Bharath
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Declared as Deemed - to - be - University under section 3 of UGC Act 1956)

BHARATH INSTITUTE OF SCIENCE & TECHNOLOGY

173, Agaram Road, Selaiyur, Chennai-600073. Tamil Nadu, India.

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

BACHELOR OF TECHNOLOGY

COURSE CODE: U20CSCJ02 PROGRAMMING FOR DATA STRUCTURES AND ALGORITHMS LABORATORY RECORD

Name of the Student:

Batch: 2021-2025 Year: II Term: 3 Section:

Register No: U21

JAN 2023

BHARATH INSTITUTE OF SCIENCE AND TECHNOLOGY

173, Agaram Road, Selaiyur, Chennai-600073. Tamil Nadu, India.

Name____________________________________

Programme_________**Branch**_________________

Year _______________

Register No: U21

Certified that this is the bonafide record of work done by the above student in the **PROGRAMMING FOR DATA STRUCTURES AND ALGORITHMS LABORATORY** during the Term 3 in the Academic Year 2022 -2023

Faculty in-charge   Head of Department

Submitted for the practical Examination held on ………………………

Internal Examiner   External Examiner

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LIST OF PROGRAMS TO IMPLEMENT

S.No	Date	Programs	Page number	Staff sign
1a		Simple Structures		
1b		Nested Structure In C: Struct Inside Another Struct		
2a		Fibonacci Series Up To N Terms		
2b		Factorial Of A Given Number Using Recursive Function		
3a		Implementation Of Singly Linked List		
3b		Implementation Of Doubly Linked List		
4a		Implementation Of Stack Using Array		
4b		Implementation Of Stack using Linked List		
5a		Array Implementation Of Queue		
5b		Implement Queue Using Linked List		
6		Implementation Of Tree		
7a		Sorting Algorithm- Insertion Sort		
7b		Sorting Algorithm- Selection Sort		
7c		Sorting Algorithm- Merge Sort		
8		Implementing Hash Table In C		
9a		Depth First Graph Traversal		
9b		Breadth First Graph Traversal		
10		Shortest Path Using Dijkstra's Algorithm		

Pr. No. 1A **Date:**

SIMPLE STRUCTURES

PROBLEM DEFINITION:
Create a struct (or structure) with a collection of variables (can be of different types) under a single name.

ALGORITHM:
create a structure StudentData
The three data members are stu_name, stu_id and stu_age.

1. Insert the values for data members student name, id and age into the structure
2. And accessing structure data members to display these values as an output.

PROGRAM:
```
#include<stdio.h>
/* Created a structure here. The name of the structure is
 * StudentData.
 */
structStudentData{
char *stu_name;
int stu_id;
int stu_age;
};
int main()
{
/* student is the variable of structure StudentData*/
structStudentData student;

/*Assigning the values of each struct member here*/
    student.stu_name = "Steve";
    student.stu_id = 1234;
    student.stu_age = 30;

/* Displaying the values of struct members */
    printf("Student Name is: %s", student.stu_name);
    printf("\nStudent Id is: %d", student.stu_id);
    printf("\nStudent Age is: %d", student.stu_age);
return0;
}
```

Output:

```
StudentNameis: Steve
StudentIdis: 1234
StudentAgeis: 30
```

RESULT: Thus the structure created with a collection of variables under a single name successfully.

Pr. No. 1B **Date:**

NESTED STRUCTURE IN C: STRUCT INSIDE ANOTHER STRUCT

PROBLEM DEFINITION:
Create a nested structure in c, struct inside another struct for a Student

ALGORITHM:

```
struct stu_address
{
int street;
char *state;
char *city;
char *country;
};
```

Structure 2: stu_data

```
struct stu_data


{
int stu_id;
int stu_age;
char *stu_name;
struct stu_address stuAddress;
};
```

PROGRAM:

```
#include<stdio.h>
struct stu_address
{
int street;
char *state;
char *city;
char *country;
};
struct stu_data
{
int stu_id;
int stu_age;
char *stu_name;
struct stu_address stuAddress;
};
int main(){
struct stu_data mydata;
  mydata.stu_id = 1001;
  mydata.stu_age = 30;
  mydata.stu_name = "Chaitanya";
  mydata.stuAddress.state = "UP";
  mydata.stuAddress.street = 101;
  mydata.stuAddress.city = "Delhi";
  mydata.stuAddress.country = "India";
  printf("Printing student Data: ");
  printf("\nStudent id: %d",mydata.stu_id);
  printf("\nStudent age: %d",mydata.stu_age);
  printf("\nStudent name: %s",mydata.stu_name);
  printf("\nStudent street: %d",mydata.stuAddress.street);
  printf("\nStudent state: %s",mydata.stuAddress.state);
  printf("\nStudent city: %s",mydata.stuAddress.city);
  printf("\nStudent country: %s",mydata.stuAddress.country);

return0;
}
```

OUTPUT:

```
Printing student Data:
Student id: 1001
Student age: 30
Student name: Chaitanya
Student street: 101
Student state: UP
Student city: Delhi
Student country: India
```

RESULT: Created a nested structure in c, struct inside another struct for a Student Successfully.

Pr.No:2 A **FIBONACCI SERIES UP TO N TERMS** **Date:**

PROBLEM DEFINITION:

Write a Program in C to Print the Fibonacci series up to N Terms

ALGORITHM:

```
Step1:  Start

Step2: Declare variables i, t1, t2, n

Step3: Initialize the variables, t1=0, t2=1

Step4: Enter the number of terms of Fibonacci series to be printed

Step5: Print first two terms of series
Step6: Use loop for the following steps
            Next term=t1+t2
        t1=t2
        t2=next term
        increase value of i each time by 1
        print the value of next term

Step7: End
```

PROGRAM:

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

  int i, n;

  // initialize first and second terms
  int t1 = 0, t2 = 1;

  // initialize the next term (3rd term)
  int nextTerm = t1 + t2;

  // get no. of terms from user
  printf("Enter the number of terms: ");
  scanf("%d", &n);

  // print the first two terms t1 and t2


  printf("Fibonacci Series: %d, %d, ", t1, t2);

  // print 3rd to nth terms
  for (i = 3; i <= n; ++i) {
    printf("%d, ", nextTerm);
    t1 = t2;
    t2 = nextTerm;
    nextTerm = t1 + t2;
  }

  return 0;
}
```

OUTPUT:

```
Enter the number of terms: 10
Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
```

RESULT:

The C program to print the Fibonacci series up to N Terms has been verified and executed successfully.

Pr. No:2 B **Date:**

FACTORIAL OF A GIVEN NUMBER USING RECURSIVE FUNCTION

PROBLEM DEFINITION:

Write a Program in C to find the Factorial of a given number using Recursive function.

ALGORITHM:

```
Step 1: Start
Step 2: Read number n
Step 3: Call factorial(n)
Step 4: Print factorial f
Step 5: Stop
            factorial(n)
Step 1: If n==1 then return 1
Step 2: Else
             f=n*factorial(n-1)
Step 3: Return f
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main ()
{
clrscr ()
 int factorial(int);
 int n,f;
 printf("Enter the number: ");
 scanf("%d",&n);
 f=factorial(n);
 printf("Factorial of the number is %d",f);
 getch();
}
int factorial (int n)
{
  int f;
  if(n==1)
     return 1;
  else
     f=n*factorial(n-1);
     return f;
}
```

OUTPUT:

```
Enter the number: 5

Factorial of the number is 120.
```

RESULT: The C Program to find the Factorial of a given number using Recursive function has been verified and executed successfully.

Pr.No:3a **IMPLEMENTATION OF SINGLY LINKED LIST** **Date:**

PROBLEM DEFINITION:
To implement the singly linked list operations to insert, Delete, Count and Display.
ALGORITHM:

Inserting

- Step 1 - Create a **newNode** with given value.
- Step 2 - Check whether list is **Empty** (**head** == **NULL**)
- Step 3 - If it is **Empty** then, set **newNode→next** = **NULL** and **head** = **newNode**.
- Step 4 - If it is **Not Empty** then, set **newNode→next** = **head** and **head** = **newNode**.

Deleting

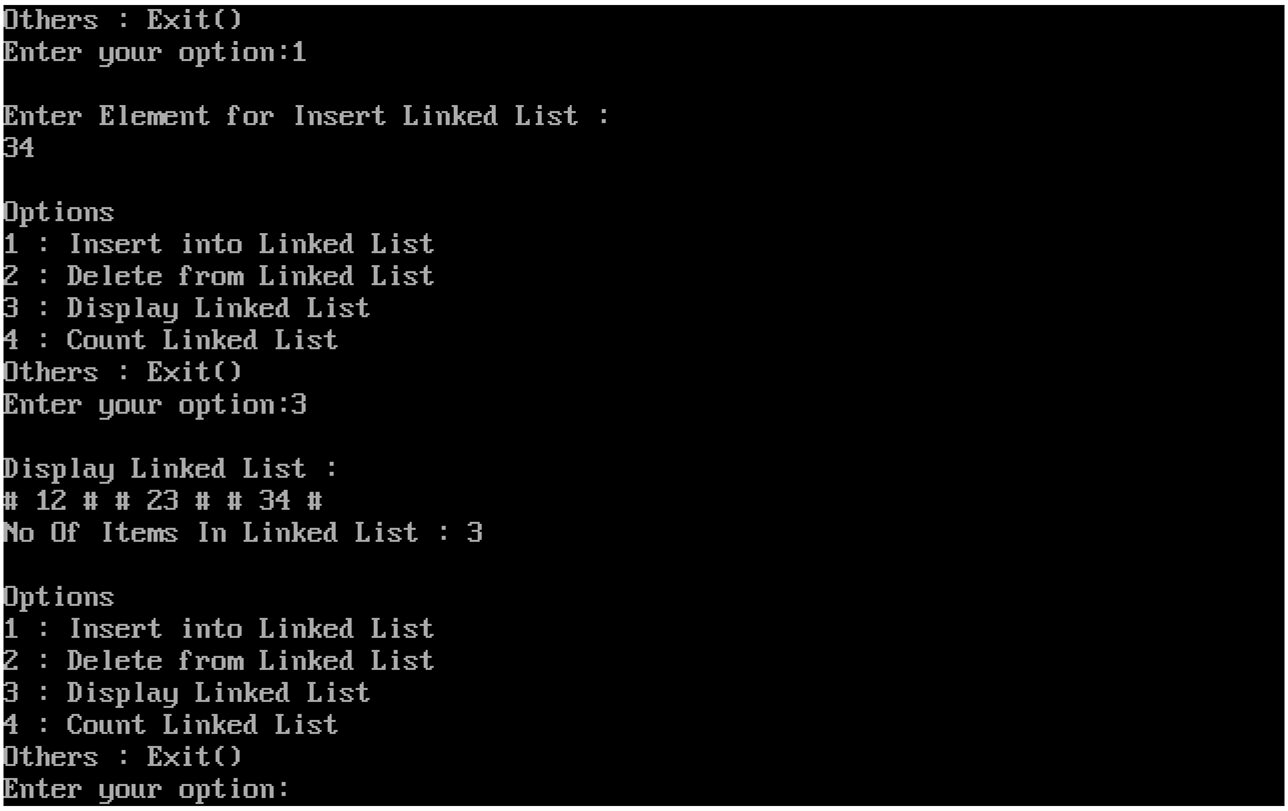
- Step 1 - Check whether list is **Empty** (**head** == **NULL**)
- Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
- Step 3 - If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **head**.
- Step 4 - Check whether list is having only one node (**temp → next** == **NULL**)
- Step 5 - If it is **TRUE** then set **head** = **NULL** and delete **temp** (Setting **Empty** list conditions)
- Step 6 - If it is **FALSE** then set **head** = **temp → next**, and delete **temp**.

Display

- Step 1 - Check whether list is **Empty** (**head** == **NULL**)
- Step 2 - If it is **Empty** then, display **'List is Empty!!!'** and terminate the function.
- Step 3 - If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **head**.
- Step 4 - Keep displaying **temp → data** with an arrow (**--->**) until **temp** reaches to the last node
- Step 5 - Finally display **temp → data** with arrow pointing to **NULL** (**temp → data ---> NULL**).

PROGRAM
//Singly Linked List
```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>

struct node {
  int value;
  struct node *next;
};

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

typedef struct node DATA_NODE;

DATA_NODE *head_node, *first_node, *temp_node = 0, *prev_node, next_node;
int data;

int main() {
  int option = 0;
  printf("Singly Linked List Example - All Operations\n");

  while (option < 5) {

    printf("\nOptions\n");
                              printf("1 : Insert into Linked List \n");
                              printf("2 : Delete from Linked List \n");
                              printf("3 : Display Linked List\n");
                              printf("4 : Count Linked List\n");
    printf("Others : Exit()\n");
                              printf("Enter your option:");
    scanf("%d", &option);
                              switch (option) {
                              case 1:
                              insert();
                              break;
                              case 2:
                              delete();
                              break;
                              case 3:
                                display();
                              break;
                              case 4:
                              count();
                              break;
                              default:
                              break;
                              }
  }

  return 0;
}

void insert() {
  printf("\nEnter Element for Insert Linked List : \n");
  scanf("%d", &data);

  temp_node = (DATA_NODE *) malloc(sizeof (DATA_NODE));

  temp_node->value = data;

  if (first_node == 0) {
                              first_node = temp_node;
  } else {
                              head_node->next = temp_node;
  }
  temp_node->next = 0;
  head_node = temp_node;
  fflush(stdin);
}

void delete() {
  int countvalue, pos, i = 0;
  countvalue = count();
  temp_node = first_node;
  printf("\nDisplay Linked List : \n");

  printf("\nEnter Position for Delete Element : \n");
  scanf("%d", &pos);

  if (pos > 0 && pos <= countvalue) {
                              if (pos == 1) {
                              temp_node = temp_node -> next;
                              first_node = temp_node;
      printf("\nDeleted Successfully \n\n");
                              } else {
                              while (temp_node != 0) {
                              if (i == (pos - 1)) {
          prev_node->next = temp_node->next;
                                   if(i == (countvalue - 1))
                              {
                                head_node = prev_node;
                                }
          printf("\nDeleted Successfully \n\n");
                              break;
                              } else {
                              i++;
                              prev_node = temp_node;
                              temp_node = temp_node -> next;
                               }
                              }
                              }
  } else
    printf("\nInvalid Position \n\n");
}

void display() {
  int count = 0;
  temp_node = first_node;
  printf("\nDisplay Linked List : \n");
  while (temp_node != 0) {
                              printf("# %d # ", temp_node->value);
                              count++;
                              temp_node = temp_node -> next;
  }
  printf("\nNo Of Items In Linked List : %d\n", count);
}

int count() {
  int count = 0;
  temp_node = first_node;
  while (temp_node != 0) {
                              count++;
                              temp_node = temp_node -> next;
  }
  printf("\nNo Of Items In Linked List : %d\n", count);
  return count;
}
```

Output:



```
Others : Exit()
Enter your option:1

Enter Element for Insert Linked List :
34

Options
1 : Insert into Linked List
2 : Delete from Linked List
3 : Display Linked List
4 : Count Linked List
Others : Exit()
Enter your option:3

Display Linked List :
# 12 # # 23 # # 34 #
No Of Items In Linked List : 3

Options
1 : Insert into Linked List
2 : Delete from Linked List
3 : Display Linked List
4 : Count Linked List
Others : Exit()
Enter your option:
```

RESULT: Thus the C program is implemented and verified the output for Singly Linked list to insert Delete and display.

Pr.No:3B **TO IMPLEMENT DOUBLY LINKED LIST.** **Date:**

PROBLEM DEFINITION:

Write a C program to implement Doubly linked List

ALGORITHM:

insert

```
Step 1: IF ptr = NULL
Write OVERFLOW
 Go to Step 9
 [END OF IF]
Step 2: SET NEW_NODE = ptr
Step 3: SET ptr = ptr -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> PREV = NULL
Step 6: SET NEW_NODE -> NEXT = START
Step 7: SET head -> PREV = NEW_NODE
Step 8: SET head = NEW_NODE
Step 9: EXIT
```

Algorithm: Delete

```
STEP 1: IF HEAD = NULL
WRITE UNDERFLOW
GOTO STEP 6
STEP 2: SET PTR = HEAD
STEP 3: SET HEAD = HEAD → NEXT
STEP 4: SET HEAD → PREV = NULL
STEP 5: FREE PTR
STEP 6: EXIT
```

Program

//Doubly Linked List

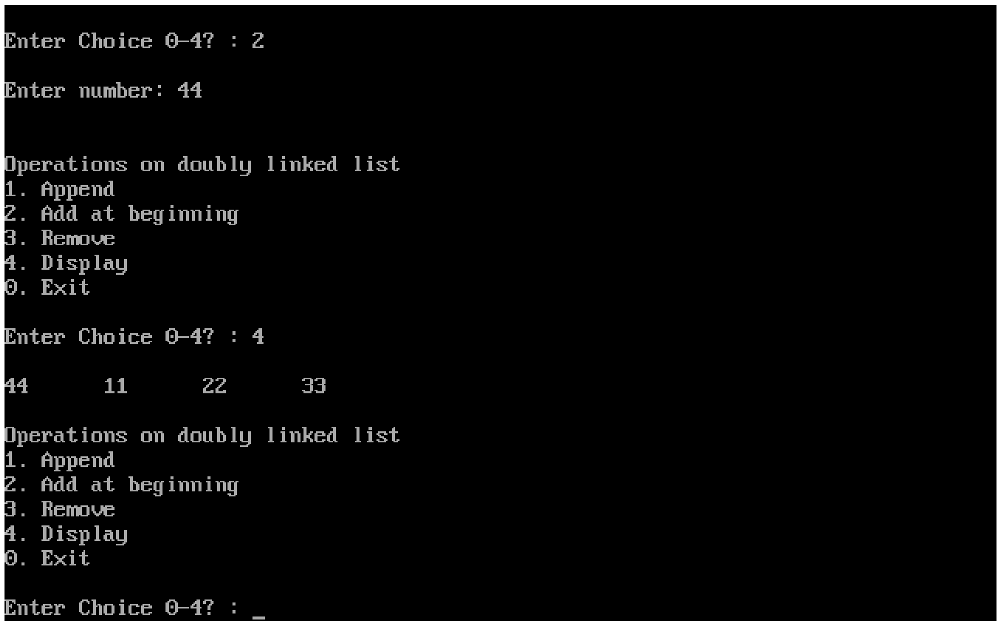
```
#include <stdio.h>
#include <stdlib.h>

/* structure representing a node of the doubly linked list */
struct dnode
{
                struct dnode *prev;
                int data;
                struct dnode *next;
};

struct dnode *start = NULL;

void append(int);
void addatbeg(int);
void remov(int);
void display();

int main()
{
                int n, ch;
```

```
                        do
                        {
                        printf("\n\nOperations on doubly linked list");
                        printf("\n1. Append \n2. Add at beginning \n3. Remove\n4. Display\n0. Exit\n");
                        printf("\nEnter Choice 0-4? : ");
                        scanf("%d", &ch);
                        switch (ch)
                        {
                        case 1:
            printf("\nEnter number: ");
            scanf("%d", &n);
                        append(n);
                        break;
                        case 2:
            printf("\nEnter number: ");
            scanf("%d", &n);
                        addatbeg(n);
                        break;
                        case 3:
                             printf("\nEnter number to delete: ");
            scanf("%d", &n);
                        remov(n);
                        break;
                            case 4:
                        display();
                        break;
                        }
                        }while (ch != 0);
}

/* adds a new node at the end of the doubly linked list */
void append(int num)
{
                        struct dnode *nptr,  *temp = start;

                        /*create a new node */
                        nptr = malloc(sizeof(struct dnode));
                        nptr->data = num;
                        nptr->next = NULL;
                        nptr->prev = NULL;

                        /* if the linked list is empty */
                        if (start == NULL)
                        {
                        start = nptr;
                        }
                        else
                        {
                        /* traverse the linked list till the last node is reached */
                        while (temp->next != NULL)
                        temp = temp->next;

                        nptr->prev = temp;
                        temp->next = nptr;
                        }
}

/* adds a new node at the begining of the linked list */
```

```
void addatbeg(int num)
{
                struct dnode *nptr;

                /* create a new node */
                nptr = malloc(sizeof(struct dnode));

                /* assign data and pointer to the new node */
                nptr->prev = NULL;
                nptr->data = num;
                nptr->next = start;

                if (start != NULL)
                start->prev = nptr;
                start = nptr;
}


/* deletes the specified node from the doubly linked list */
void remov(int num)
{
                struct dnode *temp = start;

                /* traverse the entire linked list */
                while (temp != NULL)
                {
                /* if node to be deleted is found */
                if (temp->data == num)
                {
                /* if node to be deleted is the first node */
                if (temp == start)
                {
                start = start->next;
                start->prev = NULL;
                }
                else
                {
                /* if node to be deleted is the last node */
                if (temp->next == NULL)
          temp->prev->next = NULL;
                else
                /* if node to be deleted is any intermediate node */
                {
          temp->prev->next = temp->next;
          temp->next->prev = temp->prev;
                }
                free(temp);
                }
                return ; /* return back after deletion */
                }
                temp = temp->next; /* go to next node */
                }
                printf("\n%d not found.", num);
}

/* displays the contents of the linked list */
void display()
{
```

```
                    struct dnode *temp = start;
                    printf("\n");

                    /* traverse the entire linked list */
                    while (temp != NULL)
                    {
                    printf("%d\t", temp->data);
                    temp = temp->next;
                    }
}
```

OUTPUT:



```
Enter Choice 0-4? : 2

Enter number: 44


Operations on doubly linked list
1. Append
2. Add at beginning
3. Remove
4. Display
0. Exit

Enter Choice 0-4? : 4

44      11      22      33

Operations on doubly linked list
1. Append
2. Add at beginning
3. Remove
4. Display
0. Exit

Enter Choice 0-4? : _
```

RESULT: Thus the C program is implemented and verified the output for Doubly Linked list to Append, Add at beginning, Delete and Display.

Pr.No:4A **IMPLEMENTATION OF STACK USING ARRAY** **Date:**

PROBLEM DEFINITION:
To implement a Stack using array.

ALGORITHM:
push(value) - Inserting value into the stack
In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at top position. Push function takes one integer value as parameter and inserts that value into the stack. We can use the following steps to push an element on to the stack...

```
Step 1: Check whether stack is FULL. (top == SIZE-1)
Step 2: If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function.
Step 3: If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (stack[top] = value).
```

pop() - Delete a value from the Stack
In a stack, pop() is a function used to delete an element from the stack. In a stack, the element is always deleted from top position. Pop function does not take any value as parameter. We can use the following steps to pop an element from the stack...

```
Step 1: Check whether stack is EMPTY. (top == -1)
Step 2: If it is EMPTY, then display "Stack is EMPTY!!! Deletion is not possible!!!" and terminate the function.
Step 3: If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).
```

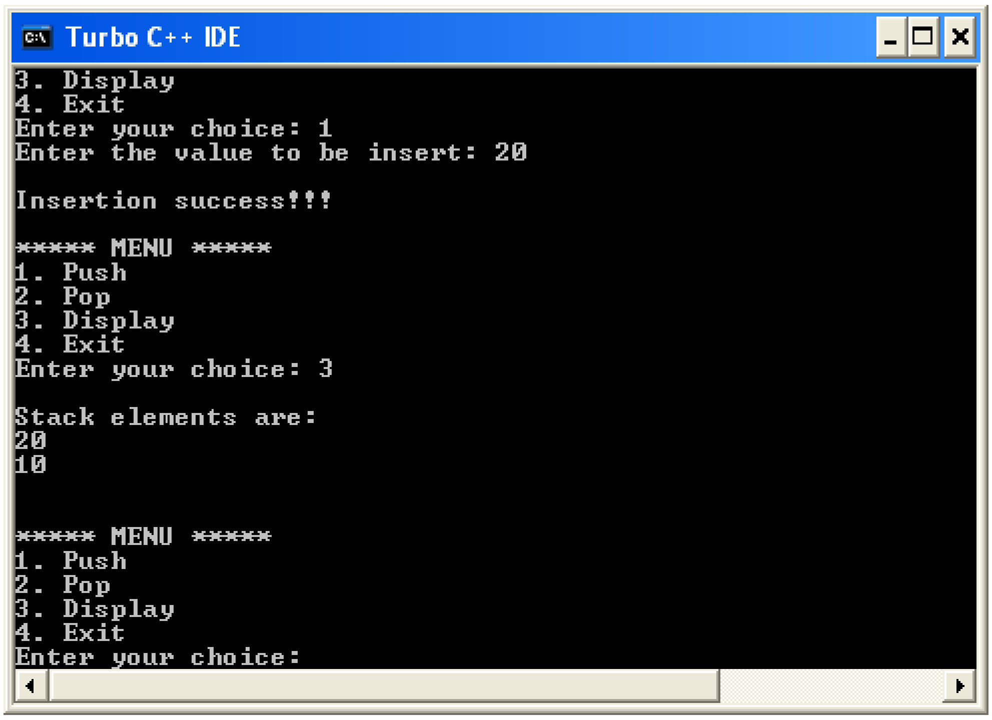
display() - Displays the elements of a Stack
We can use the following steps to display the elements of a stack...

```
Step 1: Check whether stack is EMPTY. (top == -1)
Step 2: If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.
Step 3: If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).
Step 3: Repeat above step until i value becomes '0'.
```

PROGRAM:
```c
#include<stdio.h>
#include<conio.h>

#define SIZE 10

void push(int);
void pop();
void display();

int stack[SIZE], top = -1;

void main()
{
  int value, choice;
  clrscr();
  while(1){
    printf("\n\n***** MENU *****\n");
    printf("1. Push\n2. Pop\n3. Display\n4. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice){
                    case 1: printf("Enter the value to be insert: ");
```

```
                        scanf("%d",&value);
                        push(value);
                        break;
                    case 2: pop();
                        break;
                    case 3: display();
                        break;
                    case 4: exit(0);
                    default: printf("\nWrong selection!!! Try again!!!");
    }
  }
}
void push(int value){
  if(top == SIZE-1)
    printf("\nStack is Full!!! Insertion is not possible!!!");
  else{
    top++;
    stack[top] = value;
    printf("\nInsertion success!!!");
  }
}
void pop(){
  if(top == -1)
    printf("\nStack is Empty!!! Deletion is not possible!!!");
  else{
    printf("\nDeleted : %d", stack[top]);
    top--;
  }
}
void display(){
  if(top == -1)
    printf("\nStack is Empty!!!");
  else{
    int i;
    printf("\nStack elements are:\n");
    for(i=top; i>=0; i--)
                    printf("%d\n",stack[i]);
  }
}
```

Output:



```
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20

Insertion success!!!

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3

Stack elements are:
20
10


***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:
```

RESULT: Thus the C program is implemented and verified the output for Stack using Array to insert Delete and display.

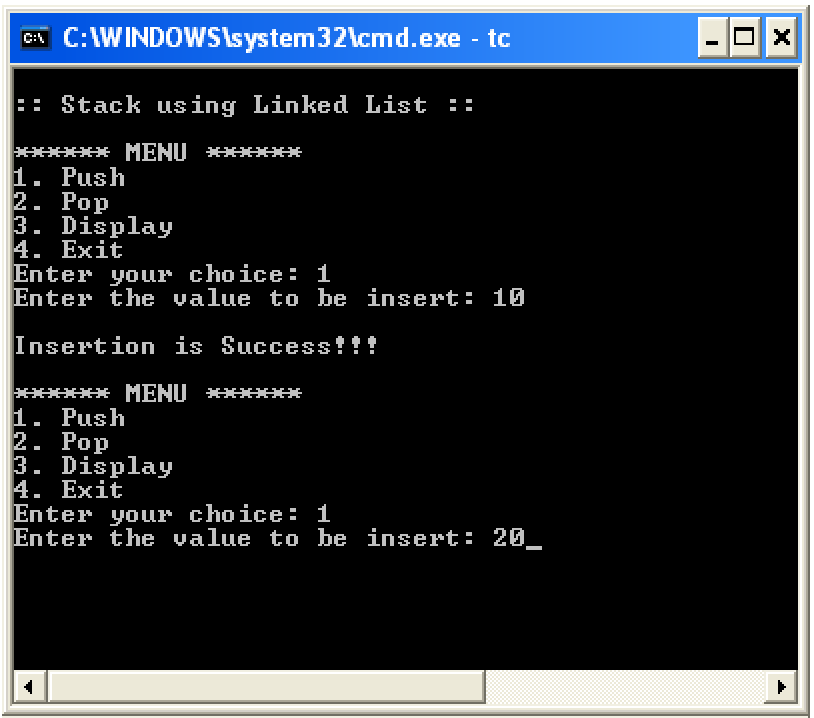
Pr.No:4b **To implement Stack using Linked List.** **Date:**

PROBLEM DEFINITION:
Write a C program to implement Stack using linked List

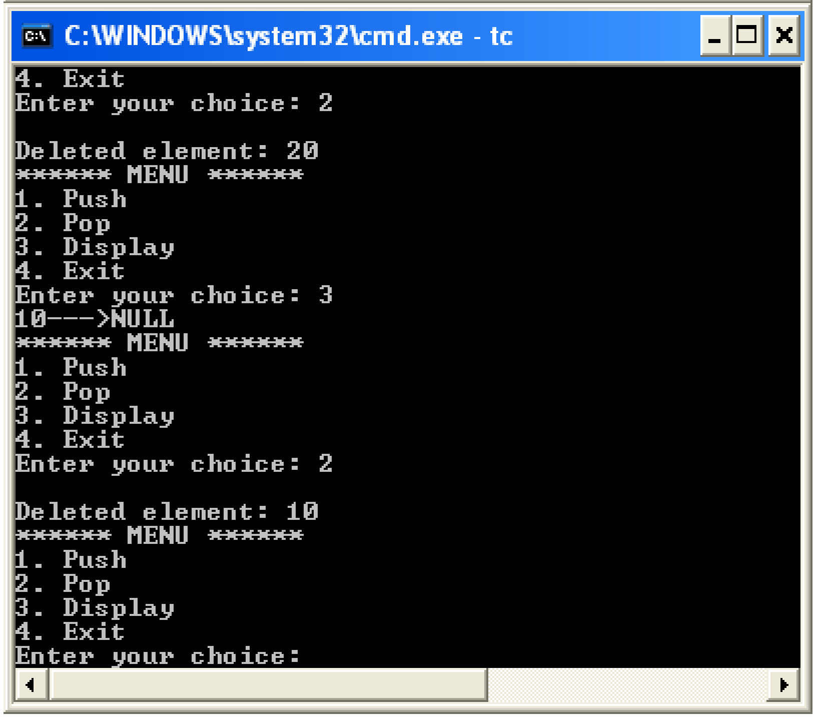
ALGORITHM:
To implement stack using linked list, we need to set the following things before implementing actual operations.
Step 1: Include all the header files which are used in the program. And declare all the user defined functions.
Step 2: Define a 'Node' structure with two members data and next.
Step 3: Define a Node pointer 'top' and set it to NULL.
Step 4: Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method.
push(value) - Inserting an element into the Stack
We can use the following steps to insert a new node into the stack...
Step 1: Create a newNode with given value.
Step 2: Check whether stack is Empty (top == NULL)
Step 3: If it is Empty, then set newNode → next = NULL.
Step 4: If it is Not Empty, then set newNode → next = top.
Step 5: Finally, set top = newNode.
pop() - Deleting an Element from a Stack
We can use the following steps to delete a node from the stack...
Step 1: Check whether stack is Empty (top == NULL).
Step 2: If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function
Step 3: If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
Step 4: Then set 'top = top → next'.
Step 7: Finally, delete 'temp' (free(temp)).
display() - Displaying stack of elements
We can use the following steps to display the elements (nodes) of a stack...
Step 1: Check whether stack is Empty (top == NULL).
Step 2: If it is Empty, then display 'Stack is Empty!!!' and terminate the function.
Step 3: If it is Not Empty, then define a Node pointer 'temp' and initialize with top.
Step 4: Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack (temp → next != NULL).
Step 4: Finally! Display 'temp → data ---> NULL'.

PROGRAM :
//Stack Using Linked List
```
#include<stdio.h>
#include<conio.h>

struct Node
{
   int data;
   struct Node *next;
}*top = NULL;

void push(int);
void pop();
void display();
void main()
{
   int choice, value;
   clrscr();
   printf("\n:: Stack using Linked List ::\n");
   while(1){
    printf("\n****** MENU ******\n");
    printf("1. Push\n2. Pop\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);
                               push(value);
                               break;
                         case 2: pop(); break;
                         case 3: display(); break;
                         case 4: exit(0);
                         default: printf("\nWrong selection!!! Please try again!!!\n");
    }
  }
}
void push(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  if(top == NULL)
    newNode->next = NULL;
  else
    newNode->next = top;
  top = newNode;
  printf("\nInsertion is Success!!!\n");
}
void pop()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    printf("\nDeleted element: %d", temp->data);
    top = temp->next;
    free(temp);
  }
}
void display()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    while(temp->next != NULL){
                         printf("%d--->",temp->data);
                         temp = temp -> next;
    }
    printf("%d--->NULL",temp->data);
  }}
```

OUTPUT:



```
:: Stack using Linked List ::

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

Insertion is Success!!!

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



```
4. Exit
Enter your choice: 2

Deleted element: 20
****** MENU ******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
10--->NULL
****** MENU ******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2

Deleted element: 10
****** MENU ******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:
```

RESULT: Thus the C program is implemented and verified the output for Stack using Linked list to Add, Delete and Display.

Pr.No:5a **ARRAY IMPLEMENTATION OF QUEUE** **Date:**

PROBLEM DEFINITION:
To implement a Queue using a Array.

ALGORITHM:
Queue data structure using array can be implemented as follows...
Before we implement actual operations, first follow the below steps to create an empty queue.
Step 1: Include all the header files which are used in the program and define a constant 'SIZE' with specific value.
Step 2: Declare all the user defined functions which are used in queue implementation.
Step 3: Create a one dimensional array with above defined SIZE (int queue[SIZE])
Step 4: Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1)
Step 5: Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.

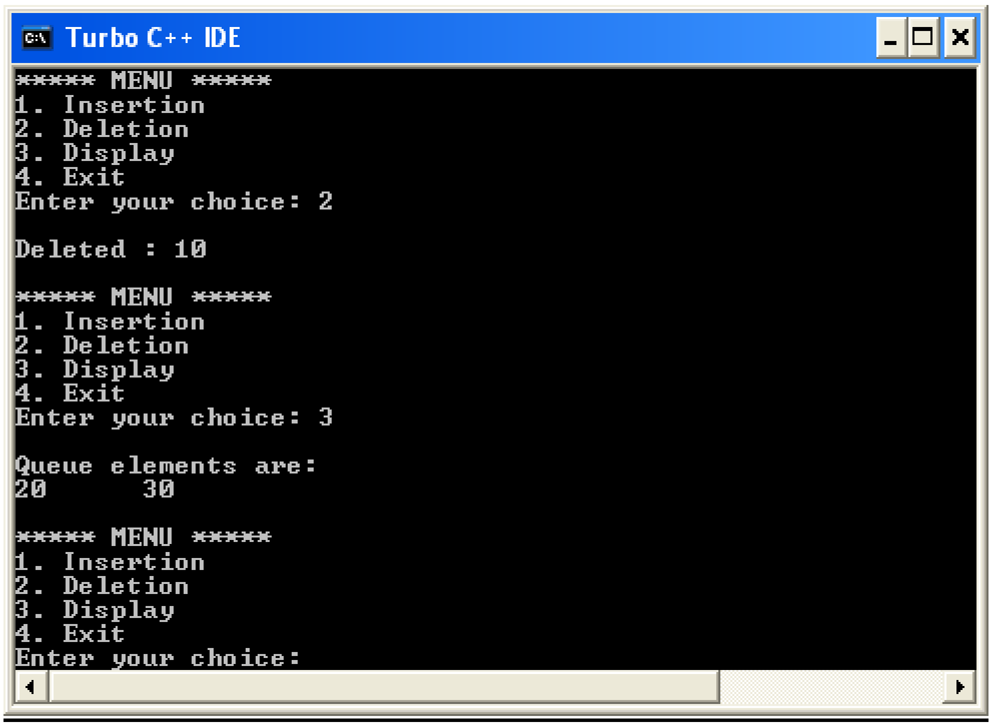
enQueue(value) - Inserting value into the queue
In a queue data structure, enQueue() is a function used to insert a new element into the queue. In a queue, the new element is always inserted at rear position. The enQueue() function takes one integer value as parameter and inserts that value into the queue. We can use the following steps to insert an element into the queue...
Step 1: Check whether queue is FULL. (rear == SIZE-1)
Step 2: If it is FULL, then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function.
Step 3: If it is NOT FULL, then increment rear value by one (rear++) and set queue[rear] = value.

deQueue() - Deleting a value from the Queue
In a queue data structure, deQueue() is a function used to delete an element from the queue. In a queue, the element is always deleted from front position. The deQueue() function does not take any value as parameter. We can use the following steps to delete an element from the queue...
Step 1: Check whether queue is EMPTY. (front == rear)
Step 2: If it is EMPTY, then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.
Step 3: If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue[front] as deleted element. Then check whether both front and rear are equal (front == rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1).

display() - Displays the elements of a Queue
We can use the following steps to display the elements of a queue...
Step 1: Check whether queue is EMPTY. (front == rear)
Step 2: If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.
Step 3: If it is NOT EMPTY, then define an integer variable 'i' and set 'i = front+1'.
Step 3: Display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i' value is equal to rear (i <= rear)

PROGRAM
```
#include<stdio.h>
#include<conio.h>
#define SIZE 10

void enQueue(int);
void deQueue();
void display();

int queue[SIZE], front = -1, rear = -1;

void main()
{
  int value, choice;
  clrscr();
  while(1){
    printf("\n\n***** MENU *****\n");
    printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice){
                        case 1: printf("Enter the value to be insert: ");
                                scanf("%d",&value);
                                enQueue(value);
                                break;
                        case 2: deQueue();
                                break;
                        case 3: display();
                                break;
                        case 4: exit(0);
                        default: printf("\nWrong selection!!! Try again!!!");
    }
  }
}
void enQueue(int value){
  if(rear == SIZE-1)
    printf("\nQueue is Full!!! Insertion is not possible!!!");
  else{
    if(front == -1)
                        front = 0;
    rear++;
    queue[rear] = value;
    printf("\nInsertion success!!!");
  }
}
void deQueue(){
  if(front == rear)
    printf("\nQueue is Empty!!! Deletion is not possible!!!");
  else{
    printf("\nDeleted : %d", queue[front]);
    front++;
    if(front == rear)
                        front = rear = -1;
  }
}
void display(){
  if(rear == -1)
    printf("\nQueue is Empty!!!");
  else{
    int i;
    printf("\nQueue elements are:\n");
    for(i=front; i<=rear; i++)
                        printf("%d\t",queue[i]);
  }
}
```

OUTPUT:



```
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2

Deleted : 10

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3

Queue elements are:
20      30

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice:
```

RESULT: Thus the C program is implemented and verified the output for Queue using Array to insert ,Delete and display.

Pr.No:5b **TO IMPLEMENT QUEUE USING LINKED LIST.** **Date:**

PROBLEM DEFINITION:
Write a C program to implement Queue using linked List

ALGORITHM:
To implement queue using linked list, we need to set the following things before implementing actual operations.

- Step 1: Include all the header files which are used in the program. And declare all the user defined functions.
- Step 2: Define a 'Node' structure with two members data and next.
- Step 3: Define two Node pointers 'front' and 'rear' and set both to NULL.
- Step 4: Implement the main method by displaying Menu of list of operations and make suitable function calls in the main method to perform user selected operation.

enQueue(value) - Inserting an element into the Queue
We can use the following steps to insert a new node into the queue...

- Step 1: Create a newNode with given value and set 'newNode → next' to NULL.
- Step 2: Check whether queue is Empty (rear == NULL)
- Step 3: If it is Empty then, set front = newNode and rear = newNode.
- Step 4: If it is Not Empty then, set rear → next = newNode and rear = newNode.

deQueue() - Deleting an Element from Queue
We can use the following steps to delete a node from the queue...

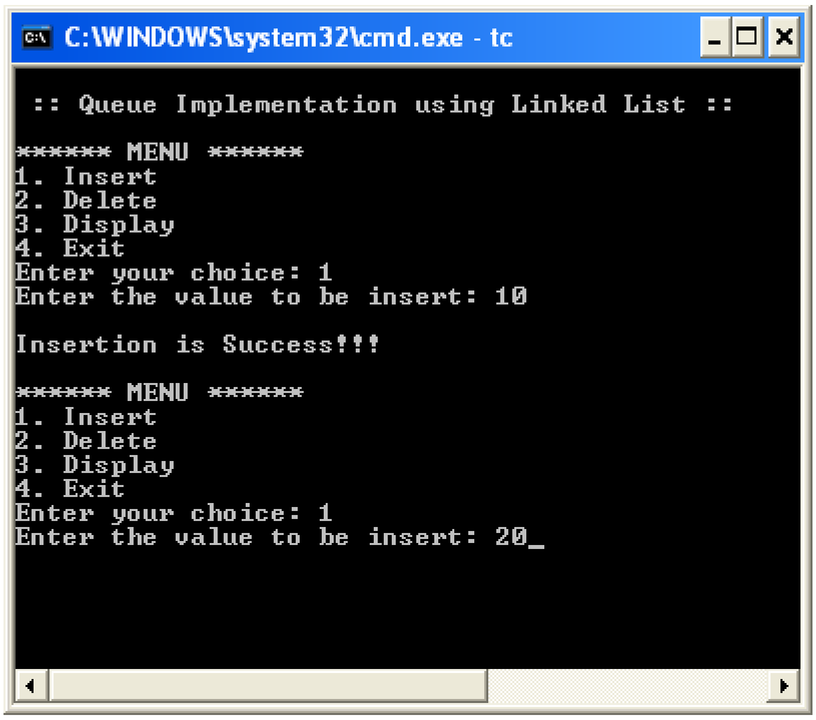
- Step 1: Check whether queue is Empty (front == NULL).
- Step 2: If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function
- Step 3: If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.
- Step 4: Then set 'front = front → next' and delete 'temp' (free(temp)).

display() - Displaying the elements of Queue
We can use the following steps to display the elements (nodes) of a queue...

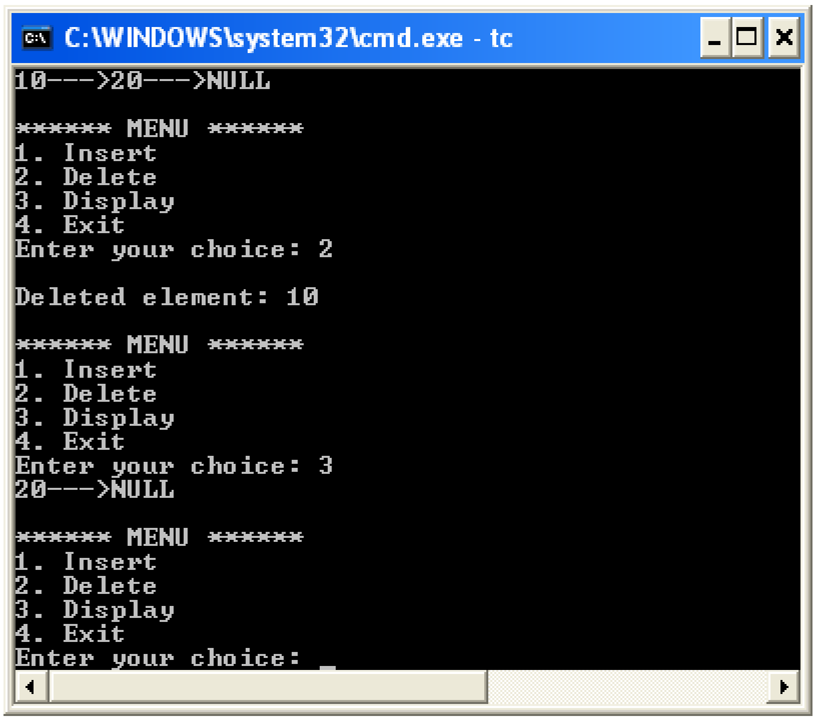
- Step 1: Check whether queue is Empty (front == NULL).
- Step 2: If it is Empty then, display 'Queue is Empty!!!' and terminate the function.
- Step 3: If it is Not Empty then, define a Node pointer 'temp' and initialize with front.
- Step 4: Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL).
- Step 4: Finally! Display 'temp → data ---> NULL'.

PROGRAM:
```
//Queue Using Linked List
#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:



```
 :: 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_
```



```
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: _
```

RESULT: Thus the C program is implemented and verified the output for Queue using Linked list to Insert, Delete and Display.

Pr.No:6 **IMPLEMENTATION OF TREE** **Date:**

PROBLEM DEFINITION:
To implement a C Program to perform operations of insertion, Deletion and Search using of Tree.
ALGORITHM:
INSERT:
1. Create a new BST node and assign values to it.
2. insert(node, key) i) If root == NULL, return the new node to the calling function. ii) if root=>data < key. call the insert function with root=>right and assign the return value in root=>right. ...
3. Finally, return the original root pointer to the calling function.

DELETE:

```
• Step 1: if tree = null. Write "item not found in the tree
  " else  if item < tree -> data.
   delete (tree->left, item)
  else if item > tree -> data.
   delete(tree -> right, item)
  Else if tree -> left and tree -> right.
  Set temp = find largest node(tree -> left)
   set tree -> data = temp -> data. ...
• Step 2: end.
```

SEARCH:

```
Step 1: Compare the current node data with the key if:
        If the key is found, then return the node.
        If the key is lesser than the node data, move the current to the left node and again
        repeat step 1.
        If the key is greater then move to the right and repeat step 1.
Step 2: If the node is not found then return NULL.
```

PROGRAM

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

struct treeNode {
      int data;
      struct treeNode *left, *right;
};

struct treeNode *root = NULL;

/* create a new node with the given data */
struct treeNode* createNode(int data) {
      struct treeNode *newNode;
      newNode = (struct treeNode *) malloc(sizeof (struct treeNode));
      newNode->data = data;
      newNode->left = NULL;
      newNode->right = NULL;
      return(newNode);
}

/* insertion in binary search tree */
```

```
void insertion(struct treeNode **node, int data) {
    if (*node == NULL) {
        *node = createNode(data);
    } else if (data < (*node)->data) {
        insertion(&(*node)->left, data);
    } else if (data > (*node)->data) {
        insertion(&(*node)->right, data);
    }
}


/* deletion in binary search tree */
void deletion(struct treeNode **node, struct treeNode **parent, int data) {
    struct treeNode *tmpNode, *tmpParent;
    if (*node == NULL)
        return;
    if ((*node)->data == data) {
        /* deleting the leaf node */
        if (!(*node)->left && !(*node)->right) {
            if (parent) {
                /* delete leaf node */
                if ((*parent)->left == *node)
                    (*parent)->left = NULL;
                else
                    (*parent)->right = NULL;
                free(*node);
            } else {
                /* delete root node with no children */
                free(*node);
            }
        /* deleting node with one child */
        } else if (!(*node)->right && (*node)->left) {
            /* deleting node with left child alone */
            tmpNode = *node;
            (*parent)->right = (*node)->left;
            free(tmpNode);
            *node = (*parent)->right;
        } else if ((*node)->right && !(*node)->left) {
            /* deleting node with right child alone */
            tmpNode = *node;
            (*parent)->left = (*node)->right;
            free(tmpNode);
            (*node) = (*parent)->left;
        } else if (!(*node)->right->left) {
            /*
             * deleting a node whose right child
             * is the smallest node in the right
             * subtree for the node to be deleted.
             */

            tmpNode = *node;

            (*node)->right->left = (*node)->left;

            (*parent)->left = (*node)->right;
            free(tmpNode);
            *node = (*parent)->left;
        } else {
                /*
                 * Deleting a node with two children.
                 * First, find the smallest node in
                 * the right subtree.  Replace the
                 * smallest node with the node to be
                 * deleted. Then, do proper connections
                 * for the children of replaced node.
                 */
                tmpNode = (*node)->right;
                while (tmpNode->left) {
                        tmpParent = tmpNode;
                        tmpNode = tmpNode->left;
                }
                tmpParent->left = tmpNode->right;
                tmpNode->left = (*node)->left;
                tmpNode->right =(*node)->right;
                free(*node);
                *node = tmpNode;
        }
    } else if (data < (*node)->data) {
        /* traverse towards left subtree */
        deletion(&(*node)->left, node, data);
    } else if (data > (*node)->data) {
        /* traversing towards right subtree */
        deletion(&(*node)->right, node, data);
    }
}

/* search the given element in binary search tree */
void findElement(struct treeNode *node, int data) {
    if (!node)
        return;
    else if (data < node->data) {
        findElement(node->left, data);
    } else if (data > node->data) {
        findElement(node->right, data);
    } else
        printf("data found: %d\n", node->data);
    return;

}

void traverse(struct treeNode *node) {
    if (node != NULL) {
        traverse(node->left);
        printf("%3d", node->data);
        traverse(node->right);
    }
    return;
}

int main() {
    int data, ch;
    while (1) {
        printf("1. Insertion in Binary Search Tree\n");
        printf("2. Deletion in Binary Search Tree\n");
        printf("3. Search Element in Binary Search Tree\n");
        printf("4. Inorder traversal\n5. Exit\n");
```

```
            printf("Enter your choice:");
            scanf("%d", &ch);
            switch (ch) {
                  case 1:
                        while (1) {
                        printf("Enter your data:");
                        scanf("%d", &data);
                        insertion(&root, data);
                        printf("Continue Insertion(0/1):");
                        scanf("%d", &ch);
                        if (!ch)
                              break;
                        }
                        break;
                  case 2:
                        printf("Enter your data:");
                        scanf("%d", &data);
                        deletion(&root, NULL, data);
                        break;
                  case 3:
                        printf("Enter value for data:");
                        scanf("%d", &data);
                        findElement(root, data);
                        break;
                  case 4:
                        printf("Inorder Traversal:\n");
                        traverse(root);
                        printf("\n");
                        break;
                  case 5:
                        exit(0);
                  default:
                        printf("u've entered wrong option\n");
                        break;
            }
      }
      return 0;
 }
```

<u>OUTPUT:</u>

```
1. Insertion in Binary Search Tree
  2. Deletion in Binary Search Tree
  3. Search Element in Binary Search Tree
  4. Inorder traversal
  5. Exit
  Enter your choice:1
  Enter your data:20
  Continue Insertion(0/1):1
  Enter your data:14
  Continue Insertion(0/1):1
  Enter your data:9
  Continue Insertion(0/1):1
  Enter your data:19
  Continue Insertion(0/1):1
  Enter your data:25
  Continue Insertion(0/1):1
  Enter your data:21
  Continue Insertion(0/1):1
```

```
 Enter your data:23
 Continue Insertion(0/1):1
 Enter your data:30
 Continue Insertion(0/1):1
 Enter your data:26
 Continue Insertion(0/1):0

Resultant Binary Search Tree after insertion operation:
                        20
                      /    \
                    14      25
                   /  \    /  \
                  9    19  21   30
                            \    /
                            23   26

 1. Insertion in Binary Search Tree
 2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit
 Enter your choice:4
 Inorder Traversal:
 9 14 19 20 21 23 25 26 30
 1. Insertion in Binary Search Tree
 2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit
 Enter your choice:2
 Enter your data:9

 Delete node 9
                        20
                      /    \
                    14      25
                      \    /  \
                       19  21   30
                            \    /
                            23   26

 1. Insertion in Binary Search Tree
 2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit
 Enter your choice:4
 Inorder Traversal:
 14 19 20 21 23 25 26 30
 1. Insertion in Binary Search Tree
 2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit
 Enter your choice:2
 Enter your data:14

 Delete node 14
```

```
                              20
                            /     \
                          19        25
                                   /  \
                                 21   30
                                  \     /
                                 23   26

1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:4
Inorder Traversal:
19 20 21 23 25 26 30
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:2
Enter your data:30

Delete node 30
                              20
                            /     \
                          19        25
                                   /  \
                                 21   26
                                  \
                                   23

1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:4
Inorder Traversal:
19 20 21 23 25 26
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:2
Enter your data:20

Delete node 20
                              21
                            /     \
                          19        25
                                   /  \
                                 23   26
```

```
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:4
Inorder Traversal:
19 21 23 25 26
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:1
Enter your data:15
Continue Insertion(0/1):1
Enter your data:14
Continue Insertion(0/1):1
Enter your data:16
Continue Insertion(0/1):1
Enter your data:17
Continue Insertion(0/1):0
Binary Search Tree After Insertion Operation:

                  21
                 /   \
               19     25
              /       / \
             15      23  26
            / \
           14  16
                \
                 17

1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:4
Inorder Traversal:
14 15 16 17 19 21 23 25 26
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:2
Enter your data:15
Delete Node 15

                  21
                 /   \
               19     25
              /       / \
             16      23  26
            / \
           14  17
```

```
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:4
Inorder Traversal:
14 16 17 19 21 23 25 26
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:3
Enter value for data:21
data found: 21
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:5
```

RESULT: Thus, the C program was implemented and performed the operations using of Tree

Pr.No:7a **SORTING ALGORITHM- INSERTION SORT** **Date:**

PROBLEM DEFINITION:

To write a program to arrange an array of numbers using Insertion Sorting Algorithm.

ALGORITHM:

Step 1 – Start the Program
Step2 - If the element is the first element, assume that it is already sorted. Return 1.
Step3 - Pick the next element, and store it separately in a key.
Step 4 - Now, compare the key with all elements in the sorted array.
Step 5 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.
Step 6 Insert the value.
Step 7 - Repeat until the array is sorted.
Step 8 – Stop the program.

PROGRAM:

```
#include <stdio.h>
 int main()
{
  int n, array[1000], c, d, t;
   printf("Enter number of elements\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (c = 0; c < n; c++)
{
    scanf("%d", &array[c]);
  }
  for (c = 1; c <= n - 1; c++)
{
    d = c;
       while (d > 0 && array[d] < array[d-1])
{
      t = array[d];
      array[d]   = array[d-1];
      array[d-1] = t;
       d--;
    }
  }
   printf("Sorted list in ascending order:\n");
   for (c = 0; c <= n - 1; c++)
{
    printf("%d\n", array[c]);
  }
   return 0;
}
```

OUTPUT:

```
Enter number of elements
5
Enter 5 integers
9 2 8 3 6
Sorted list in ascending order
2 3 6 8 9
```

RESULT: Thus, the program for sorting the given set of numbers using Inserting sort is executed successfully.

Pr.No: 7b **SELECTION SORT** **Date:**

PROBLEM DEFINITION:

To write a program to arrange an array of numbers using Selection Sorting Algorithm.

ALGORITHM:

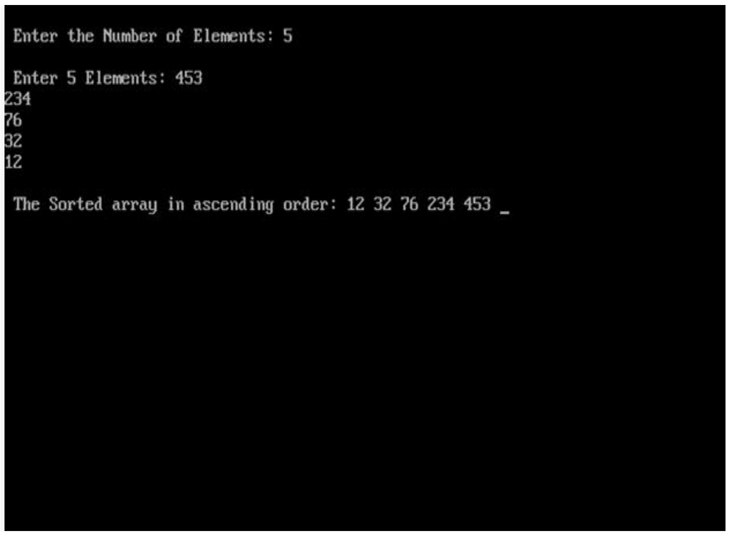
```
Step 1 − Set MIN to location 0
Step 2 − Search the minimum element in the list
Step 3 − Swap with value at location MIN
Step 4 − Increment MIN to point to next element
Step 5 − Repeat until list is sorted
```

PROGRAM:

//Selection Sort

```
#include<stdio.h>

#include<conio.h>

void main()

{

int a[100],n,i,j,min,temp;

clrscr();

printf("\n Enter the Number of Elements: ");

 scanf("%d",&n);

printf("\n Enter %d Elements: ",n);

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

            {

             scanf("%d",&a[i]);

            }

  for(i=0;i<n-1;i++)

            {

            min=i;

             for(j=i+1;j<n;j++)

            {
                    if(a[min]>a[j])
                    min=j;
                    }
        if(min!=i)
        {
        temp=a[i];
        a[i]=a[min];
        a[min]=temp;
        }
        }
        printf("\n The Sorted array in ascending order: ");
        for(i=0;i<n;i++)
        {
          printf("%d ",a[i]);
        }
getch();
}
```

OUTPUT:



```
 Enter the Number of Elements: 5

 Enter 5 Elements: 453
234
76
32
12

 The Sorted array in ascending order: 12 32 76 234 453 _
```

RESULT: Thus, the program for sorting the given set of numbers using Selection sort is executed successfully.

Pr.No:7c **MERGE SORT** **Date:**

PROBLEM DEFINITION:

To write a program to arrange an array of numbers using Merge Sorting Algorithm.

ALGORITHM:

```
Step 1: Start the program.
Step 2: Begin MergeSort (arr[], low,  high)
Step 3: If high>low, Begin
Step 4:Find the middle point to divide the array into two halves:
        middle mid = l+ (high-low)/2
Step 5: Call mergeSort for first half:
        Call mergeSort(arr, low, mid)
Step 6: Call mergeSort for second half:
        Call mergeSort(arr, mid+1, low)
Step 7:Merge the two halves sorted in step 2 and 3:
        Call merge(arr, l, mid, high)
Step 8: Stop the program
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int K[10],N=10;
merge(int low, int mid, int high)
{
int i=low, j=mid+1, l=0, temp[N];
while (i<=mid && j<=high){
        if (K[i]<K[j]){
                temp[l]=K[i];
                l++;
                i++;
}
        else{
                temp[l]=K[j];
                l++;
                j++;
        }
if (i>mid)
```

```
          while (j<=high){
                    temp[l]=K[j];
                    l++;
                    j++;
          }
else
          while (i<=mid){
                    temp[l]=K[i];
                    l++;
                    j++:
          }
for (m=0; m<=l; m++)
          K[low+m]=temp[m];
}
mergesort(int low, int high)
{
          int mid;
          if (low<high)
          {
                    mid=(low+high)/2;
                    mergesort(low,mid);
                    mergesort(mid+1, high);
                    merge(low, mid, high);
          }
}
main(){
int i;
printf("\nEnter the values);
for(i=0;i<N;i++)
          scanf('%d",&K[i]);
mergesort(0,N-1);
printf("\nThe Sorted Values");
for(i=0;i<N;i++)
          printf('%d",K[i]);
```

OUTPUT:

```
Enter the values
10 14 19 26 27 31 33 35 42 0

The Sorted Values
0 10 14 19 26 27 31 33 35 42
```

RESULT:

Thus, the program for sorting the given set of numbers using Merge sort is executed successfully.

Pr.No :8 **Date:**

IMPLEMENTING HASH TABLE IN C

PROBLEM DEFINITION:
Create a Hash table in C and values into the hash table by using a hash function.

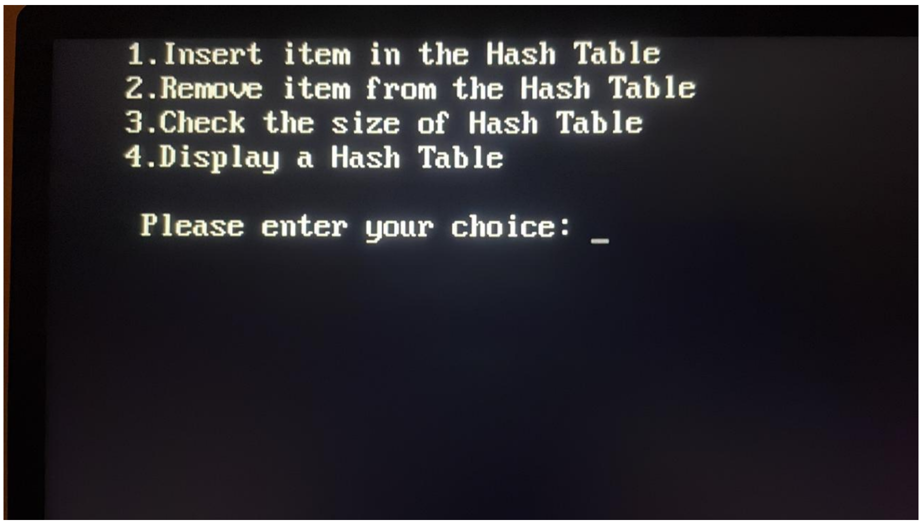
ALGORITHM:

- Hash Table is a data structure which stores data in an associative manner.
- The data is stored in an array format where each data value has its own unique index value.

PROGRAM:

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

struct set
{
  int key;
  int data;
};
struct set *array;
int capacity = 10;
int size = 0;

int hashFunction(int key)
{
  return (key % capacity);
}
int checkPrime(int n)
{
  int i;
  if (n == 1 || n == 0)
  {
  return 0;
  }
  for (i = 2; i < n / 2; i++)
  {
  if (n % i == 0)
  {
    return 0;
  }
  }
  return 1;
}
int getPrime(int n)
{
  if (n % 2 == 0)
  {
  n++;
  }
  while (!checkPrime(n))
  {
  n += 2;
  }
  return n;
}
void init_array()
{
  capacity = getPrime(capacity);
  array = (struct set *)malloc(capacity * sizeof(struct set));
  for (int i = 0; i < capacity; i++)
  {
  array[i].key = 0;
  array[i].data = 0;
  }
}

void insert(int key, int data)
{
  int index = hashFunction(key);
  if (array[index].data == 0)
  {
  array[index].key = key;
  array[index].data = data;
  size++;
  printf("\n Key (%d) has been inserted \n", key);
  }
  else if (array[index].key == key)
  {
  array[index].data = data;
  }
  else
  {
  printf("\n Collision occured  \n");
  }
}

void remove_element(int key)
{
  int index = hashFunction(key);
  if (array[index].data == 0)
  {
  printf("\n This key does not exist \n");
  }
  else
  {
  array[index].key = 0;
  array[index].data = 0;
  size--;
  printf("\n Key (%d) has been removed \n", key);
  }
}
void display()
{
  int i;
  for (i = 0; i < capacity; i++)
  {
  if (array[i].data == 0)
  {
    printf("\n array[%d]: / ", i);
  }
  else
  {
    printf("\n key: %d array[%d]: %d \t", array[i].key, i, array[i].data);
  }
  }
}

int size_of_hashtable()
{
  return size;
}

int main()
{
  int choice, key, data, n;
  int c = 0;
  init_array();

  do
  {
  printf("1.Insert item in the Hash Table"
     "\n2.Remove item from the Hash Table"
     "\n3.Check the size of Hash Table"
     "\n4.Display a Hash Table"
     "\n\n Please enter your choice: ");

  scanf("%d", &choice);
  switch (choice)
  {
  case 1:

    printf("Enter key -:\t");
    scanf("%d", &key);
    printf("Enter data -:\t");
    scanf("%d", &data);
    insert(key, data);

    break;

  case 2:

    printf("Enter the key to delete-:");
    scanf("%d", &key);
    remove_element(key);

    break;
    case 3:
    n = size_of_hashtable();
    printf("Size of Hash Table is-:%d\n", n);
    break;
    case 4:
    display();
    break;
  default:
  printf("Invalid Input\n");
  }
  printf("\nDo you want to continue (press 1 for yes): ");
  scanf("%d", &c);

  } while (c == 1);
}
```

OUTPUT



```
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice: _
```

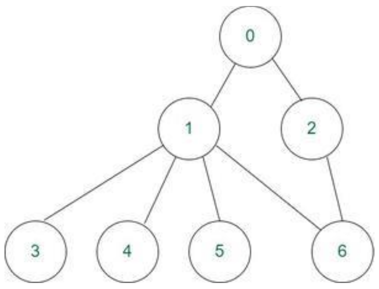
RESULT: Successfully Created a Hash table in C and values into the hash table by using a hash function.

Pr. No:9a **Date:**

DEPTH FIRST GRAPH TRAVERSAL

PROBLEM DEFINITION:

To implement Depth first graph traversal using adjacency matrix for the given graph.



ALGORITHM:

1. Initially all vertices are marked unvisited (false).
2. The DFS algorithm starts at a vertex **u** in the graph. By starting at vertex **u** it considers the edges from **u** to other vertices.
   a. If the edge leads to an **already visited vertex**, then backtrack to current vertex **u**.
   b. If an edge leads to an **unvisited vertex,** then go to that vertex and start processing from that vertex. That means the new vertex becomes the current root for traversal.
3. Follow this process until a vertices are marked visited.

PROGRAM:

```
// DFS algorithm in C

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

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

struct node* createNode(int v);

struct Graph {
  int numVertices;
  int* visited;

  // We need int** to store a two dimensional array.
  // Similary, we need struct node** to store an array of Linked lists
  struct node** adjLists;
};

// DFS algo
void DFS(struct Graph* graph, int vertex) {
  struct node* adjList = graph->adjLists[vertex];
  struct node* temp = adjList;

  graph->visited[vertex] = 1;
printf("Visited %d \n", vertex);

  while (temp != NULL) {
    int connectedVertex = temp->vertex;

    if (graph->visited[connectedVertex] == 0) {
DFS(graph, connectedVertex);
    }
    temp = temp->next;
  }
}

// Create a node
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
newNode->vertex = v;
newNode->next = NULL;
  return newNode;
}
     int i;
// Create graph
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));

  graph->visited = malloc(vertices * sizeof(int));
  for (i = 0; i< vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}

// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
  // Add edge from src to dest
  struct node* newNode = createNode(dest);
newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  // Add edge from dest to src
newNode = createNode(src);
newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}

// Print the graph
void printGraph(struct Graph* graph) {
  int v;
  for (v = 0; v < graph->numVertices; v++) {
    struct node* temp = graph->adjLists[v];
printf("\n Adjacency list of vertex %d\n ", v);
    while (temp) {
printf("%d -> ", temp->vertex);
      temp = temp->next;
    }
printf("\n");
  }
}

int main() {
  struct Graph* graph = createGraph(4);
addEdge(graph, 0, 1);
addEdge(graph, 0, 2);
addEdge(graph, 1, 2);
addEdge(graph, 2, 3);

printGraph(graph);

DFS(graph, 2);

  return 0;
}
```

OUTPUT:

```
Adjacency list of vertex 0
2 -> 1->
Adjacency list of vertex 1
2->0->
Adjacency list of vertex 2
3 -> 1 -> 0 ->
Ad jacency list of vertex 3
2 ->
Visited 2
Visited 3
Visited 1
Visited 0
```

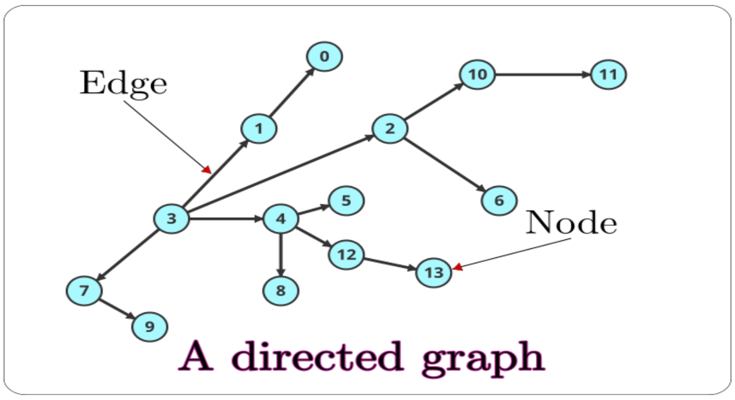
RESULT:

Thus the implementation of Depth first graph traversal using adjacency matrix for the given graph was executed successfully.

Pr.No:9b **BREADTH FIRST GRAPH TRAVERSAL** **Date:**

PROBLEM DEFINITION:

To implement Breadth first graph traversal using adjacency matrix for the given graph.



ALGORITHM:

1. Start with the initial node.
2. Mark the initial node as visited and enqueue it.
3. While the queue is not empty:

   - Dequeue a node from the queue.
   - If the dequeued node is the goal node, stop the search.
   - Otherwise, enqueue any successors (nodes that are directly connected to the dequeued node) that have not yet been visited.

4. If the queue is empty, every node on the graph has been visited, or there is no path from the initial node to the goal node.
5. If the goal node was found, return the path that was followed.

PROGRAM:

```
#include <stdio.h>

int n,i, j,visited[10], queue[10], front =-1, rear =-1;
intadj[10][10];

voidbfs(int v)
{
for(i=1;i<= n;i++)
if(adj[v][i]&&!visited[i])
         queue[++rear]=i;
if(front <= rear)
{
     visited[queue[front]]=1;
bfs(queue[front++]);
}
}

voidmain()
```

```
{
int v;
printf("Enter the number of vertices: ");
scanf("%d",&n);
for(i=1;i<= n;i++)
{
        queue[i]=0;
        visited[i]=0;
}
printf("Enter graph data in matrix form:    \n");
for(i=1;i<= n;i++)
for(j =1; j <= n;j++)
scanf("%d",&adj[i][j]);
printf("Enter the starting vertex: ");
scanf("%d",&v);
bfs(v);
printf("The node which are reachable are:    \n");
for(i=1;i<= n;i++)
if(visited[i])
printf("%d\t",i);
else
printf("BFS is not possible. Not all nodes are reachable");
return0;
}
```

OUTPUT:

```
Enter the number of vertices: 4
Enter graph data in matrix form:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Enter the starting vertex: 2
The node which are reachable are:
1       2       3       4
```

RESULT:

Thus the implementation of Breadth first graph traversal using adjacency matrix for the given graph was executed successfully.

Pr. No:10 SHORTEST PATH USING DIJKSTRA'S ALGORITHM IN C Date:

PROBLEM DEFINITION:
To write a C program to implement the shortest path using dijkstra’s algorithm
ALGORITHM:

```
Step 1 : Create a set shortPath to store vertices that come in the way of the shortest path tree.
Step 2 : Initialize all distance values as INFINITE and assign distance values as 0 for source
vertex
         so that it is picked first.
Step 3 : Loop until all vertices of the graph are in the shortPath.
         a : Take a new vertex that is not visited and is nearest.
         b : Add this vertex to shortPath.
         c : For all adjacent vertices of this vertex update distances. Now check every adjacent
vertex of V, if sum of distance of u and weight of edge is elss the update it.
```

PROGRAM:

```c
// A C program for Dijkstra's single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

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

// Number of vertices in the graph
#define V 9

// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{
   // Initialize min value
   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++)
      if (sptSet[v] == false && dist[v] <= min)
          min = dist[v], min_index = v;

   return min_index;
}

// A utility function to print the constructed distance array
int printSolution(int dist[], int n)
{
   printf("Vertex   Distance from Source\n");
   for (int i = 0; i < V; i++)
      printf("%d \t\t %d\n", i, dist[i]);
}
 // Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
   int dist[V]; // The output array.  dist[i] will hold the shortest
   // distance from src to i
```

```
    bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
    // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to  v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist, V);
}

// driver program to test above function
int main()
{
    /* Let us create the example graph discussed above */


 int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    dijkstra(graph, 0);

    return 0;
}
```

OUTPUT:

```
Vertex   Distance from Source
          0         0
          1         4
          2         12
          3         19
          4         21
          5         11
          6         9
          7         8
          8         14
```

RESULT:
Thus the C program to implement the shortest path using Dijkstra’s algorithm was executed successfully.