



SRM VALLIAMMAI ENGINEERING COLLEGE



(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203

DEPARTMENT OF INFORMATION TECHNOLOGY

ACADEMIC YEAR: 2020-2021

ODD SEMESTER

LAB MANUAL

(REGULATION - 2019)

**1908305–DATA STRUCTURES
LABORATORY**

THIRD SEMSTER

B.Tech - Information Technology

Prepared By

Mr.K.ELAIYARAJA, A.P (Sr.G) / IT

Ms.R.SHANTHI, A.P (Sr.G) / IT

INDEX

E.NO	EXPERIMENT NAME	Pg. No.
A	PEO,PO,PSO	4-6
B	Syllabus	7
C	Introduction/ Description of major Software & Hardware involved in lab	8
D	CO, CO-PO Matrix, CO-PSO Matrix	9
E	Mode of Assessment	10
1.1	Array Implementation of List ADTs	11-16
1.2	Linked List Implementation of List ADTs	17-22
2.1	Array Implementation of Stack ADTs	23-27
2.2	Linked List Implementation of Stack ADTs	28-33
3.1	Array Implementation of Queue ADTs	34-39
3.2	Linked List Implementation of Queue ADTs	40-45
4.1	Applications of List : Polynomial Manipulation	46-51
4.2	Applications of Stack: Infix To Postfix Expression	52-57
5	Implementation of Binary trees And operations of Binary trees	58-63
6	Implementation of Binary Search Trees	64-74
7	Implementation of AVL Trees	75-82
8	Implementation of Heaps Using Priority Queues	83-89
9.1	Depth First Search	90-93
9.2	Breadth First Search	94-98
10	Applications of Graphs	99-101

11.1	Linear Search	102-104
11.2	Binary Search	105-107
11.3	Bubble Sort	108-110
11.4	Insertion Sort	111-112
12.1	Hashing With Separate Chaining	113-116
12.2	Hashing With Open Addressing	117-124
-	TOPIC BEYOND THE SYLLABUS “ Implementation of Doubly Linked List”	125-133



PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

1. To afford the necessary background in the field of Information Technology to deal with engineering problems to excel as engineering professionals in industries.
2. To improve the qualities like creativity, leadership, teamwork and skill thus contributing towards the growth and development of society.
3. To develop ability among students towards innovation and entrepreneurship that caters to the needs of Industry and society.
4. To inculcate and attitude for life-long learning process through the use of information technology sources.
5. To prepare then to be innovative and ethical leaders, both in their chosen profession and in other activities.

PROGRAMME OUTCOMES (POs)

After going through the four years of study, Information Technology Graduates will exhibit ability to:

PO#	Graduate Attribute	Programme Outcome
1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2	Problem analysis	Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	Design/development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions
5	Modern tool usage	Create, select, and apply appropriate techniques, resources, and

		modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
6	The engineer and society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7	Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings
10	Communication	Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions
11	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
12	Life-long learning	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

PROGRAMME SPECIFIC OUTCOMES (PSOs)

By the completion of Information Technology program the student will have following Program specific outcomes

1. Design secured database applications involving planning, development and maintenance using state of the art methodologies based on ethical values.
2. Design and develop solutions for modern business environments coherent with the advanced technologies and tools.
3. Design, plan and setting up the network that is helpful for contemporary business environments using latest hardware components.
4. Planning and defining test activities by preparing test cases that can predict and correct errors ensuring a socially transformed product catering all technological needs.



1908305

DATA STRUCTURES LABORATORY

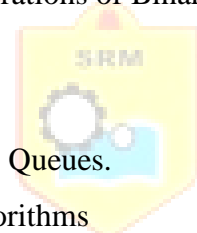
L T P C

0 0 4 2

OBJECTIVES

- To implement linear and non-linear data structures
- To implement non-linear data structures
- To understand the different operations of search trees
- To implement graph traversal algorithms
- To get familiarized to sorting and searching algorithms

1. Array and Linked list implementation of List ADT
2. Array and Linked list implementation of Stack ADT
3. Array and Linked list implementation of Queue ADT
4. Applications of List, Stack and Queue ADTs.
5. Implementation of Binary trees and operations of Binary trees.
6. Implementation of Binary Search Trees
7. Implementation of AVL Trees
8. Implementation of Heaps using Priority Queues.
9. Graph representation and Traversal algorithms
10. Applications of Graphs
11. Implementation of searching and sorting algorithms
12. Hashing – any two collision techniques



Total: 60 Periods

LIST OF EQUIPMENTS FOR A BATCH OF 30 STUDENTS

HARDWARE

- Standalone desktops 30 Nos

SOFTWARE

- C / C++ / Turbo C / Equivalent Software

C

C programming is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system. C is the most widely used computer language. It keeps fluctuating at number one scale of popularity along with Java programming language, which is also equally popular and most widely used among modern software programmers.

- The UNIX OS was totally written in C.
- Today C is the most widely used and popular System Programming Language.
- Most of the state-of-the-art software have been implemented using C.
- Today's most popular Linux OS and RDBMS MySQL have been written in C.

C++

C++ is a middle-level programming language developed by Bjarne Stroustrup starting in 1979 at Bell Labs. C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. This C++ tutorial adopts a simple and practical approach to describe the concepts of C++ for beginners to advanced software engineers.

Turbo C

Turbo C was an integrated development environment (IDE) for programming in the C language. It was developed by Borland and first introduced in 1987. At the time, Turbo C was known for its compact size, comprehensive manual, fast compile speed and low price.

COURSE OUTCOMES

1908305.1	Write functions to implement linear and non-linear data structure operations.
1908305.2	Suggest appropriate linear data structure operations for solving a given problem.
1908305.3	Appropriate use of linear / non-linear data structure operations for a given problem.
1908305.4	Apply appropriate hash functions that result in a collision free scenario for data storage and retrieval.
1908305.5	Apply the searching and sorting algorithms for problem solving.

CO- PO MATRIX

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
1908305.1	3	-	-	-	2	-	-	-	-	-	-	-
1908305.2	2	2	-	-	-	-	-	-	-	-	-	-
1908305.3	2	2	-	-	-	-	-	-	-	-	-	-
1908305.4	3	3	3	-	2	-	-	-	-	-	-	-
1908305.5	3	2	3	-	2	-	-	-	-	-	-	-
Average	3	2	3	-	2	-	-	-	-	-	-	-

CO- PSO MATRIX

1908305	PSO 1	PSO 2
1908305.1	3	2
1908305.2	3	2
1908305.3	3	-
1908305.4	3	2
1908305.5	3	2
Average	3	2

EVALUATION PROCEDURE FOR EACH EXPERIMENTS

S.No	Description	Mark
1.	Aim & Pre-Lab discussion	20
2.	Observation	20
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	20
Total		100

INTERNAL ASSESSMENT FOR LABORATORY

S.No	Description	Mark
1.	Conduction & Execution of Experiment	25
2.	Record	10
3.	Model Test	15
Total		50

Ex.No:1.1 ARRAY IMPLEMENTATION OF LIST ADTs

AIM

To write a C program to implement list using an array

PRE LAB DISCUSSION

Array of list is an important data structure used in many applications. It is an interesting structure to form a useful data structure. It combines static and dynamic structure. Static means array and dynamic means linked list used to form a useful data structure. Array elements can be stored in consecutive manner in memory. Insert and delete operation takes more time in array. Array elements cannot be added, deleted once it is declared. In array, elements can be modified easily by identifying the index value. Pointer cannot be used in array. So, it does not require extra space in memory for pointer.

ALGORITHM

Step 1: Start.

Step 2: Declare the necessary functions for implementation.

Step 3: Get the input from the user and store it an array.

Step 4: In Insertion, half of the elements to be shifted upwards and in deletion half of the elements to be shifted downwards.

Step 5: Display the output using an array.

Step 6: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
void create();
void insert();
void deletion();
```

```

void search();
void display();
int a,b[20], n, p, e, f, i, pos;

void main()
{
//clrscr();
int ch;
char g='y';
do
{
printf("\n main Menu");
printf("\n 1.Create \n 2.Delete \n 3.Search \n 4.Insert \n 5.Display\n 6.Exit \n");
printf("\n Enter your Choice");
scanf("%d", &ch);

switch(ch)
{
case 1:
create();
break;

case 2:
deletion();
break;

case 3:
search();
break;

case 4:
insert();
break;

case 5:
display();
break;

case 6:
exit();
break;

default:
printf("\n Enter the correctchoice:");
}
printf("\n Do u want tocontinue::");

```



```

scanf("\n%c", &g);
}
while(g=='y'||g=='Y');
getch();
}

```

```

void create()
{
printf("\n Enter the number of nodes");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\n Enter the Element:",i+1);
scanf("%d", &b[i]);
}
}

```

```

void deletion()
{
printf("\n Enter the position u want to delete:");
scanf("%d", &pos);
if(pos>=n)
{
printf("\n Invalid Location:");
}
else
{
for(i=pos+1;i<n;i++)
{
b[i-1]=b[i];
}
n--;
}
printf("\n The Elements after deletion");
for(i=0;i<n;i++)
{
printf("\t%d", b[i]);
}
}

```

```

void search()
{
printf("\n Enter the Element to be searched:");
scanf("%d", &e);

for(i=0;i<n;i++)

```



```

{
if(b[i]==e)
{
printf("Value is in the %d Position", i);
}
else
{
printf("Value %d is not in the list::", e);
continue;
}
}
}

void insert()
{
printf("\n Enter the position u need to insert::");
scanf("%d", &pos);

if(pos>=n)
{
printf("\n invalid Location::");
}
else
{
for(i=MAX-1;i>=pos-1;i--)
{
b[i+1]=b[i];
}
printf("\n Enter the element to insert::\n");
scanf("%d",&p);
b[pos]=p;
n++;
}
printf("\n The list after insertion::\n");

display();
}

void display()
{
printf("\n The Elements of The list ADT are:");
for(i=0;i<n;i++)
{
printf("\n\n%d", b[i]);
}
}
}

```



OUTPUT

Main Menu

- 1.Create
- 2.Delete
- 3.Search
- 4.Insert
- 5.Display
- 6.Exit

Enter your Choice1

Enter the number of nodes 2

Enter theElement:2

Enter theElement:3

Do u want to continue:::y

Main Menu

- 1.Create
- 2.Delete
- 3.Search
- 4.Insert
- 5.Display
- 6.Exit

Enter your Choice 5

The Elements of The list ADT are:

2

3

Do u want to continue:::y

Main Menu

- 1.Create
- 2.Delete
- 3.Search
- 4.Insert
- 5.Display
- 6.Exit



Enter your Choice 4

Enter the position u need to insert::0

Enter the element to inert::5

The list after insertion::

The Elements of the list ADT are:

5

2

3

Do u want to continue::y

Main Menu

1.Create

2.Delete

3.Search

4.Insert

5.Display

6.Exit

Enter your Choice 2

Enter the position u want to delete::1

The Elements after deletion

5

3

Do u want to continue:::



VIVA (PRE & POST LAB) QUESTIONS

- 1.How to delete an element using array inlist?
2. How to insert an element using array inlist?
3. Which data structure refers to linear collection of dataitems.
4. How to search an element inlist?
5. What happens when there is no space left in free storagelist?

RESULT

Thus the C program to implement list using array was completed successfully.

Ex.No: 1.2

LINKED LIST IMPLEMENTATION OF LIST ADTs

AIM

To write a C program to implement singly linked list.

PRE LAB DISCUSSION

Linked list is a linear data structure. It is a collection of data elements, called nodes pointing to the next node by means of a pointer. In linked list, each node consists of its own data and the address of the next node and forms a chain.

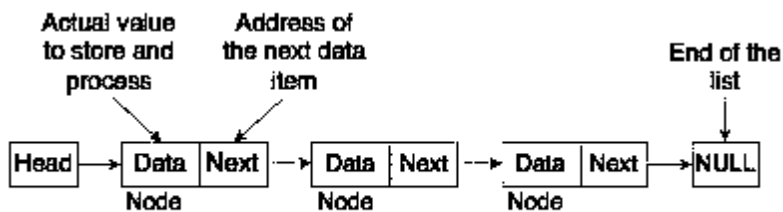
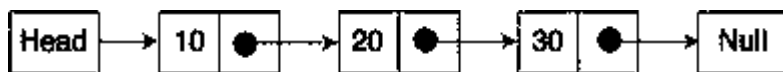


Fig. Linked List

Linked list contains a link element called first and each link carries a data item. Entry point into the linked list is called the head of the list. Link field is called next and each link is linked with its next link. Last link carries a link to null to mark the end of the list.

Linked list is a dynamic data structure. While accessing a particular item, start at the head and follow the references until you get that data item.

Linked list is used while dealing with an unknown number of objects:



Linked list contains two fields - First field contains value and second field contains a link to the next node. The last node signifies the end of the list that means NULL. The real life example of Linked List is that of Railway Carriage. It starts from engine and then the coaches follow. Coaches can traverse from one coach to other, if they connected to each other.

ALGORITHM

Step 1: Start

Step 2: Creation: Get the number of elements, and create the nodes having structures DATA, LINK and store the element in Data field, link them together to form a linked list.

Step 3: Insertion: Get the number to be inserted and create a new node store the value in DATA field. And insert the node in the required position.

Step 4: Deletion: Get the number to be deleted. Search the list from the beginning and locate the node then delete the node.

Step 5: Display: Display all the nodes in the list.

Step 6: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define NULL 0

typedef struct list
{
    int no;
    struct list *next;
}LIST;
LIST *p,*t,*h,*y,*ptr,*pt;

void create( void );
void insert( void );
void delet( void );
void display ( void );
int j,pos,k=1,count;

void main()
```



```

{
  int n,i=1,opt;
  clrscr();
  p = NULL;
  printf("%d",sizeof(LIST));
  printf( "Enter the no of nodes :\n " );
  scanf( "%d",&n );
  count = n;
  while( i <= n)
  {
  create();
  i++;
  }
  printf("\nEnter your option:\n");
  printf("1.Insert \t 2.Delete \t 3.Display \t 4.Exit\n");
  do
  {
  scanf("%d",&opt);
  switch( opt )
  {
  case 1:
  insert();
  count++;
  break;
  case 2:
  delet();
  count--;
  if ( count == 0 )
  {
  printf("\n List is empty\n");
  }
  break;

  case 3:
  printf("List elements are:\n");
  display();
  break;
  }
  printf("\nEnter your option \n");
  }while( opt != 4 );
  getch();
  }

void create ( )
{
  if( p== NULL )

```



```

{
p = ( LIST * ) malloc ( sizeof ( LIST ) );
printf( "Enter the element:\n" );
scanf( "%d",&p->no );
p->next = NULL;
h = p;
}
else
{
t = ( LIST * ) malloc (sizeof( LIST ));
printf( "\nEnter the element" );
scanf( "%d",&t->no );
t->next = NULL;
p->next = t;
p = t;
}
}

```

```

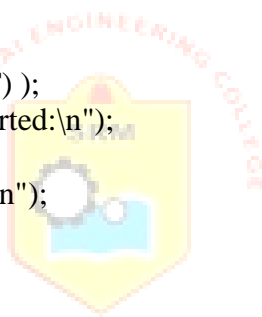
void insert()
{
t=h;
p = ( LIST * ) malloc ( sizeof(LIST) );
printf("Enter the element to be inserted:\n");
scanf("%d",&p->no);
printf("Enter the position to insert:\n");
scanf( "%d",&pos );
if( pos == 1 )
{
h = p;
h->next = t;
}
else
{
for(j=1;j<(pos-1);j++)
t = t->next;
p->next = t->next;
t->next = p;
t=p;
}
}

```

```

void delet(){
printf("Enter the position to delete:\n");
scanf( "%d",&pos );
if( pos == 1 )
{

```



```

h = h->next ;
}
else
{
t=h;
for(j=1;j<(pos-1);j++)
t = t->next;
pt=t->next->next;
free(t->next);
t->next= pt;
}
}

void display()
{
t= h;
while( t->next != NULL )
{
printf("\t%d",t->no);
t = t->next;
}
printf( "\t %d\t",t->no );
}

```



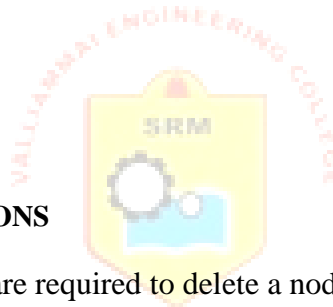
OUTPUT

```

Enter the no of nodes: 3
Enter the element: 1
Enter the element 2
Enter the element 3
Enter your option:
1. Insert 2.Delete 3.Display 4.Exit
3
List elements are:
1 2 3
Enter your option 1
Enter the element to be inserted:
12
Enter the position to insert: 1
Enter your option 3
List elements are:
12 1 2 3
Enter your option 1
Enter the element to be inserted:
13
Enter the position to insert: 3

```

Enter your option 1
Enter the element to be inserted:
14
Enter the position to insert:6
Enter your option 3
List elements are:
12 1 13 2 3 14
Enter your option2
Enter the position todelete:1
Enter your option3
List elements are:
1 13 2 3 14
Enter your option2
Enter the position todelete:3
Enter your option3
List elements are:
1 13 3 14
Enter your option2
Enter the position todelete:4
Enter your option3
List elements are:
1 13 3
Enter your option:6



VIVA (PRE & POST LAB) QUESTIONS

1. How many modifications are required to delete a node at the beginning?
- 2 How many modifications are required to insert a node in the middle of the linked list?
- 3.Which node contains NULL pointer in a single linked list?
- 4.Which node points to the first node in list?
- 5.How does array differ from linked list?

RESULT

Thus the C program to implement List was completed successfully.

Ex. No: 2.1

ARRAY IMPLEMENTATION OF STACK ADTs

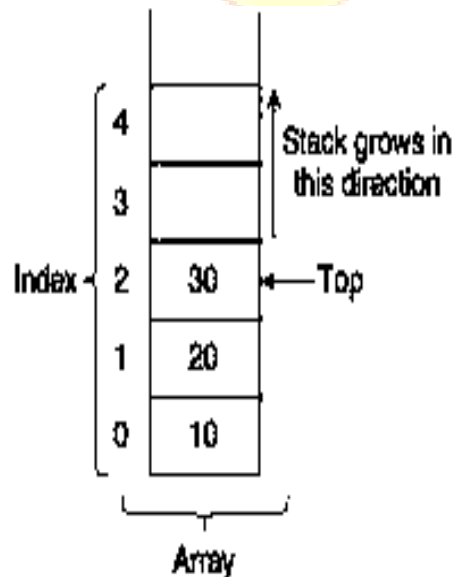
AIM

To write a C program to implement Stack operations such as push, pop and display using array.

PRE LAB DISCUSSION

An array is a random access data structure, where each element can be accessed directly and in constant time. A typical illustration of random access is a book - each page of the book can be open independently of others. Random access is critical to many algorithms, for example binarysearch.

A stack data structure can be implemented using one dimensional array. But stack implemented using array, can store only fixed number of data values. This implementation is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using LIFO principle with the help of a variable 'top'. Initially top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.



ALGORITHM

Step 1: Start.

Step 2: Initialize top = -1;

Step 3: Push operation increases top by one and writes pushed element to storage[top];

Step 4: Pop operation checks that top is not equal to -1 and decreases top variable by 1;

Step 5: Display operation checks that top is not equal to -1 and returns storage[top];

Step 6: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define size 5
int item;
int s[10];
int top;
void display()
{
int i;
if(top== -1)
{
printf("\nstack is empty");
return;
}
printf("\nContent of stack is:\n");
for(i=0;i<=top;i++)
printf("%d\t",s[i]);
}
void push()
{
```




```

if(top==size-1)
{
printf("\nStack is full");
return;
}
printf("\nEnter item:\n");
scanf("%d",&item);
s[++top]=item;
}
void pop()
{
if(top==-1)
{
printf("\nstack is empty");
return;
}
printf("\nDeleted item is: %d",s[top]);
top--;
}
void main()
{
int ch;
top=-1;
clrscr();
printf("\n1.push\t2.pop\n3.display\t4.exit\n");
do{
printf("\nEnter your choice:\n");
scanf("%d",&ch);
switch(ch)
{
case 1:// printf("Enter item:\n");

```



```
//scanf("%d",&item);
push();
break;
case 2: pop();
break;
case 3: display();
break;
case 4: exit(0);
default: printf("\nWrong entry ! try again");
}
}while(ch<=4);
getch();
}
```

OUTPUT

```
1.push 2.pop
3.display 4.exit
Enter your choice:
1
Enter item:
100
Enter your choice:
1
Enter item:
200
Enter your choice:
1
Enter item:
300
Enter your choice:
2
```



Deleted item is: 300

Enter your choice:

3

Content of stack is:

100 200

Enter your choice:4

VIVA (PRE & POST LAB) QUESTIONS

- 1.How to remove an element from stack?
- 2.How to insert an element into a stack?
- 3.Is it possible to store any number of data elements in stack?
- 4.What are the demerits of stack?
5. What happens when you pop from an empty stack while implementing using the Stack ADT ?



RESULT

Thus the C program to implement stack using array was executed successfully.

Ex. No: 2.2

LINKED LIST IMPLEMENTATION OF STACK

AIM

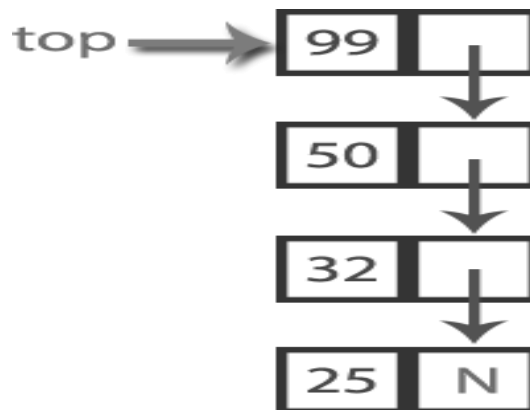
To write a C program to implement Stack operations such as push, pop and display using linked list.

PRE LAB DISCUSSION

The major problem with the stack implemented using array is, it works only for fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using array is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using linked list data structure. The stack implemented using linked list can work for unlimited number of values. That means, stack implemented using linked list works for variable size of data. So, there is no need to fix the size at the beginning of the implementation. The Stack implemented using linked list can organize as many data values as we want.

In linked list implementation of a stack, every new element is inserted as 'top' element. That means every newly inserted element is pointed by 'top'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by 'top' by moving 'top' to its next node in the list. The next field of the first element must be always NULL.

Example



In above example, the last inserted node is 99 and the first inserted node is 25. The order of elements inserted is 25, 32, 50 and 99.

There are two basic operations performed in a Stack:

1. Push():is used to add or insert new elements into thestack.
2. Pop():is used to delete or remove an element from thestack.

ALGORITHM

Step 1: Start.

Step 2: push operation inserts an element at the front.

Step 4: pop operation deletes an element at the front of the list;

Step 5: display operation displays all the elements in the list.

Step 6: Stop.

PROGRAM

```
#include "stdio.h"
#include "stdlib.h"
#include "conio.h"
void pop();
void push(int value);
void display();
struct node
{
    int data;
    struct node *link;
};
struct node *top=NULL,*temp;
```



```

void main()
{
    int choice,data;
    while(1) //infinite loop is used to insert/delete infinite number of elements in stack
    {
        printf("\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
        printf("\nEnter ur choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: //To push a new element into stack
                printf("Enter a new element :");
                scanf("%d",&data);
                push(data);
                break;
            case 2: // pop the element from stack
                pop();
                break;
            case 3: // Display the stack elements
                display();
                break;
            case 4: // To exit
                exit(0);
        }
    }
    getch();
    //return 0;
}

void display()
{
    temp=top;

```

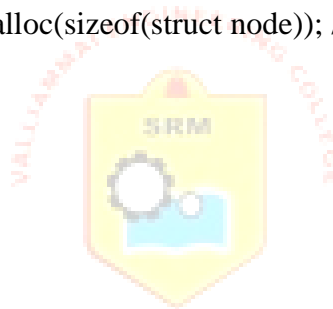
```

    if(temp==NULL)
    {
        printf("\nStack is empty\n");
    }
    printf("\n The Contents of the Stackare...");
    while(temp!=NULL)
    {
        printf(" %d->",temp->data);
        temp=temp->link;
    }
}

void push(int data)
{
    temp=(struct node *)malloc(sizeof(struct node)); // creating a space for the new
    element.
    temp->data=data;
    temp->link=top;
    top=temp;
    display();
}

void pop()
{
    if(top!=NULL)
    {
        printf("The popped element is %d",top->data);
        top=top->link;
    }
    else
    {
        printf("\nStack Underflow");
    }
}

```



```
display();  
}
```

OUTPUT

1.Push

2.Pop

3.Display

4.Exit

Enter ur choice:1

Enter a new element :10

The Contents of the Stack are... 10 ->

1.Push

2.Pop

3.Display

4.Exit

Enter ur choice:1

Enter a new element :20

The Contents of the Stack are... 20 -> 10 ->

1.Push

2.Pop

3.Display

4.Exit

Enter ur choice:1

Enter a new element :30

The Contents of the Stack are... 30 -> 20 -> 10 ->

1.Push

2.Pop

3.Display

4.Exit



Enter ur choice:2

The popped element is 30

The Contents of the Stack are... 20 -> 10 ->

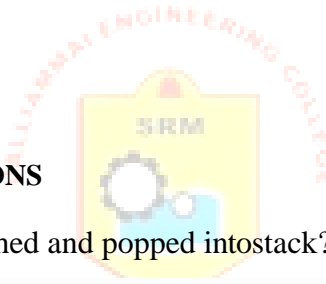
- 1.Push
- 2.Pop
- 3.Display
- 4.Exit

Enter ur choice:3

The Contents of the Stack are... 20 -> 10 ->

- 1.Push
- 2.Pop
- 3.Display
- 4.Exit

Enter ur choice:4



VIVA (PRE & POST LAB) QUESTIONS

1. How an element can be pushed and popped into stack?
2. If the elements "A", "B", "C" and "D" are placed in a stack and are deleted one at a time, what is the order of removal?
3. What is the top of stack?
4. How to represent stack using linked list?
5. What is the structure of stack?

RESULT

Thus the C program to implement Stack using linked list was completed successfully.

Ex. No: 3.1

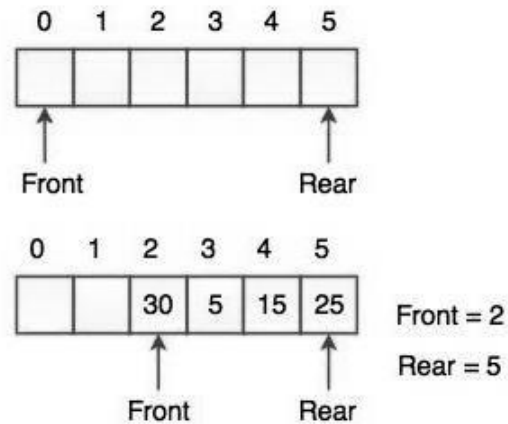
ARRAY IMPLEMENTATION OF QUEUE ADTs

AIM

To write a C program to implement Queue operations such as enqueue, dequeue and display using array.

PRE LAB DISCUSSION

Queue implemented using array can store only fixed number of data values. The implementation of queue data structure using array is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using FIFO (First In First Out) principle with the help of variables 'front' and 'rear'. Initially both 'front' and 'rear' are set to -1. Whenever, we want to insert a new value into the queue, increment 'rear' value by one and then insert at that position. Whenever we want to delete a value from the queue, then increment 'front' value by one and then display the value at 'front' position as deleted element.



- The Front and Rear of the queue point at the first index of the array. (Array index starts from 0).
- While adding an element into the queue, the Rear keeps on moving ahead and always points to the position where the next element will be inserted. Front remains at the first index.

ALGORITHM

Step 1: Start.

Step 2: Initialize front=0; rear=-1.

Step 3: Enqueue operation moves a rear by one position and inserts a element at the rear.

Step 4: Dequeue operation deletes a element at the front of the list and moves the front by one Position.

Step 5: Display operation displays all the element in the list.

Step 6: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define SIZE5      /* Size of Queue*/
int Q[SIZE],f=0,r=-1; /* Global declarations*/
Qinsert(int elem)
{
    /* Function for Insert operation*/
    if(Qfull())
printf("\n\n Overflow!!!!\n\n");
    else
    {
        ++r;
        Q[r]=elem;
    }
}
```



```

int Qdelete()
{
    /* Function for Delete operation*/
    int elem;
    if(Qempty()){ printf("\n\nUnderflow!!!!\n\n");
    return(-1); }
    else
    {
        elem=Q[f];
        f=f+1;
        return(elem);
    }
}

```

```

int Qfull()
{
    /* Function to Check Queue Full*/
    if(r==SIZE-1) return 1;
    return 0;
}

```

```

int Qempty()
{
    /* Function to Check Queue Empty*/
    if(f > r) return 1;
    return 0;
}

```

```

display()
{
    /* Function to display status of Queue*/
    inti;
    if(Qempty()) printf(" \n Empty Queue\n");
    else

```



```

    {
        printf("Front->");
        for(i=f;i<=r;i++)
            printf("%d ",Q[i]);
        printf("<-Rear");
    }
}

void main()
{
    /* Main Program */
    int opn,elem;
    do
    {
        clrscr();
        printf("\n ### Queue Operations using Arrays### \n\n");
        printf("\n Press 1-Insert, 2-Delete,3-Display,4-Exit\n");
        printf("\n Your option ? ");
        scanf("%d",&opn);
        switch(opn)
        {
            case 1: printf("\n\nRead the element to be Inserted ?");
                    scanf("%d",&elem);
                    Qinsert(elem); break;
            case 2: elem=Qdelete();
                    if( elem != -1)
                        printf("\n\nDeleted Element is %d \n",elem);
                    break;
            case 3: printf("\n\nStatus of Queue\n\n");
                    display(); break;
            case 4: printf("\n\n Terminating \n\n"); break;
            default: printf("\n\nInvalid Option !!! Try Again !! \n\n");
        }
    }
}

```

```

        break;
    }
    printf("\n\n\n Press a Key to Continue . . . ");
    getch();
}while(opn != 4);
getch();
}

```

OUTPUT

```

### Queue Operations using Arrays###
Press 1-Insert, 2-Delete,3-Display,4-Exit
Your option ? 1
Read the element to be Inserted ?100
Press a Key to Continue . . .
### Queue Operations using Arrays###
Press 1-Insert, 2-Delete,3-Display,4-Exit
Your option ? 1
Read the element to be Inserted ?200
Press a Key to Continue . . .
### Queue Operations using Arrays###
Press 1-Insert, 2-Delete,3-Display,4-Exit
Your option ? 1
Read the element to be Inserted ?300
Press a Key to Continue . . .
### Queue Operations using Arrays###
Press 1-Insert, 2-Delete,3-Display,4-Exit
Your option ? 2
Deleted Element is 100
Press a Key to Continue . . .

### Queue Operations using Arrays###

```



Press 1-Insert,2-Delete,3-Display,4-Exit

Your option ?3

Status ofQueue

Front->200 300 <-Rear

Press a Key to Continue . . .

VIVA (PRE & POST LAB) QUESTIONS

- 1.If the elements “A”, “B”, “C” and “D” are placed in a queue and are deleted one at a time, in what order will they beremoved?
2. When does a normal queue gets full if implemented using an array of sizeMAX_SIZE
3. What is the term for inserting into a full queue known as?
- 4.What is the advantage of circular queue over linear queue?
- 5.How to check an emptyqueue?



RESULT

Thus the C program to implement Queue using array was completed successfully.

Ex. No: 3.2

LINKED LIST IMPLEMENTATION OF QUEUE

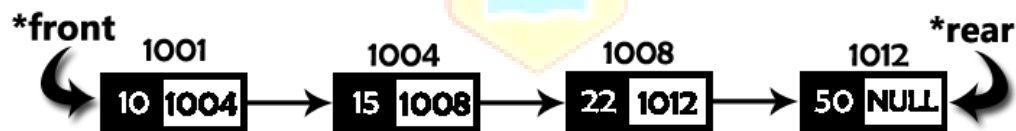
AIM

To write a C program to implement Queue operations such as enqueue, dequeue and display using linked list.

PRE LAB DISCUSSION

The major problem with the queue implemented using array is, It will work for only fixed number of data. That means the amount of data must be specified in the beginning itself. Queue using array is not suitable when we don't know the size of data which we are going to use. A queue data structure can be implemented using linked list data structure. The queue which is implemented using linked list can work for unlimited number of values. That means, queue using linked list can work for variable size of data (No need to fix the size at beginning of the implementation). The Queue implemented using linked list can organize as many data values as we want. In linked list implementation of a queue, the last inserted node is always pointed by '**rear**' and the first node is always pointed by '**front**'.

Example



In above example, the last inserted node is 50 and it is pointed by '**rear**' and the first inserted node is 10 and it is pointed by '**front**'. The order of elements inserted is 10, 15, 22 and 50.

There are two basic operations performed on a Queue.

Enqueue(): This function defines the operation for adding an element into queue. Dequeue(): This function defines the operation for removing an element from queue.

ALGORITHM

Step 1: Start.

Step 2: Enqueue operation inserts an element at the rear of the list.

Step 4: Dequeue operation deletes an element at the front of the list.

Step 5: Display operation display all the element in the list.

Step 6: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct Node
{
    int data;
    struct Node *next;
}*front = NULL,*rear = NULL;

void insert(int);
void delete();
void display();

void main()
{
    int choice, value;
    clrscr();
    printf("\n:: Queue Implementation using Linked List ::\n");
    while(1){
        printf("\n***** MENU *****\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d",&choice);
```



```

switch(choice){
case 1: printf("Enter the value to be insert: ");
        scanf("%d", &value);
        insert(value);
        break;
case 2: delete(); break;
case 3: display(); break;
case 4: exit(0);
default: printf("\nWrong selection!!! Please try again!!!\n");
}
}
}
void insert(int value)
{
struct Node *newNode;
newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode -> next = NULL;
if(front == NULL)
    front = rear = newNode;
else{
    rear -> next = newNode;
    rear = newNode;
}
printf("\nInsertion is Success!!!\n");
}
void delete()
{
if(front == NULL)
    printf("\nQueue is Empty!!!\n");
else{

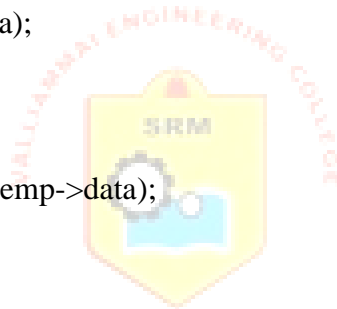
```



```

    struct Node *temp = front;
    front = front -> next;
    printf("\nDeleted element: %d\n", temp->data);
    free(temp);
}
}
void display()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else{
        struct Node *temp = front;
        while(temp->next != NULL){
            printf("%d-->",temp->data);
            temp = temp -> next;
        }
        printf("%d-->NULL\n",temp->data);
    }
}
}

```



OUTPUT

```
C:\WINDOWS\system32\cmd.exe - tc

:: Queue Implementation using Linked List ::

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 10

Insertion is Success!!!

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20_
```

```
C:\WINDOWS\system32\cmd.exe - tc

10--->20--->NULL

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2

Deleted element: 10

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
20--->NULL

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: _
```

VIVA (PRE & POST LAB) QUESTIONS

1. What is structure of queue?
2. How an element can be inserted and deleted from queue?
3. How to represent queue using linked list?
4. In linked list implementation of a queue, where does a new element be inserted?
5. In linked list implementation of a queue, from where is the item deleted?



RESULT

Thus the C program to implement Queue using linked list was completed successfully.

Ex.No:4.1

APPLICATIONS OF LIST

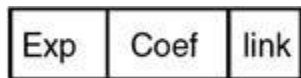
AIM

To write a 'C' program to represent a polynomial as a linked list and write functions for polynomial addition

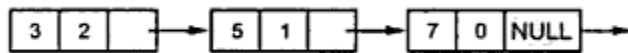
PRE LAB DISCUSSION

A **polynomial equation** is an **equation** that can be written in the form. $ax_n + bx_{n-1} + \dots + rx + s = 0$, where a, b, . . . , r and s are constants. We call the largest exponent of x appearing in a nonzero term of a **polynomial** the degree of that **polynomial**. A Polynomial has mainly two fields Exponent and coefficient.

Node of a Polynomial:



For example $3x^2 + 5x + 7$ will represent as follows.



In each node the exponent field will store the corresponding exponent and the coefficient field will store the corresponding coefficient. Link field points to the next item in the polynomial. Given two polynomial numbers represented by a linked list. Write a function that add these lists means add the coefficients which have same variable powers.

Example:

Input:

$$\text{1st number} = 5x^2 + 4x^1 + 2x^0$$

$$\text{2nd number} = 5x^1 + 5x^0$$

Output:

$$5x^2 + 9x^1 + 7x^0$$

ALGORITHM

- Step1:Start the program
- Step2:Get the coefficients and powers for the two polynomials to be added.
- Step3:Add the coefficients of the respective powers.
- Step4:Display the added polynomial.
- Step5:Terminate the program.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct polynomial
{
    int coff;
    int pow;
    struct polynomial *link;
}*ptr,*start1,*node,*start2,*start3,*ptr1,*ptr2;
typedef struct polynomial pnl;
int temp1,temp2;

void main()
{
    void create(void);
    void prnt(void);
    void suml(void);
    void sort(void);
    clrscr();
    printf("Enrter the elements of the first polynomial :");
    node = (pnl *) malloc(sizeof (pnl));
    start1=node;
    if (start1==NULL)
    {
        printf(" Unable to create memory.");
        getch();
        exit();
    }
    create();
    printf("Enter the elements of the second poly :");
    node = (pnl *) malloc(sizeof (pnl));
```



```

start2=node;
if (start2==NULL)
{
    printf("Unable to create memory.");
    getch();
    exit();
}
create();
clrscr();
//printing the elements of the lists
printf("The elements of the poly first are :");
ptr=start1;
prnt();
printf("The elements of the poly second are :");
ptr=start2;
prnt();
printf("The first sorted list is :");
ptr=start1;
sort();
ptr=start1;
prnt();
printf("The second sorted list is :");
ptr=start2;
sort();
ptr=start2;
prnt();
printf("The sum of the two lists are :");
suml();
ptr=start3;
prnt();
getch();
}
/*.....*/
voidcreate()
{
    char ch;
    while(1)
    {
        printf(" Enter the coff and pow :");
        scanf("%d%d",&node->coff,&node->pow);
        if (node->pow==0 )
        {
            ptr=node;
            node=(pnl *)malloc(sizeof(pnl));
            node=NULL;
            ptr->link=node;
        }
    }
}

```




```

        break;
    }
    printf("Do u want enter more coff?(y/n)");
    fflush(stdin);
    scanf("%c",&ch);
    if (ch=='n')
    {
        ptr=node;
        node=(pnl *)malloc(sizeof(pnl));
        node=NULL;
        ptr->link=node;
        break;
    }
    ptr=node;
    node=(pnl *)malloc(sizeof(pnl));
    ptr->link=node;
}
}
/*-----*/
voidprint()
{
    int i=1;
    while(ptr!=NULL)
    {
        if(i!=1)
            printf("+ ");
        printf(" %dx^%d\n ",ptr->coff,ptr->pow);
        ptr=ptr->link;
        i++;
    }
    //printf(" %d^%d",ptr->coff,ptr->pow);
}
/*-----*/
voidsort()
{
    for(;ptr->coff!=NULL;ptr=ptr->link)
    for(ptr2=ptr->link;ptr2->coff!=NULL;ptr2=ptr2->link)
    {
        if(ptr->pow>ptr2->pow)
        {
            temp1=ptr->coff;
            temp2=ptr->pow;
            ptr->coff=ptr2->coff;
            ptr->pow=ptr2->pow;
            ptr2->coff=temp1;
            ptr2->pow=temp2;
        }
    }
}

```



```
        }
    }
}
/*-----*/
void sum1()
{
    node=(pnl *)malloc (sizeof(pnl));
    start3=node;

    ptr1=start1;
    ptr2=start2;

    while(ptr1!=NULL && ptr2!=NULL)
    {
        ptr=node;
        if (ptr1->pow > ptr2->pow )
        {
            node->coff=ptr2->coff;
            node->pow=ptr2->pow;
            ptr2=ptr2->link; //update ptr list B
        }
        else if ( ptr1->pow < ptr2->pow )
        {
            node->coff=ptr1->coff;
            node->pow=ptr1->pow;
            ptr1=ptr1->link; //update ptr list A
        }
        else
        {
            node->coff=ptr2->coff+ptr1->coff;
            node->pow=ptr2->pow;
            ptr1=ptr1->link; //update ptr list A
            ptr2=ptr2->link; //update ptr list B
        }

        node=(pnl *)malloc (sizeof(pnl));
        ptr->link=node; //update ptr listC
    }//end of while

    if(ptr1==NULL) //end of listA
    {
        while(ptr2!=NULL)
        {
            node->coff=ptr2->coff;
            node->pow=ptr2->pow;
            ptr2=ptr2->link; //update ptr list B
        }
    }
}
```



```

ptr=node;
node=(pnl *)malloc (sizeof(pnl));
ptr->link=node; //update ptr list C
}
}
elseif(ptr2==NULL) //end of listB
{
while(ptr1!=NULL)
{
node->coff=ptr1->coff;
node->pow=ptr1->pow;
ptr1=ptr1->link; //update ptr list B
ptr=node;
node=(pnl *)malloc (sizeof(pnl));
ptr->link=node; //update ptr list C
}
}
node=NULL;
ptr->link=node;
}

```

OUTPUT

Enter the elements of the first polynomial : Enter the coff and pow :1 1
Do u want enter more coff?(y/n)y
Enter the coff and pow :1 0
Enter the elements of the second poly : Enter the coff and pow :1 1
Do u want enter more coff?(y/n)y
Enter the coff and pow :2 0
The elements of the poly first are : $1x^1 + 1x^0$
The elements of the poly second are : $1x^1 + 2x^0$
The first sorted list is : $1x^0 + 1x^1$
The second sorted list is : $2x^0 + 1x^1$
The sum of the two lists are : $3x^0 + 2x^1$

VIVA (PRE & POST LAB) QUESTIONS

1. Give the linked representation of the following polynomial: $7x^3y^2 - 8x^2y + 3xy + 11x - 4$
2. Which data structure is suited for polynomial manipulation. Give reason.
3. How to add two polynomial using linked list.
4. Specify the need for malloc() function.
5. Mention the structure of polynomial node.

RESULT

Thus the C program to implement Polynomial using linked list was completed successfully.

AIM

To write a C program to implement the conversion of infix to postfix expression using Stack.

PRE LAB DISCUSSION

One of the applications of Stack is in the conversion of arithmetic expressions in high-level programming languages into machine readable form. An arithmetic expression may consist of more than one operator and two operands e.g. $(A+B)*C(D/(J+D))$. These complex arithmetic operations can be converted into polish notation using stacks which then can be executed in two operands and an operator form.

Infix Expression:

It follows the scheme of **<operand><operator><operand>** i.e. an **<operator>** is preceded and succeeded by an **<operand>**. Such an expression is termed infix expression. E.g., **A+B**

Postfix Expression:

It follows the scheme of **<operand><operand><operator>** i.e. an **<operator>** is succeeded by both the **<operand>**. E.g., **AB+**

Steps to convert Infix To Postfix

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1. Push “(“ onto Stack, and add “)” to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered, then:
 1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
 2. Add operator to Stack.
[End of If]
6. If a right parenthesis is encountered, then:
 1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
 2. Remove the left Parenthesis.
[End of If]
[End of If]
7. END.

ALGORITHM

Step 1: Start.

Step 2: Create a stack to store operand and operator.

Step 3: In Postfix notation the operator follows the two operands and in the infix notation the operator is in between the two operands.

Step 4: Consider the sum of A and B. Apply the operator “+” to the operands A and B and write the sum as A+B is INFIX. + AB is PREFIX. AB+ is POSTFIX

Step 5: Get an Infix Expression as input and evaluate it by first converting it to postfix and then evaluating the postfix expression.

Step 6: The expressions with in innermost parenthesis must first be converted to postfix so that they can be treated as single operands. In this way Parentheses can be successively eliminated until the entire expression is converted.

Step 7: The last pair of parentheses to be opened with in a group of parentheses encloses the first expression with in that group to be transformed. This last-in first-out immediately suggests the use of Stack. Precedence plays an important role in the transforming infix to postfix.

Step 8: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 20
int top=-1;
```

```

char pop();
char stack[MAX];
void push(char item);

int pred(char symbol){
switch(symbol){
case '+':
case '-':return 2;
break;
case '*':
case '/':return 4;
break;

case '^':
case '$':return 6;
break;
case '(':
case ')':
case '#':return 1;
break;
}
}

int isoperator(char symbol){
switch(symbol){
case '+':
case '-':
case '*':
case '/':
case '^':
case '$':
case '(':

```



```

case ')':return 1;
break;
default:
return 0;
}
}
void convertip(char infix[],char postfix[]){
int i,symbol,j=0;
stack[++top]='#';
for(i=0;i<strlen(infix);i++){
symbol=infix[i];
if(isoperator(symbol)==0){
postfix[j]=symbol;
j++;
}
else{
if(symbol=='(')
push(symbol);
else if(symbol==')'){
while(stack[top]!='('){
postfix[j]=pop();
j++;
}
pop();//pop out (
}
else{
if(prcd(symbol)>prcd(stack[top]))
push(symbol);
else{
while(prcd(symbol)<=prcd(stack[top])){
postfix[j]=pop();

```



```

j++;
}
push(symbol);
} //end of else.
} //end of else.
} //end of else.
} //end of for.
while(stack[top]!='#'){
postfix[j]=pop();
j++;
}
postfix[j]='\0'; //null terminate string.
}

```

```

void main(){
char infix[20], postfix[20];
clrscr();
printf("Enter the valid infix string:\n");
gets(infix);
convertip(infix, postfix);
printf("The corresponding postfix string is:\n");
puts(postfix);
getch();
}
void push(char item){
top++;
stack[top]=item;
}
char pop(){
char a;
a=stack[top];

```




```
top--;  
return a;  
}
```

OUTPUT

Enter the valid infix string:

$(a+b)*c$

The corresponding postfix string is:

$ab+c*$

VIVA (PRE & POST LAB) QUESTIONS

1. What is the output of the following expression: $2\ 3\ 4\ 5\ +\ * -$
2. What is the maximum difference between number of operators and operands?
3. What is the advantage of postfix expression?
4. Which expression doesn't require parenthesis?
5. What is the output of the following expression: $+ * - 2\ 3\ 4\ 5$

RESULT

Thus the C program to convert infix to postfix expression using Stack was completed successfully.

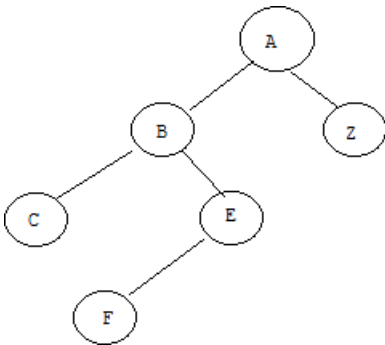
Ex.No:5 IMPLEMENTATION OF BINARY TREES AND OPERATIONS OF BINARY TREES

AIM

To write a C program to implement the binary trees and its operations.

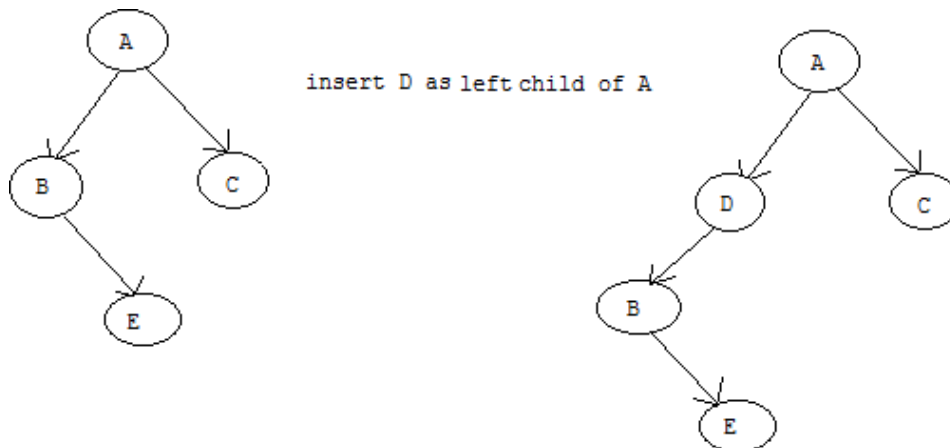
PRE LAB DISCUSSION

A **binary tree** is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.



Insertion:

In binary trees, a new node before insert has to specify 1) whose child it is going to be 2) mention whether new node goes as left/right child. For example(below image),



To add a new node to leaf node, a new node should also mention whether new node goes as left/right child.

Deletion:

For deletion, **only certain nodes** in a binary tree can be removed unambiguously. Suppose that the node to delete is node A. If A has no children, deletion is accomplished by setting the child of A's parent to null. If A has one child, set the parent of A's child to A's parent and set the child of A's parent to A's child. In a binary tree, a node with two children **cannot** be deleted unambiguously.

ALGORITHM

- Step 1: Start.
- Step 2: Create a Binary Tree for N elements.
- Step 3: Insert an element in binary tree
- Step 4: Traverse the tree in inorder.
- Step 5: Traverse the tree in preorder
- Step 6: Traverse the tree in postorder.
- Step 7: Stop

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct node
{
int data;
struct node *rlink;
struct node *llink;
}*tmp=NULL;
typedef struct node NODE;
NODE *create();
void preorder(NODE *);
void inorder(NODE *);
```



```

void postorder(NODE *);
void insert(NODE *);
void main()
{
int n,i,m;
clrscr();
do
{
printf("\n\n0.create\n\n1.insert \n\n2.preorder\n\n3.postorder\n\n4.inorder\n\n5.exit\n\n");
printf("\n\nEnter ur choice");
scanf("%d",&m);
switch(m)
{
case 0:
tmp=create();
break;
case 1:
insert(tmp);
break;
case 2:
printf("\n\nDisplay tree in Preorder traversal\n\n");
preorder(tmp);
break;
case 3:
printf("\n\nDisplay Tree in Postorder\n\n");
postorder(tmp);
break;
case 4:
printf("\n\nInorder\n\n");
inorder(tmp);
break;
case5:
exit(0);
}
}
while(n!=5);

getch();
}
void insert(NODE *root)
{
NODE *newnode;
if(root==NULL)
{
newnode=create();
root=newnode;
}
}

```



```

postorder(tmp->rlink);
printf(“%d->”,tmp->data);
}
}
void inorder(NODE *tmp)
{
if(tmp!=NULL)
{
inorder(tmp->llink);
printf(“%d->”,tmp->data);
inorder(tmp->rlink);
}
}
void preorder(NODE *tmp)
{
if(tmp!=NULL)
{
printf(“%d->”,tmp->data);
preorder(tmp->llink);
preorder(tmp->rlink);
}
}

```



OUTPUT

0.Create

1.insert

2.preorder

3.postorder

4.inorder

5.exit

Enter ur choice 0

Enter the Data 3

Enter ur choice 1

Enter the Data 7

Enter ur choice 1

Enter the Data 8

Enter ur choice 1

Enter the Data 6

Enter ur choice 1

Enter the Data 4

Enter ur choice2

Display tree in Preorder traversal

3->7->6->4->8

Enter ur choice3

Display tree in Postorder 4 -> 6 -> 8 -> 7 ->3

Enter ur choice4

Inorder

3 -> 4 -> 6 -> 7 -> 8

Enter ur choice 5



VIVA (PRE & POST LAB) QUESTIONS

1. What are the data structures used for BinaryTrees?
2. Explain implementation of traversal of a binarytree.
3. How to perform insertion into a binarytree.
4. Define post-ordertraversal.
5. Define in-ordertraversal.

RESULT

Thus the C program to implement binary tree and its operation was completed successfully.

Ex.No:6

IMPLEMENTATION OF BINARY SEARCH TREES

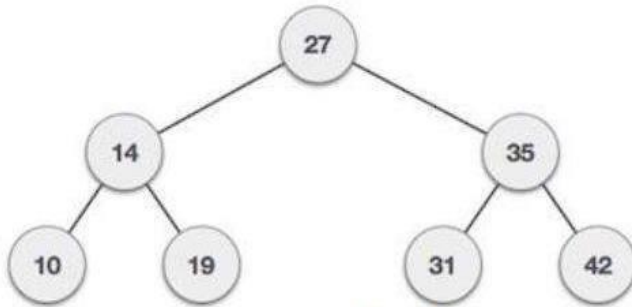
AIM

To write a C program to implement the binary search trees.

PRE LAB DISCUSSION

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
$$\text{left_subtree (keys)} \leq \text{node (key)} \leq \text{right_subtree (keys)}$$



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.
- Delete** – removes an existing node from the 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 the process.
- Step 2: Initialize and declare variables.
- Step 3: Construct the Tree
- Step 4: Data values are given which we call a key and a binary search tree
- Step 5: To search for the key in the given binary search tree, start with the root node and Compare the key with the data value of the root node. If they


```

case1:
    printf("\nEnter the data:");
    scanf("%d",&element);
    t=create(t,element);
    inorder(t);
    break;
case2:
    printf("\nEnter the data:");
    scanf("%d",&element);
    t=insert(t,element);
    inorder(t);
    break;
case3:
    printf("\nEnter the data:");
    scanf("%d",&element);
    t=del(t,element);
    inorder(t);
    break;
case4:
    printf("\nEnter the data:");
    scanf("%d",&element);
    temp=find(t,element);
    if(temp->data==element)
        printf("\nElement %d is at %d",element,temp);
    else
        printf("\nElement is not found");
    break;
case 5:

    temp=findmin(t);
    printf("\nMax element=%d",temp->data);
    break;
case6:
    temp=findmax(t);
    printf("\nMax element=%d",temp->data);
    break;
case7:
    inorder(t);
    break;
case8:
    preorder(t);
    break;
case9:
    postorder(t);
    break;
case 10:exit(0);

```

```

        }
    }while(ch<=10);
}

struct tree * create(struct tree *t, int element)
{
    t=(struct tree *)malloc(sizeof(structtree));
    t->data=element;
    t->lchild=NULL;
    t->rchild=NULL;
    return t;
}

struct tree * find(struct tree *t, int element)
{
    if(t==NULL)
        return NULL;
    if(element<t->data)
        return(find(t->lchild,element));
    else
        if(element>t->data)
            return(find(t->rchild,element));
        else
            return t;
}

struct tree *findmin(struct tree *t)
{
    if(t==NULL)
        return NULL;
    else
        if(t->lchild==NULL)
            return t;
        else
            return(findmin(t->lchild));
}

struct tree *findmax(struct tree *t)
{
    if(t!=NULL)
    {
        while(t->rchild!=NULL)
            t=t->rchild;
    }
    return t;
}

```



```

struct tree *insert(struct tree *t,int element)
{
    if(t==NULL)
    {
        t=(struct tree *)malloc(sizeof(struct tree));
        t->data=element;
        t->lchild=NULL;
        t->rchild=NULL;
        return t;
    }
    else
    {
        if(element<t->data)
        {
            t->lchild=insert(t->lchild,element);
        }
        else
            if(element>t->data)
            {
                t->rchild=insert(t->rchild,element);
            }
            else
                if(element==t->data)
                {
                    printf("element already present\n");
                }
                return t;
            }
    }
}

```

```

struct tree * del(struct tree *t, int element)
{
    if(t==NULL)
        printf("element not found\n");
    else
        if(element<t->data)
            t->lchild=del(t->lchild,element);
        else
            if(element>t->data)
                t->rchild=del(t->rchild,element);
            else
                if(t->lchild&& t->rchild)
                {
                    temp=findmin(t->rchild);
                    t->data=temp->data;
                }
            }
    }
}

```

```

        t->rchild=del(t->rchild,t->data);
    }
    else
    {
        temp=t;
        if(t->lchild==NULL)
            t=t->rchild;
        else
            if(t->rchild==NULL)
                t=t->lchild;
            free(temp);
    }
    return t;
}

```

```

void inorder(struct tree *t)
{
    if(t==NULL)
        return;
    else
    {
        inorder(t->lchild);
        printf("%t%d",t->data);
        inorder(t->rchild);
    }
}

```

```

void preorder(struct tree *t)
{
    if(t==NULL)
        return;
    else
    {
        printf("%t%d",t->data);
        preorder(t->lchild);
        preorder(t->rchild);
    }
}

```

```

void postorder(struct tree *t)
{
    if(t==NULL)
        return;
    else
    {
        postorder(t->lchild);

```



```

        postorder(t->rchild);
        printf("\t%d",t->data);
    }
}

```

OUTPUT

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :1

Enter the data:10

10

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :2

Enter the data:20

10 20

BINARY SEARCH TREE



Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :2

Enter the data:30

10 20 30

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :2

Enter the data:25

10 20 25 30

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin



6.FindMax
7.Inorder
8.Preorder
9.Postorder
10.Exit
Enter ur choice :4

Enter the data:25

Element 25 is at 2216

BINARY SEARCH TREE

Main Menu

1.Create
2.Insert
3.Delete
4.Find
5.FindMin
6.FindMax
7.Inorder
8.Preorder
9.Postorder
10.Exit

Enter ur choice :5

Max element=10

BINARY SEARCH TREE

Main Menu

1.Create
2.Insert
3.Delete
4.Find
5.FindMin
6.FindMax
7.Inorder
8.Preorder
9.Postorder
10.Exit

Enter ur choice :6

Max element=30

BINARY SEARCH TREE

Main Menu



- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :7

10 20 25 30

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :8

10 20 30 25

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :9

25 30 20 10

BINARY SEARCH TREE



Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :3

Enter the data:10

20 25 30

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit



Enter ur choice :10

VIVA (PRE & POST LAB) QUESTIONS

1. Write the various operation performed in the binary searchtree?
2. How many nodes will be there in given nthlevel?
3. Write the necessary condition for inserting element intoBST
4. How will you find the maximum and minimum element inBST?
5. How will you perform deletion in BST with 2children

RESULT

Thus the C program to implement the binary search trees was completed successfully.

Ex.No:7

IMPLEMENTATION OF AVL TREES

AIM

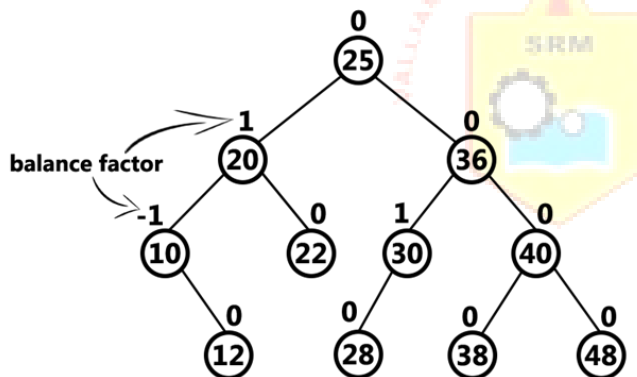
To write a C program to implement AVL trees.

PRE LAB DISCUSSION

AVL tree is a self balanced binary search tree. That means, an AVL tree is also a binary search tree but it is a balanced tree. A binary tree is said to be balanced, if the difference between the heights of left and right subtrees of every node in the tree is either -1, 0 or +1. In other words, a binary tree is said to be balanced if for every node, height of its children differ by at most one. In an AVL tree, every node maintains an extra information known as **balance factor**. An AVL tree is defined as follows...

An AVL tree is a balanced binary search tree. In an AVL tree, balance factor of every node is either -1, 0 or +1.

Balance factor of a node is the difference between the heights of left and right subtrees of that node. The balance factor of a node is calculated either **height of left subtree - height of right subtree** (OR) **height of right subtree - height of left subtree**.



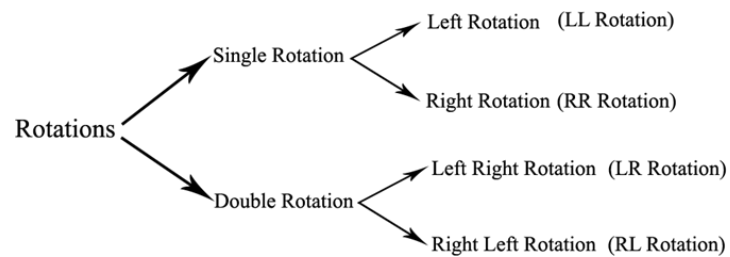
AVL Tree Rotations

In AVL tree, after performing every operation like insertion and deletion we need to check the balance factor of every node in the tree. If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced. We use **rotation** operations to make the tree balanced whenever the tree is becoming imbalanced due to any operation.

Rotation operations are used to make a tree balanced.

Rotation is the process of moving the nodes to either left or right to make tree balanced.

There are **four** rotations and they are classified into **two** types.



ALGORITHM

Step 1: Start

Step 2: Insert the new element into the tree using Binary Search Tree insertion logic.

Step 3: After insertion, check the Balance Factor of every node.

- If the Balance Factor of every node is 0 or 1 or -1 then go for next operation.
- If the Balance Factor of any node is other than 0 or 1 or -1 then tree is said to be imbalanced. Then perform the suitable Rotation to make it balanced. And go for next operation.

Step 4: Compare, the search element with the value of root node in the tree.

If search element is larger, then continue the search process in right subtree.

If we reach to the node with search value, then display "Element is found" and terminate the function.

Step 5: Stop

PROGRAM

```
#include<stdio.h>
#include<malloc.h>
typedef enum { FALSE ,TRUE } ;
struct node
{
    int info;
    int balance;
    struct node *lchild;
    struct node *rchild;
};
struct node *insert (int , struct node *, int *);
struct node* search(struct node *,int);

struct node* search(struct node *ptr,int info)
{
    if(ptr!=NULL)
        if(info < ptr->info)
            ptr=search(ptr->lchild,info);
        else if( info > ptr->info)
            ptr=search(ptr->rchild,info);
    return(ptr);
}/*End of search()*/

struct node *insert (int info, struct node *pptr, int *ht_inc)
{
    struct node *aptr;
    struct node *bptr;
    if(pptr==NULL)
    {
        pptr = (struct node *) malloc(sizeof(structnode));
        pptr->info =info;
        pptr->lchild = NULL;
        pptr->rchild = NULL;
        pptr->balance = 0;
        *ht_inc = TRUE;
        return (pptr);
    }
    if(info < pptr->info)
    {
        pptr->lchild = insert(info, pptr->lchild, ht_inc);
        if(*ht_inc==TRUE)
        {
            switch(pptr->balance)
            {
                case -1: /* Right heavy */
```

```

        pptr->balance = 0;
        *ht_inc = FALSE;
        break;
    case 0: /* Balanced */
        pptr->balance = 1;
        break;
    case 1: /* Left heavy */
        aptr = pptr->lchild;
        if(aptr->balance == 1)
        {
            printf("Left to Left Rotation\n");
            pptr->lchild= aptr->rchild;
            aptr->rchild = pptr;
            pptr->balance = 0;
            aptr->balance=0;
            pptr = aptr;
        }
        else
        {
            printf("Left to righrotation\n");
            bptr =aptr->rchild;
            aptr->rchild =bptr->lchild;
            bptr->lchild =aptr;
            pptr->lchild =bptr->rchild;
            bptr->rchild =pptr;
            if(bptr->balance == 1 )
                pptr->balance = -1;
            else
                pptr->balance = 0;
            if(bptr->balance == -1)
                aptr->balance = 1;
            else
                aptr->balance = 0;
            bptr->balance=0;
            pptr=bptr;
        }
        *ht_inc = FALSE;
    } /*End of switch */
} /*End of if */
} /*End of if*/
if(info > pptr->info)
{
    pptr->rchild = insert(info, pptr->rchild, ht_inc);
    if(*ht_inc==TRUE)
    {

```

```

switch(pptr->balance)
{
case 1: /* Left heavy */
    pptr->balance = 0;
    *ht_inc = FALSE;
    break;
case 0: /* Balanced */
    pptr->balance = -1;
    break;
case -1: /* Right heavy */
    aptr = pptr->rchild;
    if(aptr->balance == -1)
    {
        printf("Right to Right Rotation\n");
        pptr->rchild= aptr->lchild;
        aptr->lchild = pptr;
        pptr->balance = 0;
        aptr->balance=0;
        pptr = aptr;
    }
    else
    {
        printf("Right to LeftRotation\n");
        bptr =aptr->lchild;
        aptr->lchild =bptr->rchild;
        bptr->rchild =aptr;
        pptr->rchild =bptr->lchild;
        bptr->lchild =pptr;
        if(bptr->balance == -1)
            pptr->balance = 1;
        else
            pptr->balance = 0;
        if(bptr->balance == 1)
            aptr->balance = -1;
        else
            aptr->balance = 0;
        bptr->balance=0;
        pptr = bptr;
    }
    /*End of else*/
    *ht_inc = FALSE;
} /*End of switch */
} /*End of if*/
} /*End of if*/

return(pptr);
} /*End of insert()*/

```

```

void display(struct node *ptr,int level)
{
    int i;
    if ( ptr!=NULL )
    {
        display(ptr->rchild, level+1);
        printf("\n");
        for (i = 0; i < level; i++)
            printf(" ");
        printf("%d", ptr->info);
        display(ptr->lchild, level+1);
    }/*End of if*/
}/*End of display()*/

void inorder(struct node *ptr)
{
    if(ptr!=NULL)
    {
        inorder(ptr->lchild);
        printf("%d ",ptr->info);
        inorder(ptr->rchild);
    }
}/*End of inorder()*/
main()
{
    int ht_inc;
    int info ;
    int choice;
    struct node *root = (struct node *)malloc(sizeof(struct node));
    root = NULL;

    while(1)
    {
        printf("1.Insert\n");
        printf("2.Display\n");
        printf("3.Quit\n");
        printf("Enter your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                printf("Enter the value to be inserted : ");
                scanf("%d", &info);
                if( search(root,info) == NULL )
                    root = insert(info, root, &ht_inc);

```



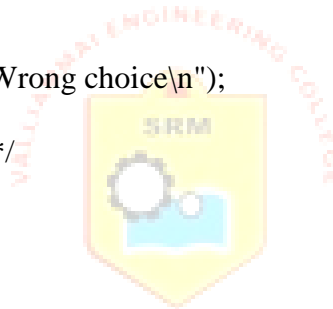

```

else
    printf("Duplicate value ignored\n");
break;
case 2:

if(root==NULL)
{
    printf("Tree is empty\n");
    continue;
}
printf("Tree is :\n");
display(root, 1);
printf("\n\n");
printf("Inorder Traversal is: ");
inorder(root);
printf("\n");
break;
case 3:
    exit(1);
default:
    printf("Wrong choice\n");

}/*End of switch*/
}/*End of while*/
}/*End of main()*/

```



OUTPUT

```

DOS
BOX DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0,
1.Insert
2.Display
3.Quit
Enter your choice : 1
Enter the value to be inserted : 7
1.Insert
2.Display
3.Quit
Enter your choice : 1
Enter the value to be inserted : 12
1.Insert
2.Display
3.Quit
Enter your choice : 1
Enter the value to be inserted : 9
Right to Left Rotation
1.Insert
2.Display
3.Quit

```

```
Enter your choice : 1
Enter the value to be inserted : 4
1.Insert
2.Display
3.Quit
Enter your choice : 2
Tree is :
    12
   9
  7
 4
Inorder Traversal is: 4 7 9 12
1.Insert
2.Display
3.Quit
Enter your choice : 3_
```

VIVA (PRE & POST LAB) QUESTIONS

1. How is an AVL tree better than a binary searchtree?
2. Create an AVL tree using the following sequence of data: 16, 27, 9, 11, 36, 54.
3. Given an empty AVL tree, how would you construct AVL tree when a set of numbers are given without performing any rotations?
4. Which rotation is done when the new node is inserted in the right sub-tree of the right sub-tree of the critical node?
5. How to insert a new node into an AVL tree.

RESULT

Thus the C program to implement AVL tree was completed successfully.

Ex.No:8

IMPLEMENTATION OF HEAPS USING PRIORITY QUEUES

AIM

To write a C program to implement the heaps using priority queues

PRE LAB DISCUSSION

A Priority Queue is an abstract data type to efficiently support finding an item with the highest priority across a series of operations. The basic operations are :

1. Insert
2. Find - minimum (or maximum),and
3. Delete-minimum (or maximum).

A heap is a specialized tree-based data structure that satisfies the heap property: if B is a child node of A, then $\text{key}(A) \geq \text{key}(B)$. This implies that an element with the greatest key is always in the root node, and so such a heap is sometimes called a max-heap. (Alternatively, if the comparison is reversed, the smallest element is always in the root node, which results in a min-heap). Heaps are usually implemented in an array, and don't require pointers between elements.

The operations commonly performed with a heap are :

- delete-max or delete-min : removing the root node of a max or min heap, respectively.
- increase-key or decrease-key : updating a key within a max or min heap, respectively.
- insert : adding a new key to the heap.
- merge : joining two heaps to form a valid new heap containing all the elements of both.

ALGORITHM

Step 1: Start

Step 2: Declaring the maximum size of the queue

Step 3: Lower bound of the array is initialized to 0

Step 4: Inserts an element in the queue

Step 5: Deletes an element from the queue

Step 6: Display the queue

Step 7 :Stop

PROGRAM

```
#include<stdio.h>
#include<math.h>
#define MAX 100/*Declaring the maximum size of the queue*/
void swap(int*,int*);
main()
{
int choice,num,n,a[MAX],data,s,lb;
void display(int[],int);
void insert(int[],int,int,int);
int del_hi_piori(int[],int,int);
n=0;/*Represents number of nodes in the queue*/
lb=0;/*Lower bound of the array is initialized to 0*/
while(1)
{
printf("\n.....MAINMENU ..... ");
printf("\n1.Insert.n");
printf("\n2.Delete.n");
printf("\n3.Display.n");
printf("\n4.Quit.n");
printf("\nEnter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1:/*choice to accept an element and insert it in the queue*/
printf("Enter data to be inserted : ");
scanf("%d",&data);
insert(a,n,data,lb);
n++;
break;
case2:
s=del_hi_piori(a,n+1,lb);
if(s!=0)
printf("\nThe deleted value is : %d n",s);
if(n>0)
n--;
```

```

break;
case 3:/*choice to display the elements of the queue*/
printf("n");
display(a,n);
break;
case 4:/*choice to exit from the program*/
return;
default:
printf("Invalid choice.n");
}
printf("nn");
}
}
/*This function inserts an element in the queue*/
void insert(int a[],int heapsize,int data,int lb)
{
int i,p;
int parent(int);
if(heapsize==MAX)
{
printf("Queue Is Full!!n");
return;
}
i=lb+heapsize;
a[i]=data;
while(i>lb&& a[p=parent(i)]<a[i])
{
swap(&a[p],&a[i]);
i=p;
}
}
/*This function deletes an element from the queue*/
int del_hi_priori(int a[],int heapsize,int lb)
{
int data,i,l,r,max_child,t;
int left(int);
int right(int);
if(heapsize==1)
{
printf("Queue Is Empty!!n");
return 0;
}
t=a[lb];
swap(&a[lb],&a[heapsize-1]);
i=lb;
heapsize--;

```



```

while(1)
{
if((l=left(i))>=heapsize)
break;
if((r=right(i))>=heapsize)
max_child=l;
else
max_child=(a[l]>a[r]?l:r;
if(a[i]>=a[max_child])
break;
swap(&a[i],&a[max_child]);
i=max_child;
}
return t;
}
/*Returns parent index*/
int parent(int i)
{
float p;
p=((float)i/2.0)-1.0;
return ceil(p);
}
/*Returns leftchild index*/
int left(int i)
{
return 2*i+1;
}
/*Returns rightchild index*/
int right(int i)
{
return 2*i+2;
}
/*This function displays the queue*/
void display(int a[],int n)
{
int i;
if(n==0)
{
printf("Queue Is Empty!!\n");
return;
}
for(i=0;i<n;i++)
printf("%d ",a[i]);
printf("\n");
}
/*This function is used to swap two elements*/

```



```
void swap(int*p,int*q)
{
int temp;
temp=*p;
*p=*q;
*q=temp;
}
```

OUTPUT

.....MAIN MENU.....

1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 1
Enter data to be inserted : 52

.....MAIN MENU.....

1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 1
Enter data to be inserted : 63

.....MAIN MENU.....

1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 1
Enter data to be inserted : 45

.....MAIN MENU.....

1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 1
Enter data to be inserted : 2

.....MAIN MENU.....



1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 1
Enter data to be inserted : 99

.....MAIN MENU.....

1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 3
99 63 45 2 52

.....MAIN MENU.....

1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 2
The deleted value is : 99

.....MAIN MENU.....

1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 3
63 52 45 2

.....MAIN MENU.....

1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 2
The deleted value is : 63

.....MAIN MENU.....

1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 2
The deleted value is : 52



.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 3

45 2

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 4

VIVA (PRE & POST LAB) QUESTIONS

1. How to insert a node based on position in a priorityqueue?
2. How to delete a minimum element in priority queue usingheap?
3. In a max-heap, element with the greatest key is always in the whichnode?
4. Heaps are excellent data structures to implement priority queues. Justify thisstatement
5. What do you mean by min-heap andmax-heap.

RESULT

Thus the 'C' program to implement priority queue using heaps was executed successfully.

Ex.No:9.1

DEPTH FIRST SEARCH

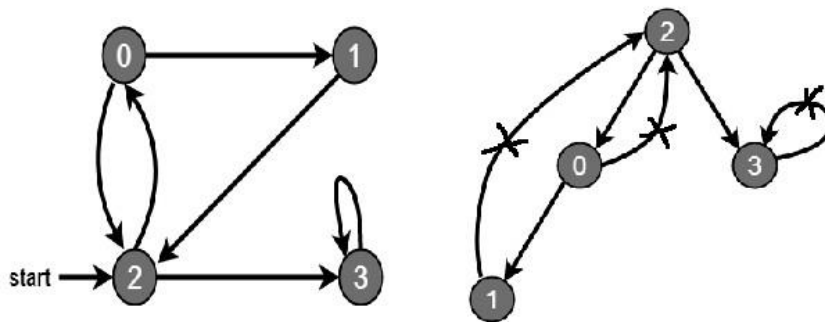
AIM

To write a C program for implementing the traversal algorithm for Depth first traversal

PRE LAB DISCUSSION

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array.

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.



ALGORITHM

- Step 1: Start from any vertex, say V_i .
- Step 2: V_i is visited and then all vertices adjacent to V_i are traversed recursively using DFS.
- Step 3: Since, a graph can have cycles. We must avoid revisiting a node. To do this, when we visit a vertex V , we mark it visited.

Step 4: A node that has already been marked as visited should not be selected for traversal.

Step 5: Marking of visited vertices can be done with the help of a global array visited[].

Step 6: Array visited[] is initialized to false (0).

PROGRAM

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

typedef structnode
{
    struct node *next;
    int vertex;
}node;

node *G[20];
//heads of linked list
int visited[20];
int n;
void read_graph();
//create adjacency list
voidinsert(int,int);
//insert an edge (vi,vj) in te adjacency list
void DFS(int);

void main()
{
    int i;
    read_graph();
    //initialised visited to 0

    for(i=0;i<n;i++)
        visited[i]=0;

    DFS(0);
}

void DFS(int i)
{
    node *p;

    printf("\n%d",i);
    p=G[i];
    visited[i]=1;
```



```

while(p!=NULL)
{
    i=p->vertex;

    if(!visited[i])
        DFS(i);
    p=p->next;
}
}

void read_graph()
{
    int i,vi,vj,no_of_edges;
    printf("Enter number of vertices:");

    scanf("%d",&n);

    //initialise G[] with a null

    for(i=0;i<n;i++)
    {
        G[i]=NULL;
        //read edges and insert them in G[]

        printf("Enter number of edges:");
        scanf("%d",&no_of_edges);

        for(i=0;i<no_of_edges;i++)
        {
            printf("Enter an edge(u,v):");
            scanf("%d%d",&vi,&vj);
            insert(vi,vj);
        }
    }
}

void insert(int vi,int vj)
{
    node *p,*q;

    //acquire memory for the new node
    q=(node*)malloc(sizeof(node));
    q->vertex=vj;
    q->next=NULL;

    //insert the node in the linked list number vi

```



```

if(G[vi]==NULL)
    G[vi]=q;
else
{
    //go to end of the linked list
    p=G[vi];

    while(p->next!=NULL)
        p=p->next;
    p->next=q;
}
}

```

OUTPUT

```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter number of vertices:8
Enter number of edges:10
Enter an edge(u,v):0 1
Enter an edge(u,v):0 2
Enter an edge(u,v):0 3
Enter an edge(u,v):0 4
Enter an edge(u,v):1 5
Enter an edge(u,v):2 5
Enter an edge(u,v):3 6
Enter an edge(u,v):4 6
Enter an edge(u,v):5 7
Enter an edge(u,v):6 7

0
1
5
7
2
3
6
4Enter number of vertices:

```

VIVA (PRE & POST LAB) QUESTIONS

1. A person wants to visit some places. He starts from a vertex and then wants to visit every vertex till it finishes from one vertex, backtracks and then explore other vertex from same vertex. What algorithm he should use?
2. When the Depth First Search of a graph is unique?
3. In Depth First Search, how many times a node is visited?
4. Give the applications of DFS.
5. Depth First Search is equivalent to which of the traversal in the Binary Trees?

RESULT

Thus the C program for implementing the traversal algorithm for Depth first traversal was implemented successfully.

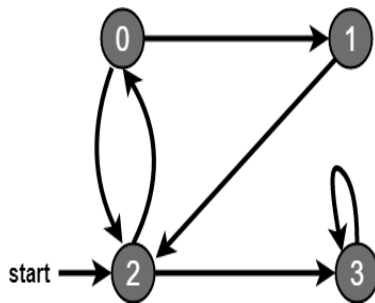
Ex.No:9.2**BREADTH FIRST SEARCH****AIM**

To write a C program for implementing the traversal algorithm for Breadth first traversal

PRE LAB DISCUSSION

Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.

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 Breadth First Traversal of the following graph is 2, 0, 3, 1.

**PRE LAB QUESTIONS**

1. What is BFS?
2. What is the application of BFS?
3. How does BFS work?
4. Breadth First Search is equivalent to which of the traversal in the Binary Trees?
5. What are the advantages and disadvantages of BFS?

ALGORITHM

- Step 1: Start from any vertex, say V_i .
- Step 2: V_i is visited and then all vertices adjacent to V_i are traversed recursively using BFS.
- Step 3: avoid revisiting a node. To do this, when we visit a vertex V , we mark it visited.
- Step 4: A node that has already been marked as visited should not be selected for traversal.
- Step 5: Marking of visited vertices can be done with the help of a global array visited [].
- Step 6: Array visited [] is initialized to false (0).

PROGRAM

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

#define MAX 100

#define initial 1
#define waiting 2
#define visited 3

int n;
intadj[MAX][MAX];
intstate[MAX];
void create_graph();
void BF_Traversal();
void BFS(intv);

int queue[MAX], front = -1, rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();

int main()
{
    create_graph();
    BF_Traversal();
    return 0;
}

void BF_Traversal()
{
    int v;

    for(v=0; v<n; v++)
        state[v] = initial;

    printf("Enter Start Vertex for BFS: \n");
    scanf("%d", &v);
    BFS(v);
}

void BFS(int v)
{
    int i;
```



```

insert_queue(v);
state[v] = waiting;

while(!isEmpty_queue())
{
    v = delete_queue( );
    printf("%d ",v);
    state[v] =visited;

    for(i=0; i<n;i++)
    {
        if(adj[v][i] == 1 && state[i] == initial)
        {
            insert_queue(i);
            state[i] = waiting;
        }
    }
}
printf("\n");
}

```

```

void insert_queue(int vertex)
{
    if(rear == MAX-1)
        printf("Queue Overflow\n");
    else
    {
        if(front == -1)
            front = 0;
        rear = rear+1;
        queue[rear] = vertex ;
    }
}

```

```

int isEmpty_queue()
{
    if(front == -1 || front > rear)
        return 1;
    else
        return 0;
}

```

```

intdelete_queue()
{
    intdelete_item;
}

```




```

if(front == -1 || front > rear)
{
    printf("Queue Underflow\n");
    exit(1);
}

delete_item = queue[front];
front = front+1;
return delete_item;
}

void create_graph()
{
    int count,max_edge,origin,destin;

    printf("Enter number of vertices : ");
    scanf("%d",&n);
    max_edge = n*(n-1);

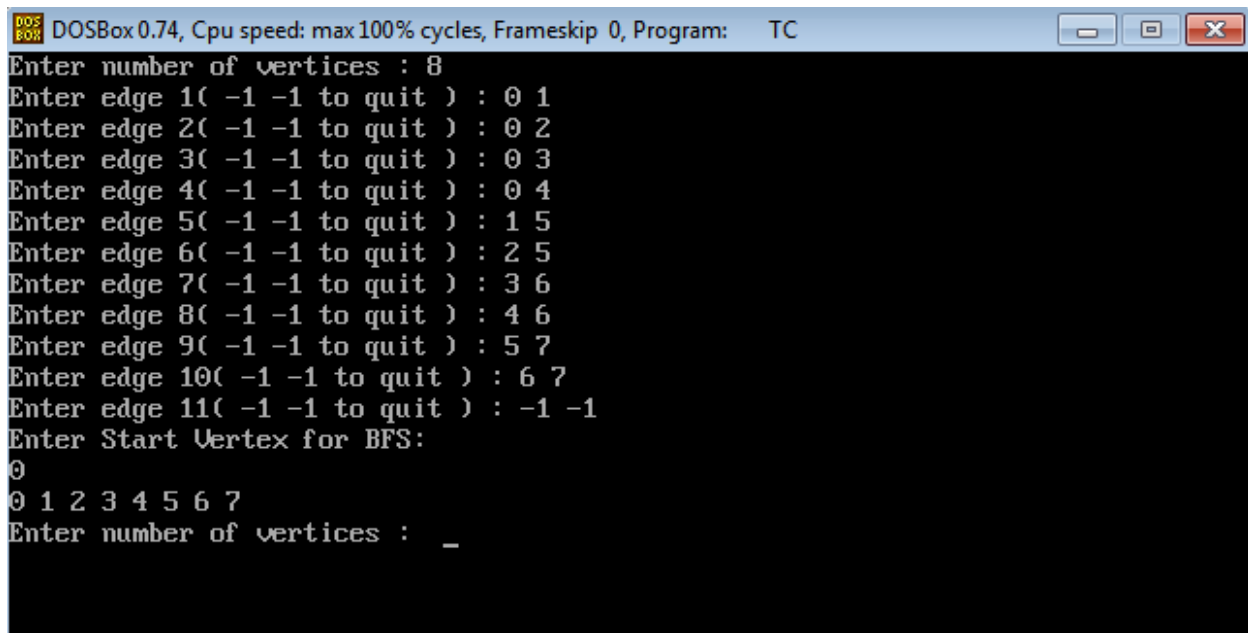
    for(count=1; count<=max_edge; count++)
    {
        printf("Enter edge %d( -1 -1 to quit ) : ",count);
        scanf("%d %d",&origin,&destin);

        if((origin == -1) && (destin == -1))
            break;

        if(origin>=n || destin>=n || origin<0 || destin<0)
        {
            printf("Invalid edge!\n");
            count--;
        }
        else
        {
            adj[origin][destin] = 1;
        }
    }
}

```

OUTPUT



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter number of vertices : 8
Enter edge 1( -1 -1 to quit ) : 0 1
Enter edge 2( -1 -1 to quit ) : 0 2
Enter edge 3( -1 -1 to quit ) : 0 3
Enter edge 4( -1 -1 to quit ) : 0 4
Enter edge 5( -1 -1 to quit ) : 1 5
Enter edge 6( -1 -1 to quit ) : 2 5
Enter edge 7( -1 -1 to quit ) : 3 6
Enter edge 8( -1 -1 to quit ) : 4 6
Enter edge 9( -1 -1 to quit ) : 5 7
Enter edge 10( -1 -1 to quit ) : 6 7
Enter edge 11( -1 -1 to quit ) : -1 -1
Enter Start Vertex for BFS:
0
0 1 2 3 4 5 6 7
Enter number of vertices : _
```

VIVA (PRE & POST LAB) QUESTIONS

1. Which data structure is used in standard implementation of Breadth First Search?
2. A person wants to visit some places. He starts from a vertex and then wants to visit every place connected to this vertex and so on. What algorithm he should use?
3. In BFS, how many times a node is visited?
4. Regarding implementation of Breadth First Search using queues, what is the maximum distance between two nodes present in the queue?
5. When the Breadth First Search of a graph is unique?

RESULT

Thus the C program for implementing the traversal algorithm for Breadth first traversal was implemented successfully

Ex.No:10

APPLICATIONS OF GRAPHS

AIM

To write a C program to implement the shortest path using dijkstra's algorithm.

PRE LAB DISCUSSION

Dijkstra's algorithm has many variants but the most common one is to find the shortest paths from the source vertex to all other vertices in the graph.

Steps:

- Set all vertices distances = infinity except for the source vertex, set the sourcedistance = 0.
- Push the source vertex in a min-priority queue in the form (distance , vertex), as the comparison in the min-priority queue will be according to verticesdistances.
- Pop the vertex with the minimum distance from the priority queue (at first thepopped vertex =source).
- Update the distances of the connected vertices to the popped vertex in case of "current vertex distance + edge weight < next vertex distance", then push thevertex with the new distance to the priority queue.
- If the popped vertex is visited before, just continue without usingit.
- Apply the same algorithm again until the priority queue isempty.

ALGORITHM

Step 1:Select the source node also called the initial node

Step 2: Define an empty set N that will be used to hold nodes to which a shortest path has been found.

Step 3: Label the initial node with , and insert it intoN.

Step 4: Repeat Steps 5 to 7 until the destination node is in N or there are no more labelled nodes in N.

Step 5: Consider each node that is not in N and is connected by an edge from the newly inserted node.

Step 6. (a) If the node that is not in N has no label then SET the label of the node = the label of the newly inserted node + the length of the edge.

(b) Else if the node that is not in N was already labelled, then SET its new label = minimum

Step 7: Pick a node not in N that has the smallest label assigned to it and add it to N.

PROGRAM

```
#include "stdio.h"
#include "conio.h"
#define infinity 999

void dij(int n,int v,int cost[10][10],int dist[])
{
    int i,u,count,w,flag[10],min;
    for(i=1;i<=n;i++)
        flag[i]=0,dist[i]=cost[v][i];
    count=2;
    while(count<=n)
    {
        min=99;
        for(w=1;w<=n;w++)
            if(dist[w]<min && !flag[w])
                min=dist[w],u=w;
        flag[u]=1;
        count++;
        for(w=1;w<=n;w++)
            if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
                dist[w]=dist[u]+cost[u][w];
    }
}

void main()
{
    int n,v,i,j,cost[10][10],dist[10];
    clrscr();
    printf("\n Enter the number of nodes:");
    scanf("%d",&n);
    printf("\n Enter the cost matrix:");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=infinity;
        }
    printf("\n Enter the source matrix:");
    scanf("%d",&v);
```



```

dij(n,v,cost,dist);
printf("\n Shortest path:\n");
for(i=1;i<=n;i++)
if(i!=v)
printf("%d->%d,cost=%d\n",v,i,dist[i]);
getch();
}

```

OUTPUT

```

DOS
BOX
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

Enter the number of nodes:3

Enter the cost matrix:0 4 6
3 0 4
6 5 0

Enter the source matrix:1

Shortest path:
1->2,cost=4
1->3,cost=6
_

```

VIVA (PRE & POST LAB) QUESTIONS

1. How to find the adjacency matrix?
2. Which algorithm solves the single pair shortest path problem?
3. How to find shortest distance from source vertex to target vertex?
4. What is the maximum possible number of edges in a directed graph with no self loops having 8 vertices?
5. Does Dijkstra's Algorithm work for both negative and positive weights?

RESULT

Thus the C program to implement shortest path was completed successfully.

Ex.No:11.1**LINEARSEARCH****AIM**

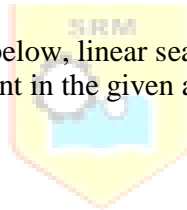
To write a C program to implement the concept of linear search.

PRE LAB DISCUSSION

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

Linear Search

To search the number 33 in the array given below, linear search will go step by step in a sequential order starting from the first element in the given array.

**ALGORITHM**

- Step 1: Start with the first item in the list.
- Step 2: Compare the current item to the target
- Step 3: If the current value matches the target then we declare victory and stop.
- Step 4: If the current value is less than the target then set the current item to be the next item and repeat from 2.
- Step 5: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int arr[20];
    int i,size,sech;
    printf("\n\t-- Linear Search --\n\n");
    printf("Enter total no. of elements : ");
    scanf("%d",&size);
    for(i=0; i<size; i++)
    {
        printf("Enter %d element : ",i+1);
        scanf("%d",&arr[i]);
    }
    printf("Enter the element to be searched:");
    scanf("%d",&sech);
    for(i=0; i<size; i++)
    {
        if(sech==arr[i])
        {
            printf("Element exits in the list at position : %d",i+1);
            break;
        }
    }
    getch();
}
```



OUTPUT

-- Linear Search --

```
Enter total no. of elements : 5
Enter 1 element : 10
Enter 2 element : 4
Enter 3 element : 2
Enter 4 element : 17
Enter 5 element : 100
Enter the element to be searched: 17
Element exits in the list at position : 4
```

VIVA (PRE & POST LAB) QUESTIONS

1. Given a list of numbers $a[] = \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21\}$. Search for value 19 using linear search technique.
2. During linear search, when the record is present in first position then how many comparisons are made
3. Is a linear search faster than a binary search?
4. What is the best case for linear search?
5. During linear search, when the record is present in last position then how many comparisons are made?



RESULT

Thus the C program to implement linear search was executed successfully.

Ex.No:11.2**BINARY SEARCH****AIM**

To write a C program to implement the concept of binary search.

PRE LAB DISCUSSION

Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

If searching for 23 in the 10-element array:

**ALGORITHM**

- Step 1: Set the list to be the whole list
- Step 2: Find the middle value of the list
- Step 3: If the middle value is equal to the target then we declare victory and stop.
- Step 4: If the middle item is less than the target, then we set the new list to be the upper half of the old list and we repeat from step 2 using the new list.
- Step 5: If the middle value is greater than the target, then we set the new list to be the bottom half of the list, and we repeat from step 2 with the new list.
- Step 6: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main(){
int n,i,search,f=0,low,high,mid,a[20];
clrscr();
printf("Enter the nvalue:");
scanf("%d",&n);
for(i=1;i<=n;i++){
printf("Enter the number in ascending order a[%d]=",i);
scanf("%d",&a[i]);
}
printf("Enter the search element:");
scanf("%d",&search);
low=1;
high=n;
while(low<=high){
mid=(low+high)/2;
if(search<a[mid]){
high=mid-1;
}
elseif(search>a[mid]){
low=mid+1;
}
else{
f=1;
printf("obtained in the position %d:",mid);
getch();
exit();
}
}
if(f==0)
printf("not present");
getch();
}
```



OUTPUT

```
Enter the n value:5
Enter the number in ascending order a[1]=10
Enter the number in ascending order a[2]=8
Enter the number in ascending order a[3]=9
Enter the number in ascending order a[4]=24
Enter the number in ascending order a[5]=1
Enter the search element:9
obtained in the position 3:
```

VIVA (PRE & POST LAB) QUESTIONS

1. Given an array `arr = {5,6,77,88,99}` and `key = 88`; How many iterations are done until the element is found?
2. Which technique of searching an element in an array would you prefer to use?
3. What does the following piece of code do?

```
{  
    f=1;  
    printf("obtained in the position %d:",mid);  
}
```

4. What is the best case for binary search?



RESULT

Thus the C program to implement the binary search was completed successfully.

Ex.No:11.3**BUBBLE SORT****AIM**

To write a C program to implement the concept of bubble sort.

PRE LAB DISCUSSION

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example:**First Pass:**

(**5 1 4 2 8**) \rightarrow (**1 5 4 2 8**), Here, algorithm compares the first two elements, and swaps since $5 > 1$.

(**1 5 4 2 8**) \rightarrow (**1 4 5 2 8**), Swap since $5 > 4$

(**1 4 5 2 8**) \rightarrow (**1 4 2 5 8**), Swap since $5 > 2$

(**1 4 2 5 8**) \rightarrow (**1 4 2 5 8**), Now, since these elements are already in order ($8 > 5$), algorithm does not swap them.

Second Pass:

(**1 4 2 5 8**) \rightarrow (**1 4 2 5 8**)

(**1 4 2 5 8**) \rightarrow (**1 2 4 5 8**), Swap since $4 > 2$

(**1 2 4 5 8**) \rightarrow (**1 2 4 5 8**)

(**1 2 4 5 8**) \rightarrow (**1 2 4 5 8**)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:

(**1 2 4 5 8**) \rightarrow (**1 2 4 5 8**)

(**1 2 4 5 8**) \rightarrow (**1 2 4 5 8**)

(**1 2 4 5 8**) \rightarrow (**1 2 4 5 8**)

(**1 2 4 5 8**) \rightarrow (**1 2 4 5 8**)

ALGORITHM

Step 1: Start.
Step 2: Repeat Steps 3 and 4 for i=1 to 10
Step 3: Set j=1
Step 4: Repeat while j<=n

(A) if a[i] <a[j]

Then interchange a[i] and a[j]

[End of if]

(B) Set j = j+1

[End of InnerLoop]

[End of Step 1 Outer Loop]

Step 5: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main(){
int n, i, j, temp , a[100];
printf("Enter the total integers you want to enter (make it less than 100):\n");
scanf("%d",&n);
printf("Enter the %d integer array elements:\n",n);
for(i=0;i<n;i++){
scanf("%d",&a[i]);
}
for(i=0;i<n-1;i++){
for(j=0;j<n-i-1;j++){
if(a[j+1]<a[j]){
temp = a[j];
a[j] = a[j+1];
a[j+1] =temp;
}
}
}
printf("The sorted numbers are:");
```

```
for(i=0;i<n;i++){
printf("%3d",a[i]);
}
getch();
}
```

OUTPUT

Enter the total integers you want to enter (make it less than 100):

5

Enter the 5 integer array elements:

99

87

100

54

150

The sorted numbers are: 54 87 99 100 150

VIVA (PRE & POST LAB) QUESTIONS

1. Sort the elements 77, 49, 25, 12, 9, 33, 56, 81 using bubblesort
2. The given array is arr = {1,2,4,3}. Bubble sort is used to sort the array elements. How many iterations will be done to sort the array?
3. The given array is arr = {1,2,4,3}. Bubble sort is used to sort the array elements. How many iterations will be done to sort the array with improvised version?
4. What does the following piece of code do?

```
if(a[j+1]<a[j])
{
temp = a[j];
a[j] = a[j+1];
a[j+1] = temp;
}
```

RESULT

Thus the C program for the concept of bubble sort was implemented successfully.

Ex.No:11.4

INSERTION SORT

AIM

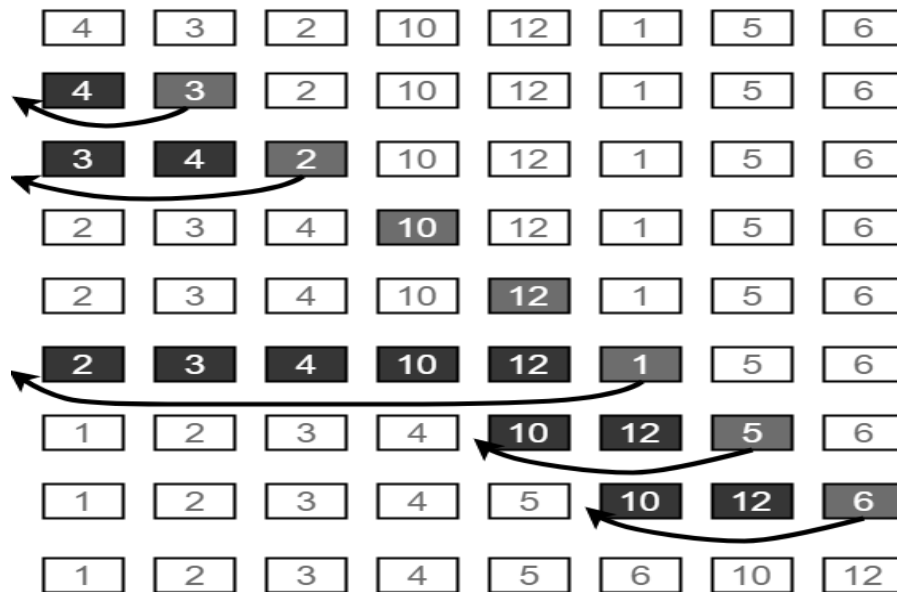
To write a C program to implement the concept of insertionsort.

PRE LAB DISCUSSION

Insertion sort is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be inserted in this sorted sub-list, has to find its appropriate place and then it has to be inserted there.

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Insertion Sort Execution Example



ALGORITHM

- Step 1: Start with an empty left hand [sorted array] and the cards face down on the table [unsorted array].
- Step 2: Then remove one card [key] at a time from the table [unsorted array], and insert it into the correct position in the left hand [sorted array].
- Step 3: To find the correct position for the card, we compare it with each of the cards already in the hand, from right to left.
- Step 4: Stop

PROGRAM

```
#include<stdio.h>
void inst_sort(int[]);
void main()
{
int num[5],count;
printf("\n enter the five elements to sort:\n");
for(count=0;count<5;count++)
scanf("%d",&num[count]);
inst_sort(num); /*function call for insertion sort*/
printf("\n\n elements after sorting:\n"); for(count=0;count<5;count++)
printf("%d\n",num[count]);
}
void inst_sort(int num[])
{ /* function definition for insertion sort*/
int i,j,k;
for(j=1;j<5;j++) { k=num[j];
for(i=j-1;i>=0&&num[i]>k;i--)
num[i+1]=num[i];
num[i+1]=k;
}}
```

OUTPUT

enter the five elements to sort:

5 4 3 2 1

elements after sorting:

1
2
3
4
5



VIVA (PRE & POST LAB) QUESTIONS

1. Which sorting algorithms in its typical implementation gives best performance when applied on an array which is sorted or almost sorted (maximum 1 or two elements are misplaced).
2. Which sorting algorithm will take least time when all elements of input array are identical?
3. Mention the uses of insertion sort.
4. Consider an array of elements $arr[5] = \{5,4,3,2,1\}$, what are the steps of insertions done while doing insertion sort in the array.

RESULT

Thus the C program to implement insertion sort was completed successfully.

Ex.No:12.1**HASHING WITH SEPARATE CHAINING****AIM**

To write a C program to implement the concept of hashing using separate chaining.

PRE LAB DISCUSSION

In hashing there is a hash function that maps keys to some values. But these hashing function may lead to collision that is two or more keys are mapped to same value. Chain hashing avoids collision. The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

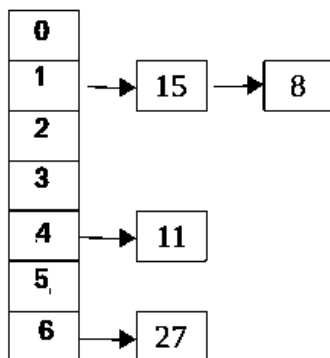
Let's create a hash function, such that our hash table has 'N' number of buckets. To insert a node into the hash table, we need to find the hash index for the given key. And it could be calculated using the hashfunction.

Example: $\text{hashIndex} = \text{key} \% \text{noOfBuckets}$

Insert: Move to the bucket corresponds to the above calculated hash index and insert the new node at the end of the list.

Let's say hash table with 7 buckets (0, 1, 2, 3, 4, 5, 6)

Keys arrive in the Order (15, 11, 27, 8)

**ALGORITHM**

Step 1:Start

Step 2:Create Table size

Step 3: Create hash function

Step 4: To insert a node into the hash table, we need to find the hash index for the given key. And it could be calculated using the hash function.

Step 5: Display hash entry.

Step 6: Stop

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 3
struct node
{
    int data;
    struct node *next;
};
struct node *head[TABLE_SIZE]={NULL},*c,*p;
void insert(int i,int val)
{
    struct node * newnode=(struct node *)malloc(sizeof(struct node));
    newnode->data=val;
    newnode->next = NULL;
    if(head[i] == NULL)
        head[i] = newnode;
    else
    {
        c=head[i];
        while(c->next != NULL)
            c=c->next;
        c->next=newnode;
    }
}
void display(int i)
{
    if(head[i] == NULL)
    {
        printf("No Hash Entry");
        return;
    }
    else
```



```

        {
            for(c=head[i];c!=NULL;c=c->next)
                printf("%d->",c->data);
        }
    }
main()
{
    int opt,val,i;
    while(1)
    {
        printf("\nPress 1. Insert\t2. Display \t3. Exit \n");
        scanf("%d",&opt);
        switch(opt)
        {
            case 1:
                printf("\nenter a value to insert into hash table\n");
                scanf("%d",&val);
                i=val%TABLE_SIZE;
                insert(i,val);
                break;
            case 2:
                for(i=0;i<TABLE_SIZE;i++)
                {
                    printf("\nentries at index %d\n",i);
                    display(i);
                }
                break;
            case 3: exit(0);
        }
    }
}

```

OUTPUT

```

C:\Users\sivaram\Desktop\hash.exe
Press 1. Insert  2. Display  3. Exit
1
enter a value to insert into hash table
21
Press 1. Insert  2. Display  3. Exit
1
enter a value to insert into hash table
18
Press 1. Insert  2. Display  3. Exit
1
enter a value to insert into hash table
31
Press 1. Insert  2. Display  3. Exit
2
entries at index 0
21->18->
entries at index 1
18->
entries at index 2
31->
No Hash Entry

```

VIVA (PRE & POST LAB) QUESTIONS

1. If several elements are competing for the same bucket in the hash table, what is it called?
2. How to insert a node in hashtable?
3. What is sizeof()function?
4. How to delete a node from hashtable.



RESULT

Thus the C program to implement hashing using separate chaining was completed successfully.

Ex.No:12.2

HASHING WITH OPEN ADDRESSING

AIM

To write a C program to implement the concept of hashing using linear probing in open addressing.

PRE LAB DISCUSSION

Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).

Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.

Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.

Delete(k): Delete operation is interesting. If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted".

Insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

Open Addressing is done following ways:

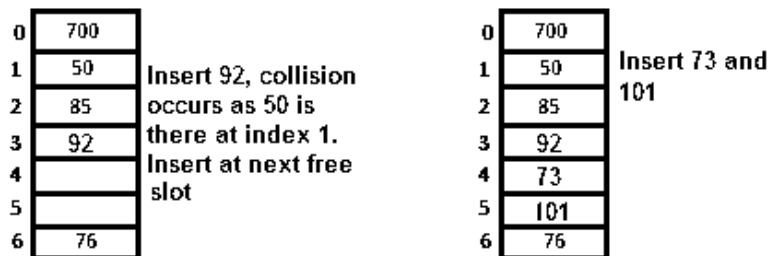
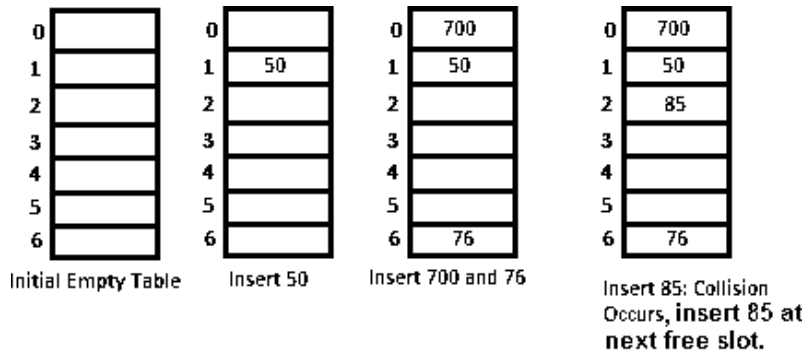
a) Linear Probing: In linear probing, we linearly probe for next slot. For example, typical gap between two probes is 1 as taken in below example also. let **hash(x)** be the slot index computed using hash function and **S** be the table size

If slot $\text{hash}(x) \% S$ is full, then we try $(\text{hash}(x) + 1) \% S$

If $(\text{hash}(x) + 1) \% S$ is also full, then we try $(\text{hash}(x) + 2) \% S$

If $(\text{hash}(x) + 2) \% S$ is also full, then we try $(\text{hash}(x) + 3) \% S$

Let us consider a simple hash function as "key mod 7" and sequence of keys as 50, 700, 76, 85, 92, 73, 101.



Clustering: The main problem with linear probing is clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search an element.

b) Quadratic Probing We look for i^2 th slot in i 'th iteration.
let $hash(x)$ be the slot index computed using hash function.

If slot $hash(x) \% S$ is full, then we try $(hash(x) + 1*1) \% S$

If $(hash(x) + 1*1) \% S$ is also full, then we try $(hash(x) + 2*2) \% S$

If $(hash(x) + 2*2) \% S$ is also full, then we try $(hash(x) + 3*3) \% S$

c) Double Hashing We use another hash function $hash2(x)$ and look for $i*hash2(x)$ slot in i 'th iteration.

let $hash(x)$ be the slot index computed using hash function.

If slot $hash(x) \% S$ is full, then we try $(hash(x) + 1*hash2(x)) \% S$

If $(hash(x) + 1*hash2(x)) \% S$ is also full, then we try $(hash(x) + 2*hash2(x)) \% S$

If $(hash(x) + 2*hash2(x)) \% S$ is also full, then we try $(hash(x) + 3*hash2(x)) \% S$

ALGORITHM

Step 1: Start

Step 2: Create hash table and hash function

Step 3: Assigning INT_MIN indicates that cell is empty

Step 4: INT_MIN and INT_MAX indicates that cell is empty. So if cell is empty loop will break and goto bottom of the loop to insert element. If table is full we should break, if not check this, loop will go to infiniteloop.

Step 5: If the hash element is empty, the deletion has immediately failed.

Otherwise, check for a match between the target and data key

- If there is a match, delete the hash element (and set theflag)
- If there is no match, probe the table until either:
 - An match is found between the target key and the data's key; the data can be deleted, and the deleted flagset.
 - A completely empty hash element isfound

Step 6: If the hash element is empty, the search has immediately failed.

Otherwise, check for a match between the search and data key

- If there is a match, return the data.
- If there is no match, probe the table until either:
 - An match is found between the search and datakey
 - A completely empty hash element isfound.

Step 7: Stop

PROGRAM

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

void insert(int ary[],int hFn, int size){
    int element,pos,n=0;

    printf("Enter key element to insert\n");
    scanf("%d",&element);

    pos = element%hFn;
```

```

while(ary[pos]!= INT_MIN)
{
// INT_MIN and INT_MAX indicates that cell is empty. So if cell is empty loop will
break and goto bottom of the loop to insert element
if(ary[pos]== INT_MAX)
break;
pos = (pos+1)%hFn;
n++;
if(n==size)
break; // If table is full we should break, if not check this, loop will go to infinite
loop.
}

```

```

if(n==size)
printf("Hash table was full of elements\nNo Place to insert this element\n\n");
else
ary[pos]=element; //Inserting element
}

```

```

void delete(int ary[],int hFn,int size)
{
/* very careful observation required while deleting. To understand code of this delete
function see the note at end of the program*/
int element,n=0,pos;

printf("Enter element to delete\n");
scanf("%d",&element);

pos = element%hFn;

while(n++ != size){
if(ary[pos]==INT_MIN){
printf("Element not found in hash table\n");
break;
}
else if(ary[pos]==element)
{
ary[pos]=INT_MAX;
printf("Element deleted\n\n");
break;
}
else
{
pos = (pos+1) % hFn;
}
}
}

```



```

    if(--n==size)
        printf("Element not found in hash table\n");
}

void search(int ary[],int hFn,int size){
    int element,pos,n=0;

    printf("Enter element you want to search\n");
    scanf("%d",&element);

    pos = element%hFn;

    while(n++ != size){
        if(ary[pos]==element){
            printf("Element found at index %d\n",pos);
            break;
        }
        else
            if(ary[pos]==INT_MAX ||ary[pos]!=INT_MIN)
                pos = (pos+1) %hFn;
    }
    if(--n==size) printf("Element not found in hash table\n");
}

void display(int ary[],int size){
    int i;

    printf("Index\tValue\n");

    for(i=0;i<size;i++)
        printf("%d\t%d\n",i,ary[i]);
}

int main(){
    int size,hFn,i,choice;

    printf("Enter size of hash table\n");
    scanf("%d",&size);

    int ary[size];

    printf("Enter hash function [if mod 10 enter 10]\n");
    scanf("%d",&hFn);

    for(i=0;i<size;i++)
        ary[i]=INT_MIN; //Assigning INT_MIN indicates that cell is empty
}

```



```

do{
    printf("Enter your choice\n");
    printf(" 1-> Insert\n 2-> Delete\n 3-> Display\n 4-> Searching\n 0-> Exit\n");
    scanf("%d",&choice);

    switch(choice){
        case 1:
            insert(ary,hFn,size);
            break;
        case 2:
            delete(ary,hFn,size);
            break;
        case 3:
            display(ary,size);
            break;
        case 4:
            search(ary,hFn,size);
            break;
        default:
            printf("Enter correct choice\n");
            break;
    }
}while(choice);

return 0;
}

```



OUTPUT

```

Enter size of hash table
10
Enter hash function [if mod 10 enter 10]
10
Enter your choice
1-> Insert
2-> Delete
3->Display
4->Searching
0->Exit
1
Enter key element to insert
12
Enter your choice
1-> Insert

```

2-> Delete
3->Display
4->Searching
0->Exit

1
Enter key element to insert
22

Enter your choice
1-> Insert
2-> Delete
3->Display
4->Searching
0->Exit

1
Enter key element to insert
32

Enter your choice
1-> Insert
2-> Delete
3->Display
4->Searching
0->Exit

3
Index Value
0 -2147483648
1 -2147483648
2 12
3 22
4 32
5 -2147483648
6 -2147483648
7 -2147483648
8 -2147483648
9 -2147483648



Enter your choice
1-> Insert
2-> Delete
3->Display
4->Searching
0->Exit

2
Enter element to delete
12
Element deleted

Enter your choice

1-> Insert

2-> Delete

3->Display

4->Searching

0->Exit

4

Enter element you want to search

32

Element found at index 4

Enter your choice

1->Insert

2->Delete

3->Display

4->Searching

0->Exit

0

VIVA (PRE & POST LAB) QUESTIONS

1. What is hashfunction?
2. How to insert an element in hash table using linearprobing?
3. How to delete an element from hash table using linearprobing?
4. What is index?
5. How to search an element in hash table using linearprobing?

RESULT

Thus the C program to implement hashing using linear probing in open addressing was completed successfully.

TOPIC BEYOND SYLLABUS



Ex.No:13

IMPLEMENTATION DOUBLY LINKEDLIST

AIM

To write a C program to implement doubly linked list with Insert, Delete and Display operations.

PRE LAB DISCUSSION

Doubly linked list is a sequence of elements in which every node has link to its previous node and next node.

- Traversing can be done in both directions and displays the contents in the whole list.

Node

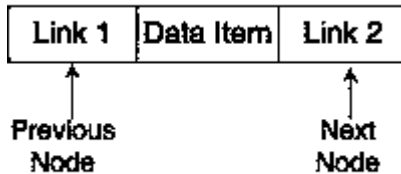


Fig. Doubly Linked List

Link1 field stores the address of the previous node and Link2 field stores the address of the next node. The Data Item field stores the actual value of that node. If we insert a data into the linked list, it will be look like asfollows:

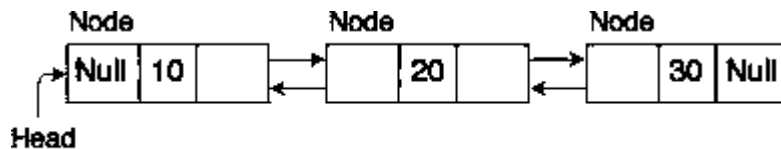


Fig. Doubly Linked List

In doubly linked list contains three fields. In this, link of two nodes allow traversal of the list in either direction. There is no need to traverse the list to find the previous node. We can traverse from head to tail as well as tail to head.

ALGORITHM

Step 1: Start

Step 2: Creation: Get the number of elements to create the list. Then create the node having the Structure: BLINK, DATA , FLINK and store the elements in Data field. Link them together to form a doubly linked list.

Step 3: Insertion: Get the number to be Inserted, create a new node to store the value. Search the list and insert the node in its right position.

Step 4: Deletion: Get the number to be deleted. Search the list from the beginning and try to locate node p with DATA. If found then delete the node.

Step 5: FLINK P's previous node to P's Next node. BLINK P's Next node to P's Previous node else display "Data notFound".

Step 6: Display: Display all the nodes in the list.

Step 7: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define NULL 0
typedef struct list
{
    int no;
    struct list *next;
    struct list *pre;
```



```

}LIST;
LIST *p,*t,*h;
void create( void );
void insert( void );
void delet( void);
void display ( void );
int j,pos,k=1,count;
void main()
{
  int n,i=1,opt;
  clrscr();
  p = NULL;
  printf( "Enter the no of nodes :\n " );
  scanf( "%d",&n );
  count = n;
  while( i <= n)
  { create();
    i++;
  }
  printf("\nEnter your option:\n");
  printf("1.Insert \t 2.Delete \t 3.Display \t 4.Exit\n");
  do
  {
    scanf("%d",&opt);
    switch( opt ){
      case 1:
        insert();
        count++;
        break;

      case 2:

```




```

delet();
count--;
if ( count == 0 )
{
printf("\n List is empty\n");
}
break;

case 3:
printf("List elements are:\n");
display();
break;
}
printf("\nEnter your option \n");
}while( opt != 4 );
getch();
}

```



```

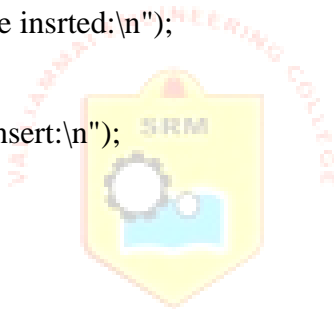
void create ( )
{
if( p == NULL )
{
p = ( LIST * ) malloc ( sizeof ( LIST ) );
printf( "Enter the element:\n" );
scanf( "%d",&p->no );
p->next = NULL;
p->pre = NULL;
h = p;
}
else
{

```

```

t= ( LIST * ) malloc (sizeof( LIST ));
printf( "\nEnter the element" );
scanf( "%d",&t->no );
t->next = NULL;
p->next = t;
t->pre = p;
p = t;
}
}
void insert()
{
t=h;
p = ( LIST * ) malloc ( sizeof(LIST) );
printf("Enter the element to be insrted:\n");
scanf("%d",&p->no);
printf("Enter the position to insert:\n");
scanf( "%d",&pos );
if( pos == 1 )
{
h = p;
h->next = t;
t->pre = h;
h->pre = NULL;
}
else
{
for(j=1;j<(pos-1);j++)
t = t->next;
p->next = t->next;
t->next = p;
p->pre = t;

```



```

}
}
void delete()
{
printf("Enter the position to delete:\n");
scanf( "%d",&pos );
if( pos == 1)
{
h = h->next;
h->pre = NULL;
}
else
{
t=h;
for(j=1;j<(pos-1);j++)
t = t->next;
t->next = t->next->next;
t->next->pre = t;
free( t->next );
}
}
void display()
{
t= h;
while( t->next != NULL )
{
printf("%d\n",t->no);
t = t->next;
}
printf( "%d",t->no );
}

```



OUTPUT

Enter the no of nodes: 3
Enter the element3
Enter your option:
1.Insert 2.Delete 3.Display4.Exit
3
List elements are:
1 2 3
Enter your option 1
Enter the element to be inserted:22
Enter the position to insert:1
Enter your option 3
List elements are:
22 1 2 3
Enter your option 1
Enter the element to be inserted:
11
Enter the position to insert:5
Enter your option 3
List elements are:
22 1 2 3 11
Enter your option
2
Enter the position to delete:
1
Enter your option
3
List elements are:
1 2 3 11
Enter your option 4



VIVA (PRE & POST LAB) QUESTIONS

1. How will you traverse double linkedlist?
2. Give the structure of node in a doubly linkedlist.
3. Specify the steps to insert an element in Doubly LinkedList.
4. How to delete an element from a doubly linkedlist.



RESULT

Thus the C program to implement Doubly Linked List was completed successfully.