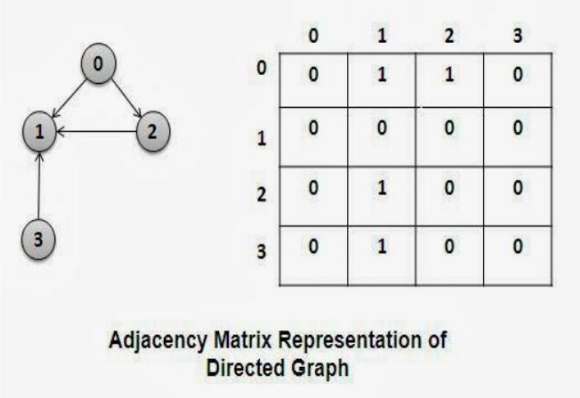
A graph G = (V, E) where v= {0, 1, 2, . . .n-1} can be represented using two dimensional integer array of size n x n.

a[20][20] can be used to store a graph with 20 vertices.
a[i][j] = 1, indicates presence of edge between two vertices i and j.
a[i][j] = 0, indicates absence of edge between two vertices i and j.

- A graph is represented using square matrix.
- Adjacency matrix of an undirected graph is always a symmetric matrix, i.e. an edge (i, j) implies the edge (j, i).
- Adjacency matrix of a directed graph is never symmetric, adj[i][j] = 1 indicates a directed edge from vertex i to vertex j.

An example of adjacency matrix representation of an undirected and directed graph is given below:

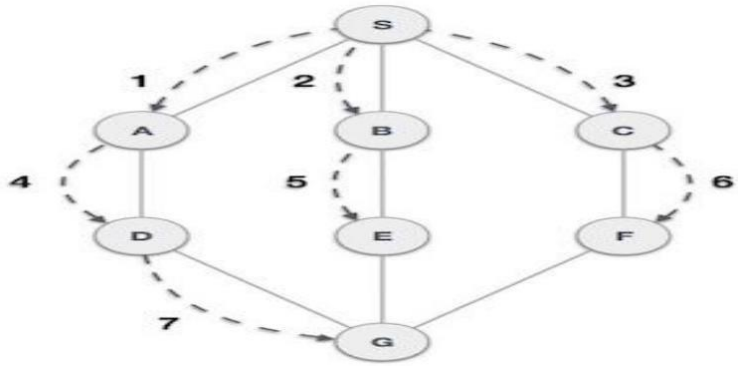




BFS

Breadth-first search (**BFS**) is analgorithmfortraversing or searchingtreeorgraphdatastructures. It starts at the tree rootand explores the neighbor nodes first, before moving to the next level neighbors.

Breadth First Search algorithm(BFS) traverses a graph in a breadth wards motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration.



As in example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs following rules.

- **Rule 1** − Visit adjacent unvisited vertex. Mark it visited. Display it. Insert it in a queue.
- **Rule 2** − If no adjacent vertex found, remove the first vertex from queue.
- **Rule 3** − Repeat Rule 1 and Rule 2 until queue is empty.

DFS

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

Depth-first search, or **DFS**, is a way to traverse the graph. Initially it allows visiting vertices of the graph only, but there are hundreds of algorithms for graphs, which are based on DFS. Therefore, understanding the principles of depth-first search is quite important to move ahead into the graph theory. The principle of the algorithm is quite simple: to go forward (in depth) while there is such possibility, otherwise to backtrack.

ALGORITHM:

Step 1: Start.
Step 2: Input the value of N nodes of the graph
Step 3: Create a graph of N nodes using adjacency matrix representation.
Step 3: Print the nodes reachable from the starting node using BFS.
Step 4: Check whether graph is connected or not using DFS.
Step 5: Stop.

PROGRAM CODE:

```
#include<stdio.h>

int a[10][10], n, m, i, j, source, s[10], vis[10], visited[10], count;

void create()
{
        printf("\nEnter the number of vertices of the digraph: "); scanf("%d",
        &n);
        printf("\nEnter the adjacency matrix of the graph:\n");
        for(i=1; i<=n; i++)
        {
                for(j=1; j<=n; j++)
                {
```

```
                        scanf("%d", &a[i][j]);
                        vis[i]=0;
                        visited[i]=0;
                }
        }
}

void bfs()
{
        int q[10], u, front=0, rear=-1;
        printf("\nEnter the source vertex: ");
        scanf("%d", &source);
        q[++rear] = source;
        visited[source] = 1;
        printf("\nThe reachable vertices are: ");
        while(front<=rear)
        {
                u = q[front++];
                for(i=1; i<=n; i++)
                {
                        if(a[u][i] == 1 && visited[i] == 0)
                        {
                                        q[++rear] = i;
                                        visited[i] = 1;
                                        printf("\n%d", i);
                        }
                }
        }
}




void dfs(int source)
{
        int v;
        vis[source] = 1;
        for(v=1; v<=n; v++)
        {
                if(a[source][v] == 1 &&vis[v] == 0)
                {
                        count++;
                        dfs(v);
                }
        }
}
```

```
void main()
{
        intch;
        while(1)
        {
                printf("\n1.Create\n2.BFS\n3.Check graph connected or not(DFS)\n4.Exit");
                printf("\nEnter your choice: ");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1: create();
                                break;
                        case 2: bfs();
                                break;
                        case 3: count=1;
                                for(i=1;i<=n;i++)
                                        if(vis[i]==0)
                                                dfs(i);
                                if(count==n)
                                        printf("\nGraph is Connected");
                                else
                                        printf("\nGraph is not Connected");
                                break;
                        default: exit(0);
                }
        }
}
```

SAMPLE OUTPUT:

1. Create 2.BFS 3.Check graph connected or not (DFS) 4.Exit
Enter your choice: 1

Enter the number of vertices of the digraph: 4

Enter the adjacency matrix of the graph:
0 0 1 1
0 0 0 0
0 1 0 0
0 1 0 0

1. Create 2.BFS 3.Check graph connected or not (DFS) 4.Exit
Enter your choice: 2

Enter the source vertex: 1 3 4 2

1. Create 2.BFS 3.Check graph connected or not (DFS) 4.Exit
Enter your choice: 3

Graph is Connected

1. Create 2.BFS 3.Check graph connected or not (DFS) 4.Exit
Enter your choice: 4

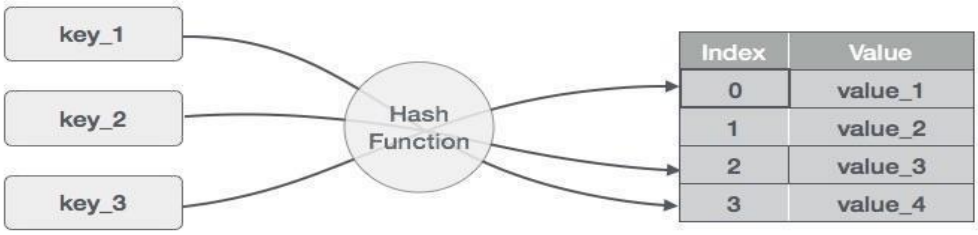
EXPERIMENT - 12

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K ®L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

ABOUT THE EXPERIMENT:

Hash Table is a data structure which store data in associative manner. In hash table, data is stored in array format where each data values has its own unique index value. Access of data becomes very fast if we know the index of desired data. Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of size of data. Hash Table uses array as a storage medium and uses hash technique to generate index where an element is to be inserted or to be located from.

Hashing: Hashing is a technique to convert a range of key values into a range of indexes of an array. We're going to use modulo operator to get a range of key values. Consider an example of hashtable of size 20, and following items are to be stored. Item are in (key, value) format.



Sl. No	Key	Hash	Index
1	1	1 % 20 = 1	1
2	2	2 % 20 = 2	2
3	42	42 % 20 = 2	2
4	4	4 % 20 = 4	4
5	12	12 % 20 = 12	12
6	14	14 % 20 = 14	14
7	17	17 % 20 = 17	17
8	13	13 % 20 = 13	13
9	37	37 % 20 = 17	17

Linear Probing:

As we can see, it may happen that the hashing technique used create already used index of the array. In such case, we can search the next empty location in the array by looking into the next cell until we found an empty cell. This technique is called linear probing.

Sl. No	Key	Hash	Index
1	1	1 % 20 = 1	1
2	2	2 % 20 = 2	2
3	42	42 % 20 = 2	3
4	4	4 % 20 = 4	4
5	12	12 % 20 = 12	12
6	14	14 % 20 = 14	14
7	17	17 % 20 = 17	17
8	13	13 % 20 = 13	13
9	37	37 % 20 = 17	18

Basic Operations

Following are basic primary operations of a hashtable which are following.

Search − search an element in a hashtable.

Insert − insert an element in a hashtable.

delete− delete an element from a hashtable.

ALGORITHM:

Step 1: Start.
Step 2: Given a File of **N** employee records with a set **K** of Keys (4-digit) which uniquely determine the records in file **F.**
Step 3: Assume that file **F** is maintained in memory by a Hash Table(HT) of **m** memory locations with **L** as the set of memory addresses (2-digit) of locations in HT.
Step 3: Let the keys in **K** and addresses in **L** are Integers
Step 4: Hash function **H: K** ®**L** as H(**K**)=**K** mod **m (remainder** method)
Step 5: Hashing as to map a given key **K** to the address space **L,** Resolve the collision (if any) is using **linear probing**.
Step6: Stop.

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 3

struct employee
{
int flag;
int eid;
char ename[15];
};

struct employee emp[MAX];

int hash(int key)
{
        int index;
        index = key % MAX;
        return index;
}

int linear_prob(int addr)
{
        int i=(addr+1)%MAX;
        while(i!=addr)
        {
                if(emp[i].flag!= -1)
                        i=(i++)%MAX;
```

```
                else
                        break;
        }
        if(i!=addr)
                return i;
        else
                return -1;
}

void insert()
{
        int id, addr;
        char name[15];
        printf("\nEnter emp id: ");
        scanf("%d", &id);
        printf("\nEnter emp name: ");
        scanf("%s", name);
        addr=hash(id);
        printf("addr=%d",addr);
        if(emp[addr].flag==-1)
        {
                emp[addr].eid=id;
                emp[addr].flag=1;
                strcpy(emp[addr].ename,name);
        }
        else
        {
                printf("\nCollision detected..");
                addr=linear_prob(addr);
                if(addr!=-1)
                {
                        emp[addr].eid=id;
                        emp[addr].flag=1;
                        strcpy(emp[addr].ename,name);
                }
                else
                {
                        printf("\n Hash Table is full.. Cannot insert");
                        return;
                }
        }
}

void display()
{
        inti;
        printf("\nThe hash table is:\n");
```

```
        printf("\nHTKey\tEmpID\tEmpName");
        for(i=0;i<MAX;i++)
        {
                if(emp[i].flag!=-1)
                {
                        printf("\n%d\t%d\t%s",i,emp[i].eid,emp[i].ename);
                        continue;
                }

        }
}




void main()
{
        int i, ch;
        printf("\n Collision handling by linear probing");
        for(i=0;i<MAX;i++)
        {
                emp[i].flag = -1;
        }
        for(;;)
        {
                printf("\n1.Insert\n2.Display\n");
                printf("Enter your choice");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: insert();
                                break;
                        case 2:display();
                                break;
                        default: exit(0);
                }
        }
}
```

SAMPLE OUTPUT:

```
Collision handling by linear probing        Enter emp id:       123
1. Insert                                   Enter emp name:     xyz
2.Display                                   addr=0

Enter your choice:  1                       Enter your choice:  1
```

```
Enter emp id:       3
Enter emp name:     jkl
addr=0
Collision detected..

Enter your choice:  1
Enter emp id:       2
Enter emp name:     pqr
addr=2

Enter your choice:  1
Enter emp id:       2
Enter emp name:     abc
addr=2
Collision detected..
Hash Table is full.. Cannot insert

Enter your choice:  2
The hash table is:
HTKey       EmpID       EmpName
  0           123          xyz
  1            3          jkl
  2            2          pqr
```

VIVA QUESTIONS FOR DATA STRUCTURES

1. What is a Register?

Ans- A register is a small amount of memory within the CPU that is used to temporarily store instructions and data.

2. An _____ data type is a keyword of a programming language that specifies the amount of memory needed to store data and the kind of data that will be stored in that memory location?

Ans-abstract

3.What is the different Abstract Data Type Groups?

Integer, Floating-Type, Character & Boolean are the four different data type groups
Explanation: You determine the amount of memory to reserve by determining the appropriate abstract data type group to use and then deciding which abstract data type within the group is right for the data. The different abstract data type groups are Integer, Floating-Type, Character & Boolean

4.Which of the following abstract data types are NOT used by Integer Abstract Data type group?

A) Short
B) Int
C) float
D) long
Explanation: The integer abstract data type group consists of four abstract data types used to reserve memory to store whole numbers: byte, short, int , and long

5.What pointer type is used to implement the heterogeneous linked list in C?

The answer is the void pointer. The heterogeneous linked list contains different data types in it's nodes and we need a link, pointer, to connect them. Since we can't use ordinary pointers for this, we use the void pointer. Void pointer is a generic pointer type, and capable of storing pointer to any type.

6.What is the minimum number of queues needed to implement the priority queue?

Two. One queue is used for the actual storing of data, and the other one is used for storing the priorities.

7.Which data structure is used to perform recursion?

The answer is Stack. Stack has the LIFO (Last In First Out) property; it remembers it's 'caller'. Therefore, it knows to whom it should return when the function has to return. On the other hand, recursion makes use of the system stack for storing the return addresses of the function calls.
Every recursive function has its equivalent iterative (non-recursive) function. Even when such equivalent iterative procedures are written explicit, stack is to be used.

8.What are some of the applications for the tree data structure?

1- Manipulation of the arithmetic expressions.
2- Symbol table construction.
3- Syntax analysis.

9.Which data strucutres algorithm used in solving the eight Queens problem?

Backtracking

10.In an AVL tree, at what condition the balancing is to be done?

If the "pivotal value", or the "height factor", is greater than one or less than minus one.

11.There are 8, 15, 13, and 14 nodes in four different trees. Which one of them can form a full binary tree?

The answer is the tree with 15 nodes. In general, there are 2^n-1 nodes in a full binary tree.
By the method of elimination:
Full binary trees contain odd number of nodes, so there cannot be full binary trees with 8 or 14 nodes. Moreover, with 13 nodes you can form a complete binary tree but not a full binary tree. Thus, the correct answer is 15.

12. What is data structure?
Answer: A data structure is a way of organizing data that considers not only the items stored, but also their relationship to each other. Advance knowledge about the relationship between data items allows designing of efficient algorithms for the manipulation of data.

13. List out the areas in which data structures are applied extensively?
Answer: The name of areas are:
a. Compiler Design,
b. Operating System,
c. Database Management System,
d. Statistical analysis package,
e. Numerical Analysis,
f. Graphics,
g. Artificial Intelligence,
h. Simulation

14. What are the major data structures used in the following areas : RDBMS, Network data model & Hierarchical data model.

The major data structures used are as follows:

a. RDBMS - Array (i.e. Array of structures)
b. Network data model - Graph
c. Hierarchical data model - Trees
d.

15. If you are using C language to implement the heterogeneous linked list, what pointer type will you use?

The heterogeneous linked list contains different data types in its nodes and we need a link, pointer to connect them. It is not possible to use ordinary pointers for this. So we go for void pointer. Void pointer is capable of storing pointer to any type as it is a generic pointer type.

16. Minimum number of queues needed to implement the priority queue?

Two. One queue is used for actual storing of data and another for storing priorities.

17. What is the data structures used to perform recursion?

Stack. Because of its LIFO (Last In First Out) property it remembers its 'caller' so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls.

Every recursive function has its equivalent iterative (non-recursive) function. Even when such equivalent iterative procedures are written, explicit stack is to be used.

18. What are the notations used in Evaluation of Arithmetic Expressions using prefix and postfix forms?

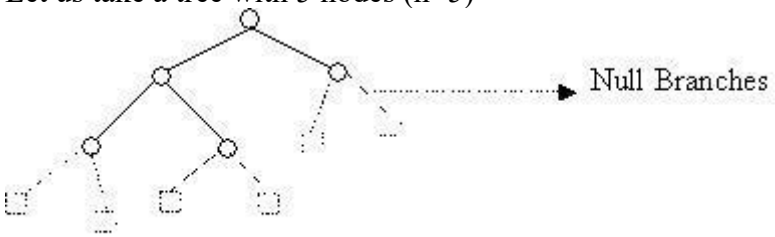
Polish and Reverse Polish notations.

19. Convert the expression ((A + B) * C - (D - E) ^ (F + G)) to equivalent Prefix and Postfix notations.

Prefix Notation: ^ - * +ABC - DE + FG
Postfix Notation: AB + C * DE - - FG + ^

20. How many null branches are there in a binary tree with 20 nodes?

Answer: 21
Let us take a tree with 5 nodes (n=5)



It will have only 6 (ie,5+1) null branches.
A binary tree with n nodes has exactly n+1 null nodes.

21. What are the methods available in storing sequential files?

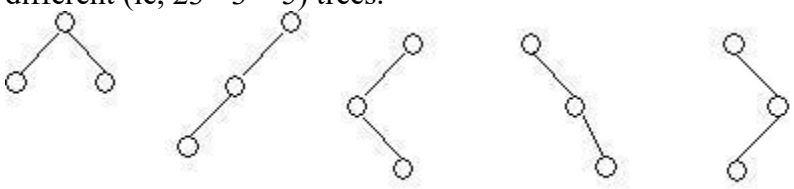
Answer: The methods available in storing sequential files are:

- Straight merging,
- Natural merging,
- Polyphase sort,
- Distribution of Initial runs.

22. How many different trees are possible with 10 nodes ?

Answer: 1014

For example, consider a tree with 3 nodes(n=3), it will have the maximum combination of 5 different (ie, 23 - 3 = 5) trees.



i ii iii iv v

In general:

If there are n nodes, there exist 2n-n different trees.

23. List out few of the Application of tree data-structure?

Answer: The list is as follows:

- The manipulation of Arithmetic expression,
- Symbol Table construction,
- Syntax analysis.

24. List out few of the applications that make use of Multilinked Structures?

Answer: The applications are listed below:

- Sparse matrix,
- Index generation.

25. In tree construction which is the suitable efficient data structure?

Answer: Linked list is the efficient data structure.

26. What is the type of the algorithm used in solving the 8 Queens problem?

Answer: Backtracking

27. In an AVL tree, at what condition the balancing is to be done?

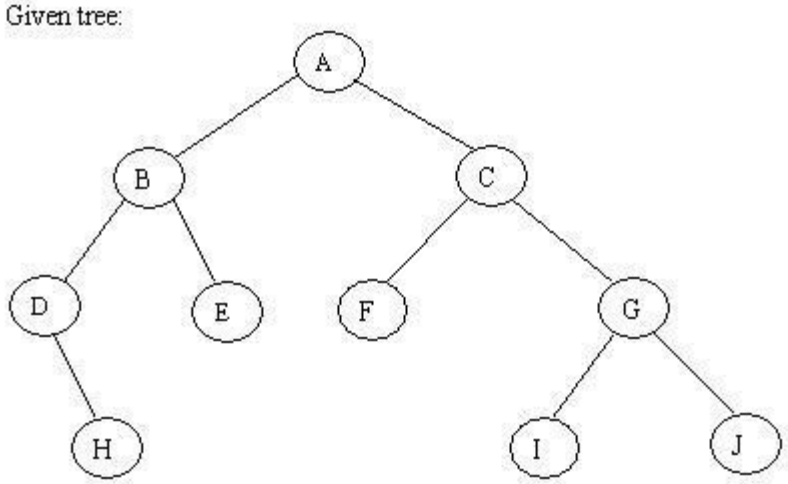
Answer: If the 'pivotal value' (or the 'Height factor') is greater than 1 or less than -1.

28. What is the bucket size, when the overlapping and collision occur at same time?

Answer: One. If there is only one entry possible in the bucket, when the collision occurs, there is no way to accommodate the colliding value. This results in the overlapping of values.

29. Traverse the given tree using Inorder, Preorder and Postorder traversals.

Given tree:



Answer:

- Inorder : D H B E A F C I G J
- Preorder: A B D H E C F G I J
- Postorder: H D E B F I J G C A

30. There are 8, 15, 13, 14 nodes were there in 4 different trees. Which of them could have formed a full binary tree?

Answer: 15.

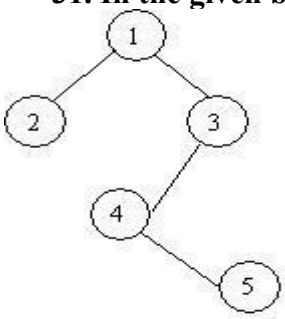
In general:

There are 2n-1 nodes in a full binary tree.

By the method of elimination:

Full binary trees contain odd number of nodes. So there cannot be full binary trees with 8 or 14 nodes, so rejected. With 13 nodes you can form a complete binary tree but not a full binary tree. So the correct answer is 15.

31. In the given binary tree, using array you can store the node 4 at which location?



Answer: At location 6

1	2	3	-	-	4	-	-	5

Root	LC1	RC1	LC2	RC2	LC3	RC3	LC4	RC4

whereLCn means Left Child of node n and RCn means Right Child of node n

32. Sort the given values using Quick Sort?

65	70	75	80	85	60	55	50	45

Answer:

Sorting takes place from the pivot value, which is the first value of the given elements, this is marked bold. The values at the left pointer and right pointer are indicated using L and R respectively.

65	70^{L}	75	80	85	60	55	50	45^{R}

Since pivot is not yet changed the same process is continued after interchanging the values at L and R positions

65	45	75^{L}	80	85	60	55	50^{R}	70
65	45	50	80^{L}	85	60	55^{R}	75	70
65	45	50	55	85^{L}	60^{R}	80	75	70
65	45	50	55	60^{R}	85^{L}	80	75	70

When the L and R pointers cross each other the pivot value is interchanged with the value at right pointer. If the pivot is changed it means that the pivot has occupied its original position in the sorted order (shown in bold italics) and hence two different arrays are formed, one from start of the original array to the pivot position-1 and the other from pivot position+1 to end.

60^{L}	45	50	55^{R}	65	85^{L}	80	75	70^{R}
$55^{L}5$	45	50^{R}	60	65	70^{R}	80^{L}	75	85
50^{L}	45^{R}	55	60	65	70	80^{L}	75^{R}	85

In the next pass we get the sorted form of the array.

45	50	55	60	65	70	75	80	85