EXPERIMENT No. 01: Implement a program in C on Frequency Histogram

1.1 Objective

Design and Implement a program in C on Frequency Histogram that builds a frequency array for data values in the range 1 to n and then prints their histogram.

The program should,

a. Read, Store and Print the data in an array.

b. Analyze the data in the array, one element at a time. Add 1 to the corresponding element in a frequency array based on the data value.

c. Print a histogram using asterisks for each occurrence of an element.

1.2 System Configuration

'C / C++' Programming Language and Linux / Windows as OS, NetBeans IDE 8.2

1.3 Pre-Requisite

Basic C Programming, Different Array operations

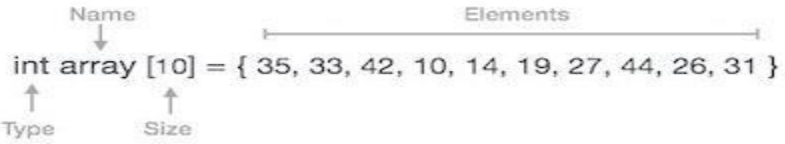
1.4 Introduction

Histograms are commonly found as a chart option in analyzing data in spreadsheet software, and in image editing software for showing the distribution of tones from black to white in an image. In the C programming language, using an array to hold the frequency count simplifies creating a histogram of your data set.

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of

each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).

Arrays can be declared in various ways in different languages. For illustration, let's take C array declaration.



Arrays can be declared in various ways in different languages. For illustration, let's take C array declaration.

Basic Operations:

- Creating an array of N Integer Elements: Creating an array by reading the value of N and N different integer elements using for loop.
- Display of array Elements with Suitable Headings: displaying the contents of array using a for loop.
- Inserting an Element (ELEM) at a given valid Position (POS): Insert operation is to insert one or more data elements into an array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array.
- Deleting an Element at a given valid Position (POS): Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

1.5 Procedure

```
#include<stdio.h>
int arr[20],n,fr[20],visited=-1;
void getdata();
void printdata();
void makefrequency();
void makehistogram ();
```

```
int main()
{
        getdata();
        printdata();
        makefrequency();
        makehistogram();
}
void getdata()
{
        int i;
        printf("Enter the number of elements:\n");
        scanf("%d",&n);
        printf("Enter the elements:\n");
        for(i=0;i<n;i++)
        scanf("%d",&arr[i]);
}
void printdata()
{
      int i;
        if(n==0)
             printf("Array is empty");
        else
        for(i=0;i<n;i++)
              printf("%d\t",arr[i]);
}
void makefrequency()
{
      int count,j;
      for(int i=0;i<n;i++)
      {
              count=1;
```

```
                for(j=i+1;j<n;j++)
                {
                    if(arr[i]==arr[j])
                    {
                        count++;
                        fr[j]=visited;
                    }
                }
                if(fr[i]!=visited)
                fr[i]=count;
        }
    }
  void makehistogram()
 {
          printf("\nElement  /  Frequency\n");
          printf("----------------------------\n");
          for(int i=0;i<n;i++)
          {
              if(fr[i]!=visited)
              printf("%d",arr[i]);
               printf(" | ");
              for(int j=0;j<fr[i];j++)
               printf(" * ");
                printf(" \n");
          }
}
```

1.6 Results

```
Enter the number of elements:
10
Enter the elements:
1 2 1 3 1 5 8 3 9 7

1    2    1    3    1    5    8    3    9    7
Element  /  Frequency
----------------------------
1 | * * *
2 | *
3 | * *
5 | *
8 | *
9 | *
7 | *
```

1.7 Prerequisite Questions

1. Explain for loop and while loop.
2. What are the difference between while and do-while loop.
3. Define Array.
4. Define Structure.

1.8 Post-Experimental Questions

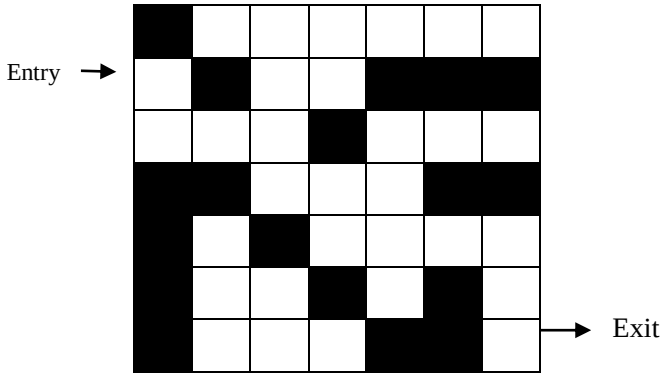
1. What are the different array operations.
2. Define pointer.
3. Explain different ways of initializing arrays.
4. Difference between structure and union.

EXPERIMENT No. 02: Implement a program in C that simulates a mouse in a maze

2.1 Objective

Design and Implement a program in C that simulates a mouse in a maze. The entrance spot, where the mouse starts its journey, is chosen by the user who runs the program. It can be changed each time.

The sample maze is represented below,



The program must print the path taken by the mouse from the starting point to the final point, including all spots that have been visited and backtracked.

2.2 Apparatus Required

'C / C++' Programming Language and Linux / Windows as OS, NetBeans IDE 8.2

2.3 Pre-Requisite

Basic C Programming, Different Array operations

2.4 Introduction

A Maze is given as N*N binary matrix of blocks where source block is the upper left most block i.e., maze[0][0] and destination block is lower rightmost block i.e., maze[N-1][N-1]. A rat starts from the source and has to reach the destination. The rat can move only in two directions: forward and down. In the maze matrix, 0 means the block is a dead end and 1 means the block can be used in the path from source to destination.

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

2.5 Procedure

```
#include <stdio.h>
#include <stdlib.h>
typedef struct
{
    short int vert;
    short int horiz;
}offsets;
offsets move[8];
typedef struct
{
    short int row;
    short int col;
    short int dir;
}Element;
Element stack[50];
int maze[5][5];
int mark[5][5];
```

```
int top=-1;
int exitrow,exitcol;
void readmaze()
{
    int r,c;
    int i,j;
    printf("enter the number of rows and cols:");
    scanf("%d%d",&r,&c);
    printf("enter the elements of the matrix:");
        for(i=0;i<r;i++)
        {
            for(j=0;j<c;j++)
            {
                scanf("%d",&maze[i][j]);
            }
        }
        exitrow=r-2;
        exitcol=c-2;
  }
  void setmovetable()
  {
        move[0].vert=-1,move[0].horiz=0;
        move[1].vert=-1,move[1].horiz=1;
        move[2].vert=0,move[2].horiz=1;
        move[3].vert=1,move[3].horiz=1;
        move[4].vert=1,move[4].horiz=0;
        move[5].vert=1,move[5].horiz=-1;
        move[6].vert=0,move[6].horiz=-1;
        move[7].vert=-1,move[7].horiz=-1;
  }
```

```
void push(Element ele)
{
        top=top+1;
        stack[top].row=ele.row;
        stack[top].col=ele.col;
        stack[top].dir=ele.dir;
        return;
}
Element pop()
{
        Element delitem;
        delitem.row=stack[top].row;
        delitem.col=stack[top].col;
        delitem.dir=stack[top].dir;
        top=top-1;
        return delitem;
}
void findpath()
{
         int i,row,col,nextrow,nextcol,dir,found=0;
        Element position;
        mark[1][1]=1,top=0;
        stack[0].row=1;
        stack[0].col=1;
        stack[0].dir=0;
        while(top>-1&&!found)
        {
              position=pop();
              row=position.row;
              col=position.col;
```

```
        dir=position.dir;
        while(dir<8&&!found)
        {
             nextrow=row+move[dir].vert;
            nextcol=col+move[dir].horiz;
             if(nextrow==exitrow&&nextcol==exitcol)
             found=1;
                  else if(!maze[nextrow][nextcol]&&!mark[nextrow][nextcol])
                 {
                          mark[nextrow][nextcol]=1;
                          position.row=row;
                          position.col=col;
                          position.dir=++dir;
                          push(position);
                          row=nextrow;
                          col=nextcol;
                          dir=0;
                 }
                      else
                      ++dir;
        }
}
if(found)
{
        printf("\n the path is :\n");
        printf("row\tcol\n");
        for(i=0;i<=top;i++)
        printf("%d%d\n",stack[i].row,stack[i].col);
        printf("%2d%5d\n",row,col);
        printf("%2d%5d",exitrow,exitcol);
```

```
        }
        else
        printf("\n No path");
}
void main()
{
        readmaze();
        setmovetable();
        findpath();
        return;
}
```

2.6 Results

```
Enter the number of rows and cols:
4   4
Enter the elements of the matrix:
1 1 1 1
1 0 0 1
1 0 0 1
1 1 1 1
The path is :
Row   Col
 1      1
 1      2
 2      2
Enter the number of rows and cols:
3   3
Enter the elements of the matrix:
1 1 1
1 0 1
1 1 1
No Path
```

2.7 Pre-Requisite Questions

1. Define Backtracking.
2. Explain how the stack is used as a data structure for the backtracking process.
3. Define stack.
4. Define recursion with an example.

2.8 Post-Experimental Questions

1. Explain stack applications.
2. Explain different operations of stack.
3. Define stack underflow with its condition to check stack underflow.
4. Define stack overflow with its condition to check stack overflow.

EXPERIMENT No. 03: Implement a Program in C for Stack Applications

3.1 Objective

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

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

b. Conversion of Arithmetic Expressions

3.2 Apparatus Required

'C / C++' Programming Language and Linux / Windows as OS, NetBeans IDE 8.2

3.3 Pre-Requisite

Basic C Programming, Stack Operations and its Applications, Converting infix to postfix

3.4 Introduction

An algebraic expression is a legal combination of operators and operands. Operand is the quantity on which a mathematical operation is performed. Operand may be a variable like x, y, z or a constant like 5, 4, 6 etc. Operator is a symbol which signifies a mathematical or logical operation between the operands. Examples of familiar operators include +, -, *, /, ^ etc. An algebraic expression can be represented using three different notations. They are infix, postfix and prefix notations.

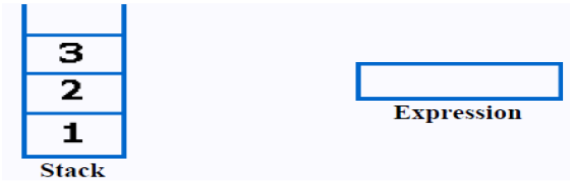
a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^

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

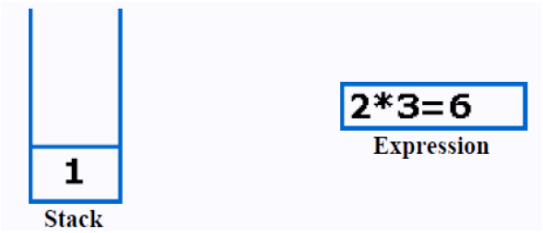
In normal algebra we use the infix notation like a+b*c. The corresponding postfix notation is abc*+

Consider the example: Postfix String: 123*+4-

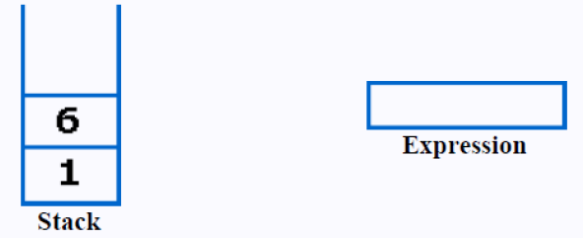
Initially the Stack is empty. Now, the first three characters scanned are 1,2 and 3, which are operands. Thus they will be pushed into the stack in that order.



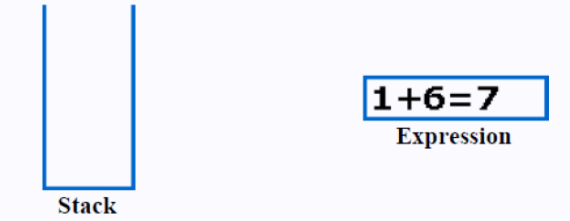
Next character scanned is "*", which is an operator. Thus, we pop the top two elements from the stack and perform the "*" operation with the two operands. The second operand will be the first element that is popped.



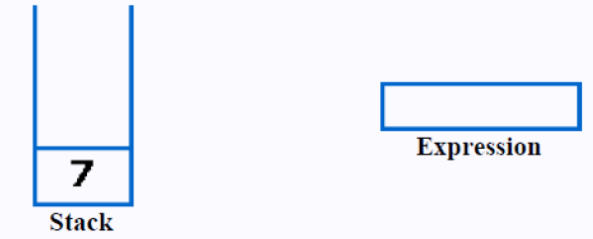
The value of the expression (2*3) that has been evaluated(6) is pushed into the stack.



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.

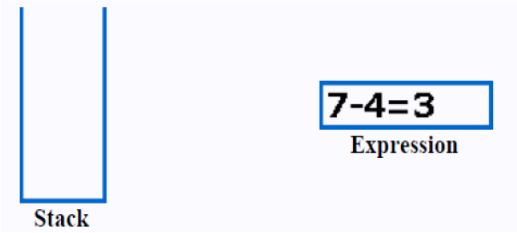


The value of the expression(1+6) that has been evaluated(7) is pushed into the stack.

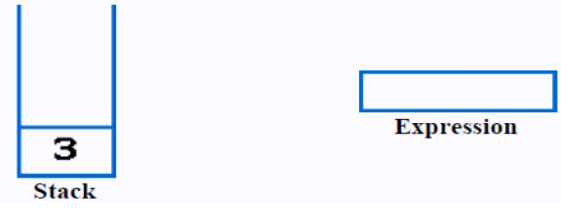


Next character scanned is "4", which is added to the stack.

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.



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



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

B. Conversion of Arithmetic Expressions

The three important features of postfix expression are:

1. The operands maintain the same order as in the equivalent infix expression.

2. The parentheses are not needed to designate the expression unambiguously.

3. While evaluating the postfix expression the priority of the operators is no longer relevant.

We consider five binary operations: +, -, *, / and $ or ↑ (exponentiation). For these binary operations, the following in the order of precedence (highest to lowest): Example: A B + C D - *

OPERATOR PRECEDENCE VALUE

OPERATOR	PRECEDENCE	VALUE
Exponentiation ($ or ↑ or ^)	Highest	4
*, /	Next highest	3
+, -	Lowest	2
#	Lowermost (endofexpn)	1

3.5 Procedure

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define MS 50
char postfix[50];
struct stack
{
        int top;
        int item[MS];
}s;
void push(int value)
{
        if(s.top==(MS-1))
        {
```

```
            printf("overflow\n");
        }
        else
        {
            s.item[++s.top]=value;
        }
}
int pop()
{
        if(s.top==-1)
        {
            printf("\n stack underflow");
            exit(0);
        }
        return(s.item[s.top--]);
}
int empty()
{
        if(s.top==-1)
        {
            return 1;
        }
        else
        {
            return 0;
        }
}
 int operation(int a,int b,char c)
{
        switch(c)
        {
```

```
            case '^':return(pow(a,b));
            case '*':return(a*b);
            case '%':return(a%b);
            case '/':return(a/b);
            case '+':return(a+b);
            case '-':return(a-b);
        }
}
int evaluate()
{
  int i,a,b,ans,value;
  char symb;
  for(i=0;postfix[i]!='\0';i++)
  {
        symb=postfix[i];
        if((symb>='0')&&(symb<='9'))
        {
            push((int)(symb-'0'));
        }
        else
        {
            a=pop();
            b=pop();
            value=operation(b,a,symb);
            push(value);
        }
  }
  ans=pop();
  return ans;
}
```

```
void main()
{
        s.top=-1;
        int ans;
        printf("Enter the postfix expression\n");
        gets(postfix);
        ans=evaluate();
        printf("The resultant ans is %d\n",ans);
}
```

B. Procedure:

```
#include<stdio.h>
#include<stdlib.h>
#define MS 5
char infix[50],postfix[50],item;
void convert();
struct stack
{
        int top;
        char item[MS];
}s;
void push(char value)
{
        if(s.top==(MS-1))
        {
             printf("The stack is overflow\n");
             exit(0);
        }

        else
        {
```

```
                s.item[++s.top]=value;
        }
}
char pop()
{
        if(s.top= =-1)
        {
            printf("stack underflow\n");
            exit(0);
        }
        return(s.item[s.top--]);
}
int empty()
{
        if(s.top= = -1)
        {
            return 1;
        }
        else
        {
            return 0;
        }
}
int precedence(char c)
{
        switch(c)
        {
            case '^':return 3;
            case '*':
            case '/':
            case '%':return 2;
```

```
            case '+':
            case '-':return 1;
            case '(':return 0;
        }
}
void main()
{
        s.top = -1;
        printf("Enter the infix expression :\n");
        gets(infix);
        convert();
        printf("The postfix expression is:\n");
        puts(postfix);
}
void convert()
{
        int i,pos=0;
        char symb,t;
        for(i=0;infix[i]!='\0';i++)
        {
            symb=infix[i];
            switch(symb)
            {
                case '(':push(symb);
                break;
                case ')':while((t=pop())!='(')
                {
                    postfix[pos++]=t;
                }
                break;
                case '^':
```

```
                case '*':
                case '/':
                case '%':
                case '+':
                case'-':
                while((!empty())&&((precedence(s.item[s.top]))>=precedence(symb)))
                    {
                        postfix[pos++]=pop();
                    }
                    push(symb);
                    break;
                    default: postfix[pos++]=symb;
                    break;
            }
        }
        while(!empty())
        {
            postfix[pos++]=pop();
        }
        postfix[pos]='\0';
}
```

3.6. A. Results

```
Enter the postfix expression
123+*321-+*
The resultant ans is 20
Enter the postfix expression
21+3-
The resultant ans is 0
Enter the postfix expression
42/
The resultant ans is 2
```

B. Result

Enter the infix expression :

(a+b)*c/d^e%f

The postfix expression is:

ab+c*de^/f%

3.7 Prerequisite Questions

1. Define infix expression.
2. Define postfix expression.
3. Give precedence hierarchy for C.
4. Which data structure is used to convert infix to postfix.

3.8 Post- Experimental Questions

1. Show the complete processing of postfix evaluation 6 2/3-4 2*+ using preferred data structure.
2. Evaluate postfix expression 4325*-+.
3. Which data structure is used to evaluate postfix expression.
4. Convert infix to postfix ((A + B) - C * (D / E)) + F.

EXPERIMENT No.04: Implement a program in C to categorize the data

4.1 Objectives:

Design and Implement a program in C to categorize the data. Consider the following sample list of numbers,

3	22	12	6	10	34	65	29	9	30	81	4	5	19	20	57	44	99

Categorize and sort them into different groups as mentioned below:

Group 1: Less than 10

Group 2: Between 10 and 19

Group 3: Between 20 and 29

Group 4: 30 and greater

4.2 Apparatus Required:

'C / C++' Programming Language and Linux / Windows as OS, NetBeans IDE 8.2

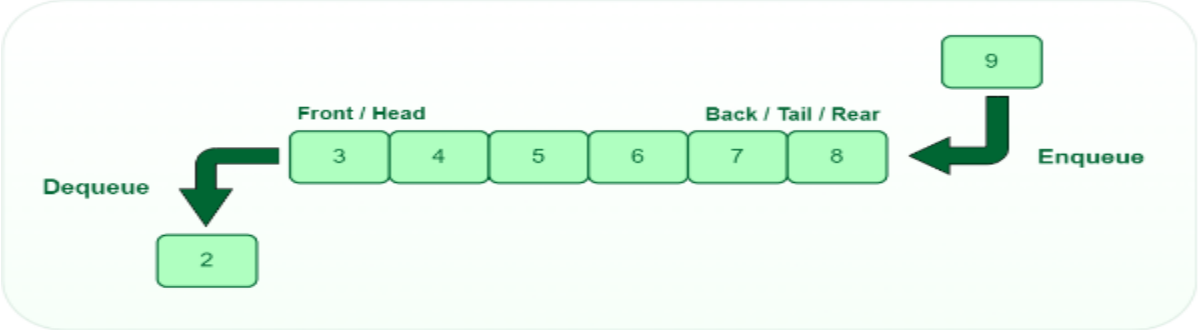
4.3 Pre-Requisite:

Basic C Programming, Different Queue operations

4.4 Introduction:

A queue in C is basically a linear data structure to store and manipulate the data elements. It follows the order of First In First Out (FIFO).

In queues, the first element entered into the array is the first element to be removed from the array.



Operations Associated with a Queue in C

A queue being an Abstract Data Structure provides the following operations for manipulation on the data elements:

- isEmpty(): To check if the queue is empty
- isFull(): To check whether the queue is full or not
- dequeue(): Removes the element from the frontal side of the queue
- enqueue(): It inserts elements to the end of the queue

Working of Queue Data Structure

Queue follows the First-In-First-Out pattern. The first element is the first to be pulled out from the list of elements.

- Front and Rear pointers keep the record of the first and last element in the queue.
- At first, need to initialize the queue by setting Front = -1 and Rear = -1
- In order to insert the element (enqueue), check whether the queue is already full i.e. check the condition for Overflow. If the queue is not full, increment the value of the Rear index by 1 and place the element at the position of the Rear pointer variable. When inserting the first element in the queue, set the value of Front to 0.
- In order to remove the element (dequeue) from the queue, check whether the queue is already empty i.e. check the condition for Underflow. If the queue is not empty, remove and return the element at the position of the Front pointer, and then increment the Front

index value by 1. To remove the last element from the queue, set the values of the Front and Rear index to -1.

4.5 Procedure:

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 20
typedef struct
{
      int front, rear;
       int item[MAXSIZE];
} queue;
     queue q1, q2, q3, q4;
int a[10], no=0, no2=0;
void insert(queue *q, int n)
{
 if(q->rear==MAXSIZE)
      {
           printf("Queue is full.");
      }
      else
      {
           q->item[++q->rear]=n;
      }
 }
 int delete(queue *q)
 {
      if(q->rear<q->front)
            printf("Queue is empty.");
      else
```

```
            return q->item[q->front++];
    }
    //Display from Queue
    void display(queue *q)
    {
        int i;
        if(q->rear<q->front)
        {
            printf("Queue is empty.");
        }
        else
        {
            no++;
            printf("Group %d: Contents of queue%d are ", no,no);
            for(i=q->front; i<=q->rear; i++)
            {
                printf("%d ", q->item[i]);
            }
        }
        printf("\n");
}
// Display the temp array
void displayQ(int a[])
{
        no2++;
        printf("Group %d: Contents of sorted array are ", no2);
        for(int i=0; a[i]!='\0';i++
        {
            printf("%d ", a[i]);
            a[i]='\0';
        }
```

```
            printf("\n");
}
// Insertion Sort Function
  void insertionSort()
  {
        for (int i = 1; a[i]!='\0'; i++)
        {
            int element = a[i];
            int j = i - 1;
            while (j >= 0 && a[j] > element)
            {
                a[j + 1] = a[j];
                j = j - 1;
            }
            a[j + 1] = element;
        }
  }
  void delete_sort(queue *q)
  {
        for(int i=0;q->item[i]!='\0';i++)
        {
            a[i]=delete(q);
        }
        insertionSort(a);
        displayQ(a);
  }
  void main()
  {
        q1.front=0,q2.front=0,q3.front=0,q4.front=0;
        q1.rear=-1,q2.rear=-1,q3.rear=-1,q4.rear=-1;
         int i,n,a[50];
```

```
printf("Enter the Elements");
scanf("%d",&n);
for(i=0;i<n;i++)
{
      scanf("%d",&a[i]);
}
// insert with grouping
for(i=0;i<n;i++)
{
      if(a[i]>=0 && a[i]<10)
      {
            insert(&q1, a[i]);
      }
      else if (a[i]>=10 && a[i]<20)
      {
            insert(&q2, a[i]);
      }
      else if (a[i]>=20 && a[i]<30)
      {
            insert(&q3, a[i]);
      }
      else if (a[i]>=30)
      {
            insert(&q4, a[i]);
       }
}
printf("Categorised data into different group:\n");
display(&q1);
display(&q2);
display(&q3);
display(&q4);
```

```
        printf("\nSorted data:\n");
        delete_sort(&q1);
        delete_sort(&q2);
        delete_sort(&q3);
        delete_sort(&q4);
    }
```

4.6 Results:

Enter the Elements 18

3 22 16 12 10 34 65 29 9 30 81 4 5 19 20 57 44 99

Categorised data into different group:

Group 1: Contents of queue1 are 3 9 4 5

Group 2: Contents of queue2 are 16 12 10 19

Group 3: Contents of queue3 are 22 29 20

Group 4: Contents of queue4 are 34 65 30 81 57 44 99

Sorted data:

Group 1: Contents of sorted array are 3 4 5 9

Group 2: Contents of sorted array are 10 12 16 19

Group 3: Contents of sorted array are 20 22 29

Group 4: Contents of sorted array are 30 34 44 57 65 81 99

4.7 Pre – Experimentation Questions

1. Drawbacks of stack.
2. Different types of data structures.
3. Define linear data structure.
4. Use of pointers used in a queue.

4.8 Post-Requisite Questions

1. Representation of Queue data structure and its operations.
2. Different types of queue.
3. Difference between stack and queue.
4. Difference between Linear Queue and Circular Queue.

EXPERIMENT No.05: Menu Driven Doubly Linked List

5.1 Objective

Design and Implement a menu driven program in C for the following operations on Doubly Linked List (DLL) of Student Data with the fields: USN, Name, Dept, Marks, PhNo

a. Create a DLL of N Students 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. Display the total and average marks for each student

5.2 Apparatus Required

'C / C++' Programming Language and Linux / Windows as OS, NetBeans IDE 8.2

5.3 Pre-Requisite

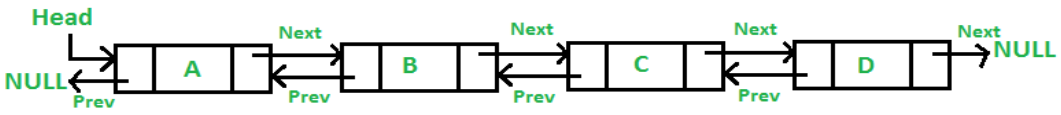
Basic C Programming, Basic of linked list

5.4 Introduction

A doubly linked list is a linked data structure that consists of a set of sequentially linked nodes. Each node contains two fields, called links that are references 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 NULL, to facilitate traversal of the list.

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, but the operations are simpler and 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.

Double-linked lists require more space per node, and their elementary operations are more expensive; but they are often easier to manipulate because they allow sequential access to the list in both directions. In particular, one can insert or delete a node in a constant number of operations given only that node's address. (Compared with singly- , which require the previous node's address in order to correctly insert or delete.) Some algorithms require access in both directions.



Advantages over singly linked list

1) A DLL can be traversed in both forward and backward direction.

2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given.

3) We can quickly insert a new node before a given node.

In a singly linked list, to delete a node, a pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using the previous pointer.

Disadvantages over singly linked list

1) Every node of DLL Require extra space for an previous pointer. It is possible to implement a DLL with a single pointer though.

2) All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers.

5.5 Procedure

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#define MS 5
```

```
int c;
struct student
{
        char usn[10],name[25],dept[12],phon[11];
        float marks1,marks2,marks3,total,average;
        struct student *next;
        struct student *prev;
};
typedef struct student node;
node *getnode(node *head)
{
        node *nn;
        nn=(node*)malloc(sizeof(node));
        printf("Enter Student details \n");
        printf("USN:");
        scanf("%s",nn->usn);
        printf("Name:");
        scanf("%s",nn->name);
        printf("Department:");
        scanf("%s",nn->dept);
        printf("Phone NO:");
        scanf("%s",nn->phon);
        printf("Enter the 3 subject Marks:\n");
        scanf("%f%f%f",&nn->marks1,&nn->marks2,&nn->marks3);
        nn->next=nn->prev=NULL;
        return nn;
}
int countnodes(node *head)
{
        node *p;
        p=head;
```

```
        c=0;
        while(p!=NULL)
        {
            p=p->next;
            c++;
        }
        return c;
}
node *create(node *head)
{
        node *nn,*p;
        p=head;
        if(head==NULL)
        {
            nn=getnode(head);
            head=nn;
        }
        else
        {
            nn=getnode(head);
            while(p->next!=NULL)
            {
                p=p->next;
            }
            p->next=nn;
            nn->prev=p;
        }
        return head;
}
```

```
node *insertfront(node *head)
{
        node *nn;
          if(countnodes(head)==MS)
          {
                printf("Insertion is not possible\n");
          }
          else
          {
                nn=getnode(head);
                if(head==NULL)
                {
                      head=nn;
                }
                else
                {
                      nn->next=head;
                      head->prev=nn;
                      head=nn;
                }
          }
          return head;
}
node *insertrear(node *head)
{
          node *nn;
          if(countnodes(head)==MS)
          {
                printf("Insertion is not possible\n");
          }
          else
```

```
        {
            head=create(head);
        }
        return head;
 }
 node *display(node *head)
 {
        node *p;
        if(head==NULL)
        {
            printf("No data\n");
        }
        else
        {
            p=head;
            printf("USN\tNAME\tDEPARTMENT\tPhoneNO\t\tMARKS1\tMARKS2\tMARKS3\t
            TOTAL\tAVERAGE\n");
            while(p!=NULL)
            {
                    p->total=p->marks1+p->marks2+p->marks3;
                    p->average=p->total/3;
                    printf("%s\t%s\t%s\t%s\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n",p->usn,p->name,
                     p->dept,p->phon,p->marks1,p->marks2,p->marks3,p->total,p->average);
                    p=p->next;
            }
        }
        printf("The number of nodes in list is %d\n",countnodes(head));
        return head;
}
```

```
node *deletefront(node *head)
{
        node *p;
        if(head==NULL)
        {
            printf("No data\n");
        }
        else if(countnodes(head)==1)
        {
            p=head;
            head=NULL;
            free(p);
        }
        else
        {
            p=head;
            (head->next)->prev=NULL;
            head=head->next;
            p->next=NULL;
            free(p);
        }
        return head;
}
node *deleterear(node *head)
{
   node *p,*q;
   if(head==NULL)
   {
        printf("No data\n");
   }
```

```
    else if(countnodes(head)==1)
    {
        p=head;
        head=NULL;
        free(p);
    }
    else
    {
        p=head;
        while(p->next!=NULL)
        {
            p=p->next;
        }
        q=p->prev;
        q->next=NULL;
        p->prev=NULL;
        free(p);
    }
    return head;
}
void main()
{
        int ch,i,n;
        node *head;
        head=NULL;
        do
        {
                printf("\n\t*....Student Data.....*");
                printf("\n1.Create\n2.Display\n3.Insert_Front\n4.Insert_Rear\n5.Delete_Front\n
                6.Delete_Rear\n7.Exit\n");
                printf("Enter your choice\n");
```

```
            scanf("%d",&ch);
            switch(ch)
            {
                case 1:printf("Enter number of  Student\n");
                        scanf("%d",&n);
                        for(i=0;i<n;i++)
                        head=create(head);
                        break;
                case 2:head=display(head);
                        break;
                case 3:head=insertfront(head);
                        break;
                case 4:head=insertrear(head);
                        break;
                case 5: head=deletefront(head);
                        break;
                case 6:head=deleterear(head);
                        break;
                case 7:exit(0);
                default:printf("Invalid choice\n");
            }
    }while(ch>=1&&ch<=6);
}
```

5.6 Results

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

1

Enter number of Students

2

Enter employee details

USN: 4SF01

Name: ARUN

Department: ISE

Phone NO: 9887654567

Enter the 3 subject Marks:

90 90 90

Enter employee details

USN: 4SF02

Name: BARUN

Department: CSE

Phone NO: 8776543456

Enter the 3 subject Marks:

75 75 75

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

2

USN	NAME	DEPARTMENT	PhoneNO	MARKS1	MARKS2	MARKS3	TOTAL	AVERAGE
4SF01	ARUN	ISE	9887654567	90.00	90.00	90.00	270.00	90.00
4SF02	BARUN	CSE	8776543456	75.00	75.00	75.00	225.00	75.00

The number of nodes in list is 2

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

3

Enter employee details

USN: 4SF03

Name: DHABU

Department: ME

Phone NO: 988765423

Enter the 3 subject Marks:

75 75 75

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

4

Enter employee details

USN: 4SF04

Name: SHANU

Department: CSE

Phone NO: 9887653456

Enter the 3 subject Marks:

90 96 95

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

2

USN	NAME	DEPARTMENT	PhoneNO	MARKS1	MARKS2	MARKS3	TOTAL	AVERAGE
4SF03	DHABU	ME	988765423	75.00	75.00	75.00	225.00	75.00
4SF01	ARUN	ISE	9887654567	90.00	90.00	90.00	270.00	90.00
4SF02	BARUN	CSE	8776543456	75.00	75.00	75.00	225.00	75.00
4SF04	SHANU	CSE	9887653456	90.00	96.00	95.00	281.00	93.67

The number of nodes in list is 4

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

5

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

2

USN	NAME	DEPARTMENT	PhoneNO	MARKS1	MARKS2	MARKS3	TOTAL	AVERAGE
4SF01	ARUN	ISE	9887654567	90.00	90.00	90.00	270.00	90.00
4SF02	BARUN	CSE	8776543456	75.00	75.00	75.00	225.00	75.00
4SF04	SHANU	CSE	9887653456	90.00	96.00	95.00	281.00	93.67

The number of nodes in list is 3

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

6

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

2

USN	NAME	DEPARTMENT	PhoneNO	MARKS1	MARKS2	MARKS3	TOTAL	AVERAGE
4SF01	ARUN	ISE	9887654567	90.00	90.00	90.00	270.00	90.00
4SF02	BARUN	CSE	8776543456	75.00	75.00	75.00	225.00	75.00

The number of nodes in list is 2

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

5

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

6

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exi

t

Enter your choice

5

No data

....Student Data.....

1. Create
2. Display
3. Insert_Front
4. Insert_Rear
5. Delete_Front
6. Delete_Rear
7. Exit

Enter your choice

7

5.7 Prerequisite Questions

1. Define doubly linked list and purpose of Linked list.
2. Write the structure of a doubly linked list.
3. Advantages of doubly linked lists over singly linked lists.
4. Overview of malloc and its usage.

5.8 Post- Experimental Questions

1. Each node has a _____number of pointers to traverse the list back and forth.
2. countnodes(head) method returns _____.
3. if(head==NULL) it means that _____.
4. ((p->next)->next) will return _____.

EXPERIMENT No.06: Implement a Program for Singly Circular Linked List (SCLL)

6.1 Objective

Design and Implement a program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes,

a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z - 4yz^5 + 3x^3yz + 2xy^5z - 2xyz^3$.

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).

6.2 Apparatus Required

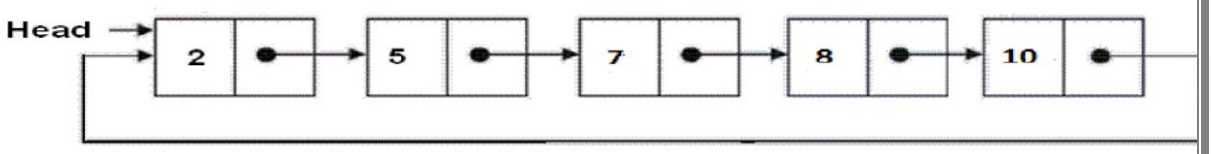
'C / C++' Programming Language and Linux / Windows as OS, NetBeans IDE 8.2

6.3 Pre-Requisite

Basic C Programming, Basic of Linked List, Circular linked list

6.4 Introduction

In a circularly-linked list, the first and final nodes are linked together. This can be done for both singly and doubly linked lists. To traverse a circular linked list, you begin at any node and follow the list in either direction until you return to the original node. Note that a circular list does not have a natural "first" or a "last" node. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.



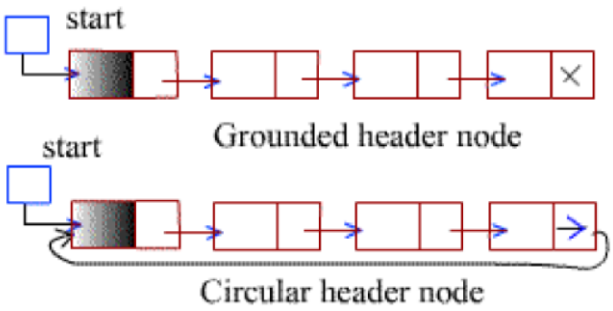
Advantages of Circular Linked Lists:

1) Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.

2) Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.

3) Multiplayer board games can be implemented using circular lists where each player waits for his turn to play in a circular fashion.

Header Linked List:



A header linked list is a linked list which always contains a special node called the header node at the beginning of the list. It is an extra node kept at the front of a list. Such a node does not represent an item in the list. The information portion might be unused.

There are two types of header list

1. Grounded header list: is a header list where the last node contains the null pointer.
2. Circular header list: is a header list where the last node points back to the header node.

Circular Singly Linked Lists:

In a singly linked circular list, the pointer field of the last node stores the address of the starting node.

Adding two polynomials are represented using linked lists:

Representation of a Polynomial: A polynomial is an expression that contains more than two terms. A term is made up of coefficient and exponent.

An example of polynomial is $P(x) = 4x^3+6x^2+7x+9$

A polynomial thus may be represented using arrays or linked lists. Array representation assumes that the exponents of the given expression are arranged from 0 to the highest value (degree), which is represented by the subscript of the array beginning with 0. The coefficients of the respective exponent are placed at an appropriate index in the array.

A polynomial may also be represented using a linked list. A structure may be defined such that it contains two parts- one is the coefficient and second is the corresponding exponent. The structure definition may be given as shown below:

```
struct polynomial
{
int coefficient;
int exponent;
struct polynomial *next;
};
```

Addition of two Polynomials:

For adding two polynomials using arrays is straightforward method, since both the arrays may be added up element wise beginning from 0 to n-1, resulting in addition of two polynomials. Addition of two polynomials using linked list requires comparing the exponents, and wherever the exponents are found to be same, the coefficients are added up. For terms with different exponents, the complete term is simply added to the result thereby making it a part of addition result.

6.5 Procedure

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

```
#include<stdlib.h>
struct poly
{
    int cf,px,py,pz;
    int flag;
    struct poly *next;
};
typedef struct poly node;
node* getnode()
{
    node *nn;
    nn=(node*)malloc(sizeof(node));
    if(nn==NULL)
    {
        printf("Insufficient memory\n");
        exit(0);
    }
    return nn;
}
void display(node *head)
{
    node *p;
    if(head->next==head)
    {
        printf("Polynomial does not exist\n");
        return;
    }
    p=head->next;
    while(p!=head)
    {
        printf("%dx^%dy^%dz^%d",p->cf,p->px,p->py,p->pz);
```

```
            if(p->next!= head)
            printf(" + ");
            p=p->next;
        }
}
node* insert_rear(int cf,int x,int y,int z,node *head)
{
        node *p,*v;
        p=getnode();
        p->cf=cf;
        p->px=x;
        p->py=y;
        p->pz=z;
        v=head->next;
        while(v->next!=head)
        {
            v=v->next;
        }
        v->next=p;
        p->next=head;
        return head;
}
node* read_poly(node *head)
{
        int px, py, pz, cf, ch;
        do
        {
            printf("Enter coeff: ");
            scanf("%d",&cf);
            printf("Enter powers of x,y,z\n ");
            scanf("%d%d%d",&px,&py,&pz);
```

```
            head=insert_rear(cf,px,py,pz,head);
            printf("If your wish to continue press 1 otherwise 0\n");
            scanf("%d", &ch);
        }while(ch != 0);
        return head;
}
node* add_poly(node *h1,node *h2,node *h3)
{
        node *p1,*p2;
        int x1,x2,y1,y2,z1,z2,cf1,cf2,cf;
        p1=h1->next;
        while(p1!=h1)
        {
            x1=p1->px;
            y1=p1->py;
            z1=p1->pz;
            cf1=p1->cf;
            p2=h2->next;
            while(p2!=h2)
            {
                x2=p2->px;
                y2=p2->py;
                z2=p2->pz;
                cf2=p2->cf;
                if(x1==x2 && y1==y2 && z1==z2)break;
                p2=p2->next;
            }
            if(p2!=h2)
            {
                cf=cf1+cf2;
                p2->flag=1;
```

```
            if(cf!=0)
                h3=insert_rear(cf,x1,y1,z1,h3);
        }
        else
            h3=insert_rear(cf1,x1,y1,z1,h3);
            p1=p1->next;
    }
    p2=h2->next;
    while(p2!=h2)
    {
        if(p2->flag==0)
        h3=insert_rear(p2->cf,p2->px,p2->py,p2->pz,h3);
        p2=p2->next;
    }
    return h3;
}
void evaluate(node *head)
{
    node *p;
    int x, y, z;
    int result=0;
    p=head->next;
    printf("\nEnter x,y,z terms to evaluate:\n");
    scanf("%d%d%d",&x,&y,&z);
    while(p!= head)
    {
        result = result + (p->cf * pow(x,p->px) * pow(y,p->py) * pow(z,p->pz));
        p=p->next;
    }
    printf("Polynomial result is: %d", result);
}
```

```
void main()
{
    node *h1,*h2,*h3;
    int ch;
    h1=getnode();
    h2=getnode();
    h3=getnode();
    h1->next=h1;
    h2->next=h2;
    h3->next=h3;
    while(1)
    {
        printf("\n\n1.Evaluate polynomial\n2.Add two polynomials\n3.Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: h1->next=h1;
            printf("\nEnter polynomial to evaluate:\n");
            h1=read_poly(h1);
            printf("The polynomial is :");
            display(h1);
            valuate(h1);
            break;
            case 2: h1->next=h1;
            printf("\nEnter the first polynomial:\n");
            h1=read_poly(h1);
            printf("\nEnter the second polynomial:\n");
            h2=read_poly(h2);
            h3=add_poly(h1,h2,h3);
            printf("\nFirst polynomial is: ");
```

```
                display(h1);
                printf("\nSecond polynomial is: ");
                display(h2);
                printf("\nThe sum of 2 polynomials is: \n");
                display(h3);
                case 3: exit(0);
                default:printf("\nInvalid entry");
                break;
            }
        }
}
```

6.6 Results

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

Enter your choice: 1

Enter polynomial to evaluate:

Enter coeff: 6

Enter powers of x,y,z

2 2 1

If your wish to continue press 1 otherwise 0

1

Enter coeff: -4

Enter powers of x,y,z

0 1 5

If your wish to continue press 1 otherwise 0

1

Enter coeff: 3

Enter powers of x,y,z

```
3 1 1
If your wish to continue press 1 otherwise 0
1
Enter coeff: 2
Enter powers of x,y,z
1 5 1
If your wish to continue press 1 otherwise 0
1
Enter coeff: -2
Enter powers of x,y,z
1 1 1
If your wish to continue press 1 otherwise 0
0
The polynomial is :6x^2y^2z^1 + -4x^0y^1z^5 + 3x^3y^1z^1 + 2x^1y^5z^1 + -2x^1y^1z^1
Enter x,y,z terms to evaluate:
1 1 1
Polynomial result is: 5

1. Evaluate polynomial
2. Add two polynomials
3. Exit
Enter your choice: 2
Enter the first polynomial:
Enter coeff: 4
Enter powers of x,y,z
2 2 2
If your wish to continue press 1 otherwise 0
1
Enter coeff: 3
Enter powers of x,y,z
1 1 2
```

If your wish to continue press 1 otherwise 0

0

Enter the second polynomial:

Enter coeff: 6

Enter powers of x,y,z

2 2 2

If your wish to continue press 1 otherwise 0

0

First polynomial is: 4x^2y^2z^2 + 3x^1y^1z^2

Second polynomial is: 6x^2y^2z^2

The sum of 2 polynomials is:

10x^2y^2z^2 + 3x^1y^1z^2

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

Enter your choice: 3

6.7 Prerequisite Questions

1. Define Circular Linked List.
2. List the advantages of circular linked list.
3. Define polynomials and their representation.
4. How to find the degree of the polynomial.

6.8 Post- Experimental Questions

1. How to add two Polynomials.
2. Define sparse polynomials.
3. How to represent zero polynomials using Circular list representation with header nodes.
4. How to represent $3x^{14} + 2x^{8} + 1$ polynomial using Circular list representation with header nodes.

EXPERIMENT No.07: Menu driven Program in C on Binary Search Tree (BST) of Integers

7.1 Objective

Design and Implement a program in C that reads a list of names and telephone numbers to inserts them into a Binary Search Tree for the following operations,

a. Search the list for a specified name.

b. Insert a new name.

c. Delete an existing name.

d. Traverse the phone list using Inorder, Preorder and Postorder.

7.2 Apparatus Required

'C / C++' Programming Language and Linux / Windows as OS, NetBeans IDE 8.2

7.3 Pre-Requisite

Basic C Programming, Basic of trees, Binary search tree

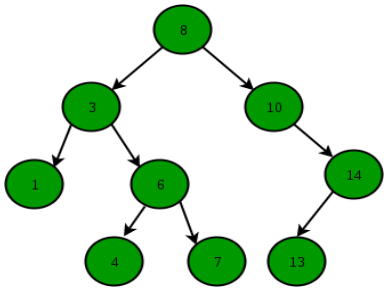
7.4 Introduction

A Binary Search Tree(BST) is a special type of binary tree data structure that has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

Representation

BST is a collection of nodes arranged in a way where they maintain BST properties. Each node has key and associated value. While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved.



Node: A node having some data, references to its left and right child nodes.

```
struct node
{
    int data;
    struct node *leftChild;
    struct node *rightChild;
};
```

Basic Operations on BST:

Insert a value in a BST: A new key is always inserted at the leaf by maintaining the property of the binary search tree. We start searching for a key from the root until we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.

Search a key in a BST: Whenever an element is to be search. Start search from root node then if data is less than key value, search element in left subtree otherwise search element in right subtree.

Deleting a key from a BST: When we delete a node, three possibilities arise.

1) Node to be deleted is the leaf: Simply remove it from the tree.

2) Node to be deleted has only one child: Copy the child to the node and delete the child

3) Node to be deleted has two children: Find inorder successor of the node. Copy contents of the inorder successor/Inorder predecessor to the node and delete the inorder successor.

Note: Inorder successor is needed only when the right child is not empty. In this particular case, in-order successor can be obtained by finding the minimum value in the right child of the node.

Tree Traversal:

Inorder Traversal Algorithm:

1. Traverse the left subtree, i.e., call Inorder(left->subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right->subtree)

Preorder Traversal Algorithm:

1. Visit the root.
2. Traverse the left subtree, i.e., call Inorder(left->subtree)
3. Traverse the right subtree, i.e., call Inorder(right->subtree)

Postorder Traversal Algorithm:

1. Traverse the left subtree, i.e., call Inorder(left->subtree)
2. Traverse the right subtree, i.e., call Inorder(right->subtree)
3. Visit the root.

7.5 Procedure

```
# include<stdio.h>
# include<stdlib.h>
#include<string.h>
struct node
{
    char name[25],phno[15];
    struct node* lc, *rc;
};
typedef struct node *NODE;
int delflag;
NODE createnode()
{
    NODE temp;
```

```
    temp = (NODE)malloc(sizeof(struct node));
    printf("Enter the name:");
    scanf("%s",temp->name);
    printf("Enter the phone number:");
    scanf("%s",temp->phno);
    temp->lc = NULL;
    temp->rc = NULL;
    return temp;
}
void insertBST(NODE root, NODE newNode)
{
    if(strcmp(newNode->name,root->name)==0)
    {
        printf("Key already exists\n");
        return;
    }
    else if (strcmp(newNode->name,root->name)<0)
    {
        if (root->lc == NULL)
        root->lc = newNode;
        else
        insertBST(root->lc, newNode);
    }
    else
    {
        if(root->rc == NULL)
        root->rc = newNode;
        else
         insertBST(root->rc, newNode);
    }
}
```

```
int search(NODE root, char keyname[])
{
    if(!root)
    return -1;
    if(strcmp(keyname,root->name)==0)
     return 1;
    else if(strcmp(keyname,root->name)<0)
    return search(root->lc, keyname);
    else
    return search(root->rc,keyname);
}

NODE getRightMin(NODE root)
{
    NODE temp = root;
    while(temp->lc != NULL)
    {
        temp = temp->lc;
    }
    return temp;
}
NODE deleteBST(NODE root, char keyname[])
{
    if(!root)
    {
        delflag=-1;
        return NULL;
    }
     if(strcmp(keyname,root->name)<0)
    root->lc=deleteBST(root->lc, keyname);
    else if(strcmp(keyname,root->name)>0)
```

```
        root->rc=deleteBST(root->rc,keyname);
        else
        {
            if(root->lc==NULL && root->rc==NULL)
            {
                free(root);
                return NULL;
            }
            else if(root->lc == NULL)
            {
                NODE temp = root->rc;
                free(root);
                return temp;
            }
            else if(root->rc == NULL)
            {
                NODE temp = root->lc;
                free(root);
                return temp;
            }
            else
            {
                NODE rightMin = getRightMin(root->rc);
                strcpy(root->name,rightMin->name);
                strcpy(root->phno,rightMin->phno);
                root->rc = deleteBST(root->rc,rightMin->name);
            }
        }
        return root;
    }
```

```
void inorder(NODE temp)
{
    if (temp != NULL)
    {
        inorder(temp->lc);
        printf("|%s|%s|\t", temp->name,temp->phno);
        inorder(temp->rc);
    }
}
void preorder(NODE temp)
{
    if (temp != NULL)
    {
         printf("|%s|%s|\t", temp->name,temp->phno);
        preorder(temp->lc);
        preorder(temp->rc);
    }
}
void postorder(NODE temp)
{
    if (temp != NULL)
    {
        postorder(temp->lc);
        postorder(temp->rc);
        printf("|%s|%s|\t", temp->name,temp->phno);
    }
}
void main()
{
    int choice,n,i,keyFound = 0;
    char keyname[25];
```

```
NODE root=NULL,newNode;
printf("--------------------Creating a BST------------------\n");
printf("Enter the number of records in the BST:");
scanf("%d",&n);
 for(i=0;i<n;i++)
{
      newNode = createnode();
      if(root == NULL)
       root = newNode;
      else
      insertBST(root,newNode);
}
while(1)
{
      choice=0;
      printf("\n-----------------MENU----------------------\n");
       printf("1. Search a list for a specified name\n");
      printf("2. Insert a new name\n");
      printf("3. Deleting existing name\n");
      printf("4. Traverse the phone list\n");
      printf("5. Exit\n");
      printf("-----------------------------------------\n");
      printf("Enter choice : ");
      scanf("%d", &choice);
      switch(choice)
      {
            case 1: printf("Enter the name to be searched:");
            scanf("%s",keyname);
            keyFound = search(root,keyname);
            if(keyFound == 1)
            printf("Name: %s is found in the BST",keyname);
```

```
else
printf("Name: %s is not found in the BST",keyname);
 break;
case 2: newNode = createnode();
if(root == NULL)
root = newNode;
else
insertBST(root,newNode);
break;
case 3: if(root == NULL)
{
      printf("Tree is empty\n");
}
else
{
      delflag=0;
      printf("Enter the name to be deleted:");
      scanf("%s",keyname);
      root=deleteBST(root,keyname);
      if(delflag==-1)
      printf("Name: %s is not found in the BST\n",keyname);
      else
      printf("Name: %s is deleted from the BST\n",keyname);
}
break;
case 4: if (root == NULL)
{
      printf("Tree is empty\n");
 }
else
{
```

```
                    printf("BST Preorder travsersal\n");
                    preorder(root);
                    printf("\nBST Inorder travsersal\n");
                    inorder(root);
                    printf("\nBST Postorder travsersal\n");
                    postorder(root);
                }
                break;
                case 5: return;
                default: printf("Wrong choice\n");
                return;
            }
        }
    }
```

7 .6 Results

```
----------Creating a BST----------
Enter the number of records in the BST: 5
Enter the name:Arun
Enter the phone number: 9665438978
Enter the name:Bhavya
Enter the phone number: 9778654567
Enter the name:Ramya
Enter the phone number: 9887652345
Enter the name:Kavya
Enter the phone number: 9876523451
Enter the name:Divya
Enter the phone number: 987987654 1

----------MENU----------
```

1. Search a list for a specified name

2. Insert a new name

3. Deleting existing name

4. Traverse the phone list

5. Exit

Enter choice: 4

BST Preorder travsersal

|Arun|9665438978| |Bhavya|9778654567| |Ramya|9887652345| |Kavya|9876523451| |Divya|9879876541|

BST Inorder travsersal

|Arun|9665438978| |Bhavya|9778654567| |Divya|9879876541| |Kavya|9876523451| |Ramya|9887652345|

BST Postorder travsersal

|Divya|9879876541| |Kavya|987652345| |Ramya|9887652345| Bhavya|9778654567| |Arun|9665438978|

----------MENU----------

1. Search a list for a specified name

2. Insert a new name

3. Deleting existing name

4. Traverse the phone list

5. Exit

Enter choice : 1

Enter the name to be searched:Arun

Name: Arun is found in the BST

----------MENU----------

1. Search a list for a specified name

2. Insert a new name

3. Deleting existing name

4. Traverse the phone list

5. Exit

Enter choice : 1

Enter the name to be searched:Kavya

Name: Kavya is found in the BST

----------MENU----------

1. Search a list for a specified name

2. Insert a new name

3. Deleting existing name

4. Traverse the phone list

5. Exit

Enter choice : 3

Enter the name to be deleted:Divya

Name: Divya is deleted from the BST

----------MENU----------

1. Search a list for a specified name

2. Insert a new name

3. Deleting existing name

4. Traverse the phone list

5. Exit

Enter choice : 4

BST Preorder travsersal

|Arun|9665438978| |Bhavya|9778654567| |Ramya|9887652345| |Kavya|9876523451|

BST Inorder travsersal

|Arun|9665438978| |Bhavya|9778654567| |Kavya|9876523451| |Ramya|9887652345|

BST Postorder travsersal

|Kavya|9876523451| |Ramya|9887652345| |Bhavya|9778654567| |Arun|9665438978|

----------MENU----------

1. Search a list for a specified name

2. Insert a new name

3. Deleting existing name

4. Traverse the phone list

5. Exit

Enter choice : 2

Enter the name:Divya

Enter the phone number: 9878764535

----------MENU----------

1. Search a list for a specified name

2. Insert a new name

3. Deleting existing name

4. Traverse the phone list

5. Exit

Enter choice: 4

BST Preorder travsersal

|Arun|9665438978| |Bhavya|9778654567| |Ramya|9887652345| |Kavya|9876523451| |Divya|9878764535|

BST Inorder travsersal

|Arun|9665438978| |Bhavya|9778654567| |Divya|9878764535| |Kavya|987652345| |Ramya|9887652345|

BST Postorder travsersal

|Divya|9878764535| |Kavya|9876523451| |Ramya|9887652345| |Bhavya|9778654567| |Arun|9665438978|

----------MENU----------

1. Search a list for a specified name
2. Insert a new name
3. Deleting existing name
4. Traverse the phone list
5. Exit

Enter choice : 5

7.7 Pre-Requisite Questions

1. Define Binary tree.
2. Define complete binary tree and skewed binary tree.
3. List out the properties of binary trees.
4. Explain left child right sibling representation?

7.8 Post- Experimental Questions

1. Explain linked representation of binary tree.
2. A full binary tree of depth k is a binary tree of depth k having _____ nodes.
3. Explain different types of traversal.
4. Define threaded binary tree with its advantages.

EXPERIMENT No.08: Implement a Program in C for the operations on Graph(G)

8.1 Objective

A company has seven top officers working for it. They are each fluent in atleast one language according to the following sample table,

Officer	Hindi	Malayalam	Kannada	Telugu
01	-	-	Y	-
02	-	-	Y	Y
03	-	-	-	Y
04	-	Y	-	Y
05	Y	Y	-	-
06	Y	-	Y	-
07	-	Y	-	-

Design and Implement a program in C for the following operations on Graphs (G),

a. Create a graph using adjacency matrix indicating people who can communicate directly with each other.

b. Print all the officers which are reachable from a given officer as a starting node in a digraph.

Example: An officer wants to send a message to each other officer: A message comes to an officer, he reads it and transmits it to another officer possibly after translation to someone who has not read it.

8.2 Apparatus Required

'C / C++' Programming Language and Linux / Windows as OS, NetBeans IDE 8.2

8.3 Pre-Requisite

Basic C Programming, Graphs, DFS/BFS

8.4 Introduction

A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices(V) and a set of edges(E). The graph is denoted by G(E, V).

The following two are the most commonly used representations of a graph.

1. Adjacency Matrix

2. Adjacency List

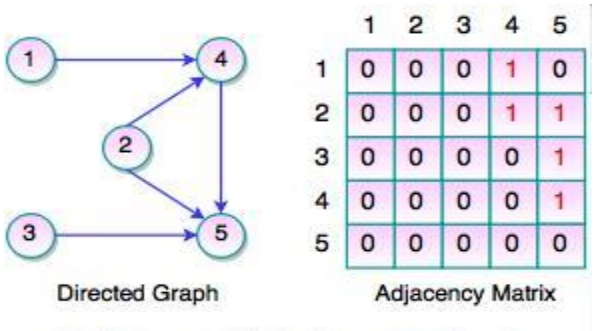
Adjacency Matrix:

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric.

In case of an undirected graph, we need to show that there is an edge from vertex i to vertex j and vice versa. In code, we assign adj[i][j] = 1 and adj[j][i] = 1

In case of a directed graph, if there is an edge from vertex i to vertex j then we just assign adj[i][j]=1

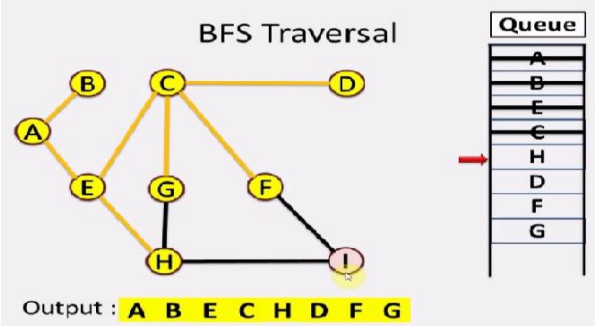
An example of adjacency matrix representation of an directed graph is given below:



Breadth-first search (BFS)

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores the neighbour nodes first, before moving to the next level neighbours.

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.



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

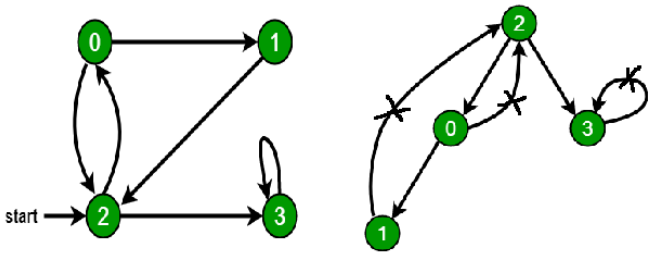
Depth-first search (DFS)

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.

Initially all vertices are unvisited. DFS starts in arbitrary vertex and runs as follows:

- Mark vertex u as visited.
- For each edge (u, v), where u is white, run depth-first search for u recursively.
- After finishing the processing of vertex u, backtrack to the parent.

For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Depth First Traversal of the following graph is 2, 0, 1, 3.



8.5 Procedure

```
#include<stdio.h>
#include<stdlib.h>
int st[10],top=-1,v[10],a[10][10],u[10];
int n,q[10],front=0;
int rear=-1;
void dfs(int s)
{
    int i;
    v[s]=1;
    st[++top]=s;
    for(i=1;i<=n;i++)
    {
        if(a[s][i]==1&&v[i]==0)
        {
            printf("Officer %d->Officer %d\n",s,i);
            dfs(i);
        }
    }
}

void bfs(int s)
{
```

```
        int m,i;
        u[s]=1;
        q[++rear]=s;
        printf("Reachable officers using BFS method from a given officer : %d are\n",s);
        while(front<=rear)
        {
                m=q[front++];
                for(i=1;i<=n;i++)
                {
                        if(a[m][i]==1&&u[i]==0)
                        {
                                q[++rear]=i;
                                printf("Officer %d\n",i);
                                u[i]=1;
                        }
                }
        }
}

void main()
{
        int s,i,j,ch;
        while(1)
        {
                printf("1.Create a graph using adjacency matrix indicating people who can
                communicate directly with each other\n2.DFS traversal method through which
                any officer can be reachable from a given node\n3.BFS traversal method through
                which any officer can be reachable from a given node\n4.Exit\n");
                printf("Enter the choice\n");
                scanf("%d",&ch);
                switch(ch)
```

```
{
        case 1: printf("Enter the number of officers\n");
        scanf("%d",&n);
        printf("Enter the adjacency matrix representation\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        break;
        case 2:printf("Depth First Search Traversal\n");
        printf("Enter Source Officer\n");
        scanf("%d",&s);
        printf("Reachable officers using DFS method from a given officer:
        %d\n",s);
        dfs(s);
        for(i=1;i<=n;i++)
        {
                if(v[i]==0)
        {
             printf("%d is not visited and it is disconnected graph\n",i);
        }
}
break;
case 3: printf("Breadth First Search Traversal\n");
printf("Enter Source Officer\n");
scanf("%d",&s);
bfs(s);
for(i=1;i<=n;i++)
```

```
            {
                if(u[i]==0)
                {
                    printf("Officer %d is not visited and that officer is
                    disconnected\n",i);
                }
            }
            break;
            case 4:exit(0);
            default: printf("Invalid choice\n");
        }
    }
}
```

8.6 Results

1. Create a graph using adjacency matrix indicating people who can communicate directly with each other
2. DFS traversal method through which any officer can be reachable from a given node
3. BFS traversal method through which any officer can be reachable from a given node
4. Exit

Enter the choice

1

Enter the number of officers

7

Enter the adjacency matrix representation

0 1 0 0 0 1 0

1 0 1 1 0 1 0

0 1 0 1 0 0 0

0 1 1 0 1 0 1

0 0 0 1 0 1 1

1 1 0 0 1 0 0

0 0 0 1 1 0 0

1. Create a graph using adjacency matrix indicating people who can communicate directly with each other
2. DFS traversal method through which any officer can be reachable from a given node
3. BFS traversal method through which any officer can be reachable from a given node
4. Exit
Enter the choice
2
Depth First Search Traversal
Enter Source Officer
1
Reachable officers using DFS method from a given officer: 1
Officer 1->Officer 2
Officer 2->Officer 3
Officer 3->Officer 4
Officer 4->Officer 5
Officer 5->Officer 6
Officer 5->Officer 7
1. Create a graph using adjacency matrix indicating people who can communicate directly with each other
2. DFS traversal method through which any officer can be reachable from a given node
3. BFS traversal method through which any officer can be reachable from a given node
4. Exit
Enter the choice
3
Breadth First Search Traversal
Enter Source Officer
1
Reachable officers using BFS method from a given officer : 1 are
Officer 2
Officer 6
Officer 3

Officer 4

Officer 5

Officer 7

1. Create a graph using adjacency matrix indicating people who can communicate directly with each other

2. DFS traversal method through which any officer can be reachable from a given node

3. BFS traversal method through which any officer can be reachable from a given node

4. Exit

Enter the choice 4

8.7 Pre-Requisite Questions

1. Define Graph.
2. Define Path, Cycle, Subgraph.
3. Define degree of a graph.
4. When graph is said to be strongly connected.

8.8 Post- Experimental Questions

1. Define adjacency list of a graph.
2. Which data structure is used for DFS.
3. Which data structure is used for BFS.
4. List different applications of graph.

EXPERIMENT No.09: Implement hashing technique to map a given key to the address space

9.1 Objective

Design and Implement a program in C that uses Hash Function H:K->L as H(K)=K mod m(reminder method) and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

9.2 Apparatus Required

'C / C++' Programming Language and Linux / Windows as OS, NetBeans IDE 8.2

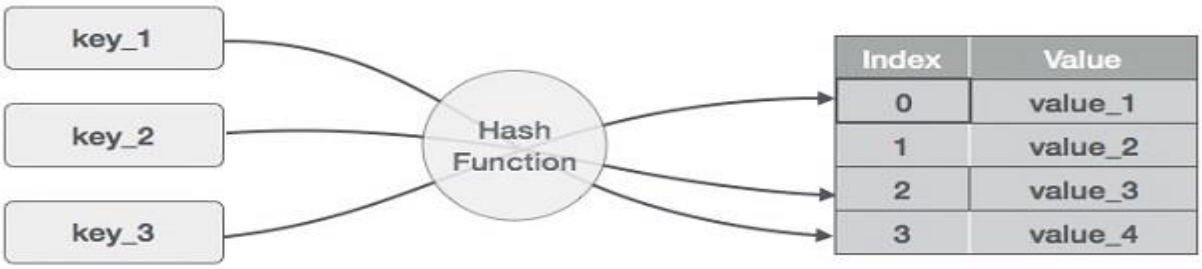
9.3 Pre-Requisite

Basic C Programming, Hashing techniques

9.4 Introduction

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 hash table of size 20, and following items are to be stored. Item are in (key, value) format.



(1,20) (2,70) (42,80) (4,25) (12,44) (14,32) (17,11) (13,78) (37,98)

Sr.No.	Key	Hash	Array 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

When using a linear probe to resolve collision, the item will be stored in the next available slot in the table, assuming that the table is not already full. This is implemented via a linear search for an empty slot, from the point of collision. If the physical end of table is reached during the linear search, the search will wrap around to the beginning of the table and continue from there. If an empty slot is not found before reaching the point of collision, the table is full.

If h is the point of collision, probe through h+1,h+2,h+3.....h+i. till an empty slot is found.

[0]	72
[1]	
[2]	18
[3]	43
[4]	36
[5]	
[6]	6
[7]	

Add the keys 10, 5, and 15 to the previous table .

Hash key = key % table size

2 = 10 % 8

5 = 5 % 8

7 = 15 % 8

[0]	72
[1]	15
[2]	18
[3]	43
[4]	36
[5]	10
[6]	6
[7]	5

9.5 Procedure

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
int *ht,c,n,m,flag;
void create()
{
    int i;
    ht=(int*)malloc(m*sizeof(int));
    if(ht==NULL||m==0)
    {
        printf("Hash table is not present\n");
    }
    for(i=0;i<m;i++)
    ht[i]=-1;
}
void display()
{
    int i;
```

```
        printf("The hash table is\n");
        for(i=0;i<m;i++)
        {
            printf("%d %d\n",i,ht[i]);
        }
}
void insert(int key)
{
        int j;
        j=key%m;
        while(ht[j]!=-1)
        {
            j=(j+1)%m;
            flag=1;
        }
        if(flag)
        {
            printf("Collision is detected and it is solved using linear probing\n");
            flag=0;
        }
        ht[j]=key;
        display();
        c++;
}
void main()
{
        int i,key;
        printf("Enter the number \n");
        scanf("%d",&n);
        printf("Enter the memory size\n");
        scanf("%d",&m);
```

```
        create();
        for(i=0;i<n;i++)
        {
            if(c!=m)
            {
                printf("Enter the key\n");
                scanf("%d",&key);
                insert(key);
            }
            else
            printf("The hash table is full\n");
        }
    }
```

9.6 Results

```
Enter the number
11
Enter the memory size
10
Enter the key
2345
The hash table is
0 -1
1 -1
2 -1
3 -1
4 -1
5 2345
6 -1
7 -1
8 -1
```

```
9 -1
Enter the key
6789
The hash table is
0 -1
1 -1
2 -1
3 -1
4 -1
5 2345
6 -1
7 -1
8 -1
9 6789
Enter the key
3456
The hash table is
0 -1
1 -1
2 -1
3 -1
4 -1
5 2345
6 3456
7 -1
8 -1
9 6789
Enter the key
4567
The hash table is
0 -1
```

1 -1

2 -1

3 -1

4 -1

5 2345

6 3456

7 4567

8 -1

9 6789

Enter the key

5666

Collision is detected and it is solved using linear probing

The hash table is

0 -1

1 -1

2 -1

3 -1

4 -1

5 2345

6 3456

7 4567

8 5666

9 6789

Enter the key

4323

The hash table is

0 -1

1 -1

2 -1

3 4323

4 -1

5 2345

6 3456

7 4567

8 5666

9 6789

Enter the key

4333

Collision is detected and it is solved using linear probing

The hash table is

0 -1

1 -1

2 -1

3 4323

4 4333

5 2345

6 3456

7 4567

8 5666

9 6789

Enter the key

1222

The hash table is

0 -1

1 -1

2 1222

3 4323

4 4333

5 2345

6 3456

7 4567

8 5666

9 6789

Enter the key

1221

The hash table is

0 -1

1 1221

2 1222

3 4323

4 4333

5 2345

6 3456

7 4567

8 5666

9 6789

Enter the key

1200

The hash table is

0 1200

1 1221

2 1222

3 4323

4 4333

5 2345

6 3456

7 4567

8 5666

9 6789

The hash table is full

9.7 Pre-Requisite Questions

1. Define hash table.
2. Define hash function.
3. Explain any one hash function.
4. List two methods to handle overflows.

9.8 Post- Experimental Questions

1. Define linear probing.
2. Define dynamic hashing.
3. List out the applications of hashing.
4. How many different insertion sequences of the key values using the hash function **h(k) = k mod 10** and linear probing will result in the hash table shown above.

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

(A) 10 **(B)** 20 **(C)** 30 **(D)** 40