**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**DATA STRUCTURES (SE201)**



**DELHI TECHNOLOGICAL UNIVERSITY(DTU)**

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Bawana Road, Delhi-110042

Submitted by:-

Name-: ASHISH KUMAR

Roll number-: 2K18/SE/041

**Vision and Mission of Department of**

**Computer Science & Engineering**

**VISION**

To be a leading world class technology department playing its role as a key node in national and global knowledge network, thus empowering the computer science industry with the wings of knowledge and power of innovation.

**MISSION**

* To nurture talent of students for research, innovation and excellence in the field of computer engineering starting from Under graduate level.
* To develop highly analytical and qualified computer engineers by importing training on cutting edge technology.
* To produce socially sensitive computer engineers with professional ethics.
* To focus R&D environment in close partnership with industry and universities.
* To produce well-rounded, up to date, scientifically tempered, design oriented engineer and scientists capable of lifelong learning.

**PROGRAM 1**

**Aim:-**To write a program on Binary Search.

**Theory:-** Binary search is an algorithm for locating the position of an item in an sorted array. 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 middle of 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.

**Algorithm:-**

Step 1: Take elements (from user) to be stored in an array.

Step 2: Now take the number that user wanted to search in array(or given list of numbers provided by user) and stored in variable , named as x and initialize the first element with zero and last element with n-1 ( where n is size of array).

Step 3: Now find mid element by mid=(first + last)/2.

Step 4: Compare x with the middle element only if first element is less than last element (this can be check by while loop)

Step 5: If x matches with middle element, return mid index (it means element is found in array).

Step 6: Otherwise, if x is greater than mid element, then x can only lie in right half array after the mid element. So we now consider right half array.

Step 7: else(x is smaller) recur for the left half (it means element not found).

**CODE:-**

#include <iostream.h>

#include <conio.h>

int binarySearch(int arr[], int l, int r, int x)

{

while (l <= r) {

int m = l + (r - l) / 2;

if (arr[m] == x)

return m;

if (arr[m] < x)

l = m + 1;

else

r = m - 1;

}

return -1;

}

void main()

{

clrscr();

int arr[50];

int n,x;

cout<<"Enter the size of array:";

cin>>n;

cout<<"Enter"<<" "<<n<<" "<<"elements:";

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

{

cin>>arr[i];

}

cout<<"Enter element to search:";

cin>>x;

int result = binarySearch(arr, 0, n - 1, x);

(result == -1) ? cout << "Element is not present in array or array is in unsorted form" : cout << "Element is present at position " << result+1;

getch();

}

**OUTPUT:-**

****

**Learning & Finding:-**

Through C++ Programming, we can easily do Binary search(in arrays)which is used to search an element in any sorted list of numbers.

**PROGRAM 2**

**Aim:-**To write a program to implement a stack using an array.

**Theory:-** A Stack is a linear data structure that can be defined as a list of items in which additions can be done from only one direction (usually known as top). Stacks can be implemented in two ways using C. First one is Arrays and the second is Linked List. In a Stack a item which is usually pushed first is popped out at the last. That means Last in First Out[**LIFO**] or First in Last Out[**FILO**]. Stack has two functions: - Push and Pop.

Push is used to insert elements in a list, whereas Pop is used to remove elements from the list.

**ALGORITHM:-**

In stack related algorithms TOP initially point 0, index of elements in stack is start from 1, and index of last element is MAX.

**INIT\_STACK (STACK, TOP)**

Algorithm to initialize a stack using array.

TOP points to the top-most element of stack.

1) TOP: = 0;

2) Exit

**Push operation is used to insert an element into stack.**

**PUSH\_STACK(STACK,TOP,MAX,ITEM)**

Algorithm to push an item into stack.

1) IF TOP = MAX then

Print “Stack is full”;

Exit;

2) Otherwise

TOP: = TOP + 1; /\*increment TOP\*/

STACK [TOP]:= ITEM;

3) End of IF

4) Exit

**Pop operation is used to remove an item from stack, first get the element and then decrease TOP pointer.**

**POP\_STACK(STACK,TOP,ITEM)**

Algorithm to pop an element from stack.

1) IF TOP = 0 then

Print “Stack is empty”;

Exit;

2) Otherwise

ITEM: =STACK [TOP];

TOP:=TOP – 1;

3) End of IF

4) Exit

**CODE:-**

#include<stdio.h>

#include<iostream.h>

#include<conio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

void topelement(void);

void main()

{

clrscr();

top=-1;

printf("\n Enter the size of STACK[MAX=100]:");

scanf("%d",&n);

printf("\n\t STACK OPERATIONS USING ARRAY");

printf("\n\t--------------------------------");

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

printf("\n\t 5.Size");

cout<<"\n\t 6.Top element";

do

{

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("\n\t EXIT POINT ");

break;

}

case 5:

printf("Stack size: %d\n", top + 1);

break;

case 6:

topelement();

break;

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

}

while(choice!=4);

getch();

}

void push()

{

if(top>=n-1)

{

printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

void pop()

{

if(top<=-1)

{ printf("\n\t Stack is under flow");

}

else

{

printf("\n\t The popped elements is %d",stack[top]);

top--;

}

}

void display()

{

if(top>=0)

{

printf("\n The elements in STACK \n");

for(i=top; i>=0; i--)

cout<<stack[i]<<"->";

cout<<"END";

printf("\n Press Next Choice");

}

else

{

printf("\n The STACK is empty");

}

}

void topelement()

{

if(top < 0)

{

cout<<"Stack is empty";

}

else {

int rv = stack[top];

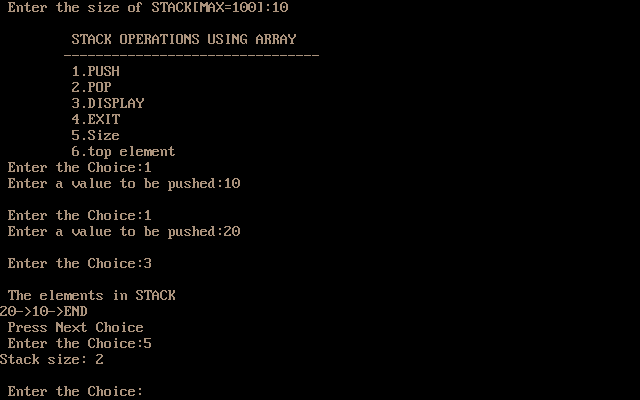
cout<<"TOP Element is:"<<rv;

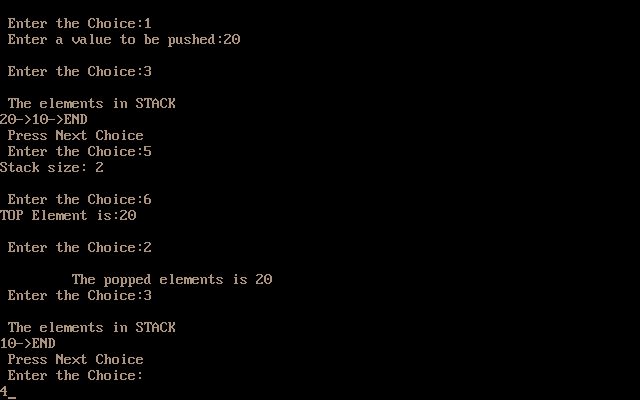
cout<<endl;

}

}

**OUTPUT:-**

****

****

**Learning & Finding:-** Through C++ Programming, we can easily implement a Stack and can perform it’s various functions using . And we found that queue follows LIFO approach.

**PROGRAM 3**

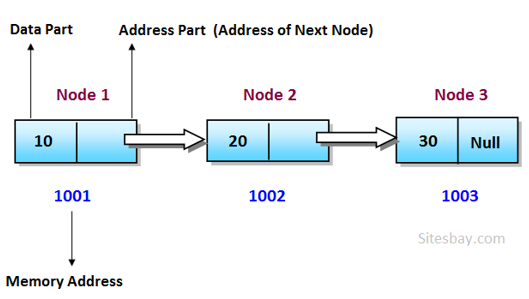
**Aim:-**To write a program to perform insertion and deletion in a ` linkedlist.

**Theory:-**

Linked list is one of the most important data structures. We often face situations, where the data is dynamic in nature and number of data can’t be predicted or the number of data keeps changing during program execution. Linked lists are very useful in this type of situations.

The implementation of a linked list in C++ is done using pointers.

 A linked list is made up of many nodes which are connected in nature. Every node is mainly divided into two parts, one part holds the data and the other part is connected to a different node.



**ALGORITHM:-**

## How to traverse a linked list

Displaying the contents of a linked list is very simple. We keep moving the temp node to the next one and display its contents.

When temp is NULL, we know that we have reached the end of linked list so we get out of the while loop.

1. struct node \*temp = head;
2. printf("\n\nList elements are - \n");
3. while(temp != NULL)
4. {
5. printf("%d --->",temp->data);
6. temp = temp->next;
7. }

## How to add elements to linked list

You can add elements to either beginning, middle or end of linked list.

### Add to beginning

* Allocate memory for new node
* Store data
* Change next of new node to point to head
* Change head to point to recently created node

1. struct node \*newNode;
2. newNode = malloc(sizeof(struct node));
3. newNode->data = 4;
4. newNode->next = head;
5. head = newNode;

### Add to end

* Allocate memory for new node
* Store data
* Traverse to last node
* Change next of last node to recently created node

1. struct node \*newNode;
2. newNode = malloc(sizeof(struct node));
3. newNode->data = 4;
4. newNode->next = NULL;
5. struct node \*temp = head;
6. while(temp->next != NULL){
7. temp = temp->next;
8. }
9. temp->next = newNode;

## How to delete from a linked list

You can delete either from beginning, end or from a particular position.

### Delete from beginning

* Point head to the second node

1. head = head->next;

### Delete from end

* Traverse to second last element
* Change its next pointer to null

1. struct node\* temp = head;
2. while(temp->next->next!=NULL){
3. temp = temp->next;
4. }
5. temp->next = NULL;

**CODE:-**

#include<iostream.h>

#include<conio.h>

#include<malloc.h>

struct node

{

int info;

struct node \*next;

}\*start, \*temp;

void insert\_beg(int);

void insert\_end(int);

void insert\_spe(int,int);

void delete\_beg();

void delete\_end();

void delete\_spe(int);

void display();

void main()

{ clrscr();

start=NULL;

int item,choice,location,element,position;

cout<<endl<<"Insert & Delete operation in linked list \n";

again:

cout<<"\n1.Insert at beg of linked list\t";

cout<<"2.Insert at end of linked list\n";

cout<<"3.Insert at specific location\t";

cout<<"4.Delete from beginning of List \n5.delete from end of the list \t6.Delete from specific location of List\n";

cout<<"7.Display linked list\t";

cout<<"8.Exit\n";

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:cout<<"Enter item to insert : ";

cin>>item;

insert\_beg(item);

goto again;

case 2: cout<<"Enter item to insert : ";

cin>>item;

insert\_end(item);

goto again;

case 3: cout<<"Enter location to insert : ";

cin>>location;

cout<<"Enter item to insert : ";

cin>>item;

insert\_spe(item,location);

goto again;

case 4: delete\_beg();

goto again;

case 5: delete\_end();

goto again;

case 6: cout<<"Enter the location to delete : ";

cin>>location;

delete\_spe(location);

goto again;

case 7:cout<<"\nInserted items are = ";

display();

goto again;

case 8 :cout<<"\nEND";

default: break;

}

getch();

}

void insert\_beg(int item)

{

temp= (node\*)malloc(sizeof(node));

temp->info=item;

temp->next=start;

start=temp;

}

void insert\_end(int item)

{

temp= (node\*)malloc(sizeof(node));

temp->info=item;

temp->next=NULL;

node\* current=start;

while(current->next!=NULL)

{

current=current->next;

}

current->next=temp;

}

void insert\_spe(int item,int location)

{

temp=(node\*)malloc(sizeof(node));

temp->info=item;

node\* current=start;

int count=1;

while (count <location-1)

{ current=current->next;

count=count+1;

}

temp->next=current->next;

current->next=temp;

}

void delete\_beg()

{ temp=start;

start=start->next;

free(temp);

}

void delete\_end()

{

temp=start;

node\* current;

while(temp->next!=NULL)

{

current=temp;

temp=temp->next;

}

current->next=NULL;

free(temp);

}

void delete\_spe(int location)

{

node\* current;

temp=start;

int count=1;

while(count<=location-1)

{ current=temp;

temp=temp->next;

count=count+1;

}

current->next=temp->next;

free(temp);

}

void display()

{ temp=start;

while(temp!=NULL)

{

cout<<temp->info<<" ";

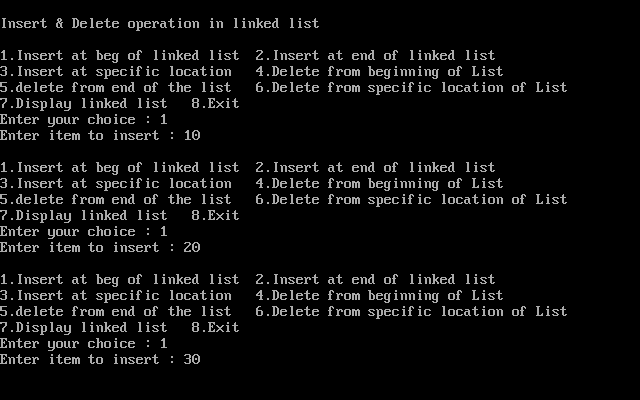
temp=temp->next;

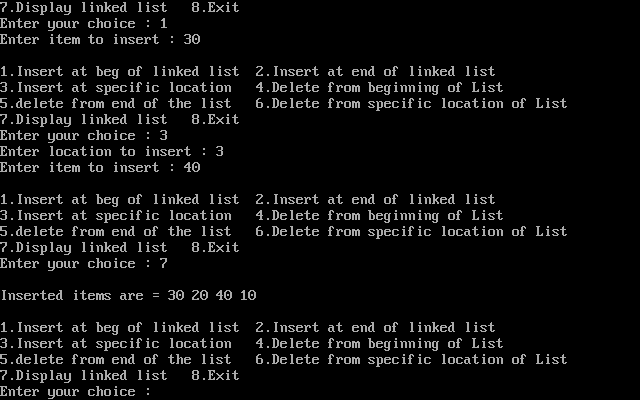
}

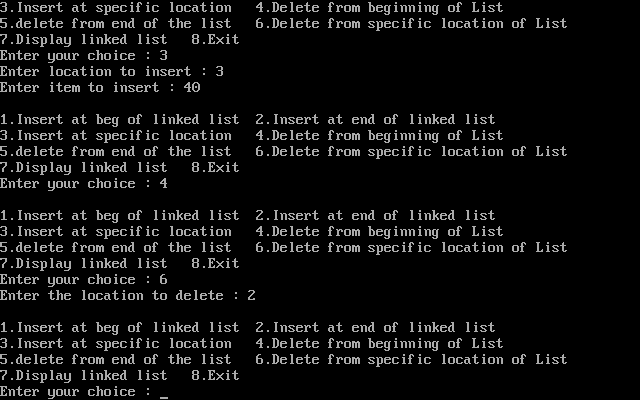
cout<<endl;

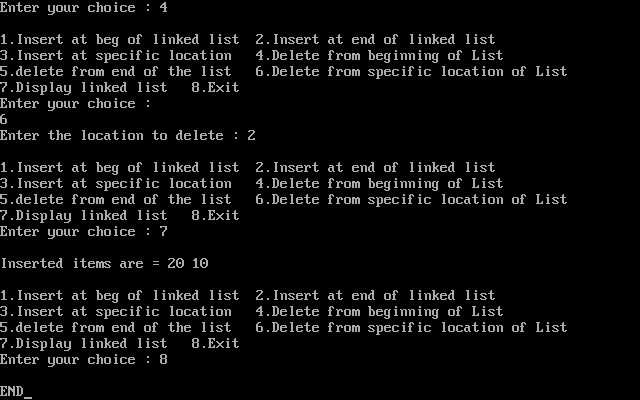
}

**OUTPUT:-**

****

****

****

****

**Learning & Finding:-** Through C++ Programming, we can insert and delete any element in Linkedlist. And we observed that linkedlists are dynamic and can expand its size during runtime. Operations like insertion and deletion in lnkedlist is fast as compared to arrays.

**PROGRAM 4**

**Aim:-**To write a program to implement a queue using Linked list.

**Theory:-** A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First in First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

* It has three components:
  + A **Container of items** that contains elements of queue.
  + A pointer **front** that points the first item of the queue.
  + A pointer **rear** that points the last item of the queue.
* Insertion is performed from **REAR** end.
* Deletion is performed from **FRONT** end.
* Insertion operation is also known as **ENQUEUE** in queue.

**ALGORITHM:-**

**Operations**

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.

**enq(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**.

**deq() - 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 5 -**Finally! Display '**temp → data** ---> **NULL**'.

**CODE:-**

#include <stdio.h>

#include <stdlib.h>

#include<conio.h>

#include<iostream.h>

struct node

{

int info;

struct node \*ptr;

}\*front,\*rear,\*temp,\*front1;

int frontelement();

void enq(int data);

void deq();

void empty();

void display();

void create();

void queuesize();

int count = 0;

void main()

{ clrscr();

int no, ch, e;

printf("\n 1 - Enque(insert)");

printf("\n 2 - Deque(delete)");

printf("\n 3 - Front element");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Display");

printf("\n 7 - Queue size");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

enq(no);

break;

case 2:

deq();

break;

case 3:

e = frontelement();

if (e != 0)

printf("Front element : %d", e);

else

printf("\n No front element in Queue as queue is empty");

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

queuesize();

break;

default:

printf("Wrong choice, Please enter correct choice ");

break;

}

}

getch();

}

/\* Create an empty queue \*/

void create()

{

front = rear = NULL;

}

/\* Returns queue size \*/

void queuesize()

{

printf("\n Queue size : %d", count);

}

/\* Enqueing the queue \*/

void enq(int data)

{

if (rear == NULL)

{

rear = (struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = NULL;

rear->info = data;

front = rear;

}

else

{

temp=(struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = temp;

temp->info = data;

temp->ptr = NULL;

rear = temp;

}

count++;

}

/\* Displaying the queue elements \*/

void display()

{

front1 = front;

if ((front1 == NULL) && (rear == NULL))

{

printf("Queue is empty");

return;

}

while (front1 != rear)

{

cout<< front1->info<<"->";

front1 = front1->ptr;

}

if (front1 == rear)

cout<< front1->info<<"->";

cout<<"END";

}

/\* Dequeing the queue \*/

void deq()

{

front1 = front;

if (front1 == NULL)

{

printf("\n Error: Trying to display elements from empty queue");

return;

}

else

if (front1->ptr != NULL)

{

front1 = front1->ptr;

printf("\n Dequed value : %d", front->info);

free(front);

front = front1;

}

else

{

printf("\n Dequed value : %d", front->info);

free(front);

front = NULL;

rear = NULL;

}

count--;

}

/\* Returns the front element of queue \*/

int frontelement()

{

if ((front != NULL) && (rear != NULL))

return(front->info);

else

return 0;

}

/\* Display if queue is empty or not \*/

void empty()

{

if ((front == NULL) && (rear == NULL))

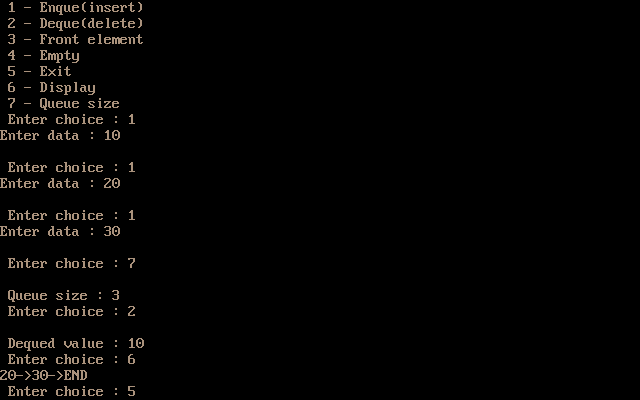
printf("\n Queue empty");

else

printf("Queue not empty");

}

**OUTPUT:-**

****

**Learning & Finding:-** Through C++ Programming, we can easily implement a Queue and can perform it’s various functions using linkedlist. And we found that queue follows FIFO approach

**PROGRAM 5**

**Aim:-**To write a C program to implement a Queue using an array and perform basic queue operations.

**Theory:-** A Queue is a linear structure which follows a particular order in which the operations are performed i.e, The order is First in First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

**ALGORITHM:-**

Implement following steps:-

1. Ask the user for the operation like insert, delete, display and exit.  
2. According to the option entered, access its respective function using switch statement. Use the variables front and rear to represent the first and last element of the queue.  
3. In the function insert(), firstly check if the queue is full. If it is, then print the output as “Queue Overflow”. Otherwise take the number to be inserted as input and store it in the variable add\_item. Copy the variable add\_item to the array queue\_array[] and increment the variable rear by 1.  
4. In the function delete(), firstly check if the queue is empty. If it is, then print the output as “Queue Underflow”. Otherwise print the first element of the array queue\_array[] and decrement the variable front by 1.  
5. In the function display(), using for loop print all the elements of the array starting from front to rear.  
6. Exit.

**CODE:-**

#include<stdio.h>

#include<iostream.h>

#include<conio.h>

#define n 5 //array’s size

int main()

{

int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;

printf("Implementing Queue using Array");

printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");

while(ch)

{

printf("\nEnter the Choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

if(rear==x)

printf("\n Queue is Full");

else

{

printf("\n Enter no %d:",j++);

scanf("%d",&queue[rear++]);

}

break;

case 2:

if(front==rear)

{

printf("\n Queue is empty");

}

else

{

printf("\n Deleted Element is %d",queue[front++]);

x++;

}

break;

case 3:

printf("\n Queue Elements are:\n ");

if(front==rear)

printf("\n Queue is Empty");

else

{

for(i=front; i<rear; i++)

{

printf("%d",queue[i]);

printf("->");

}

printf("END");

break;

case 4:

exit(0);

default:

printf("Wrong Choice: please see the options");

}

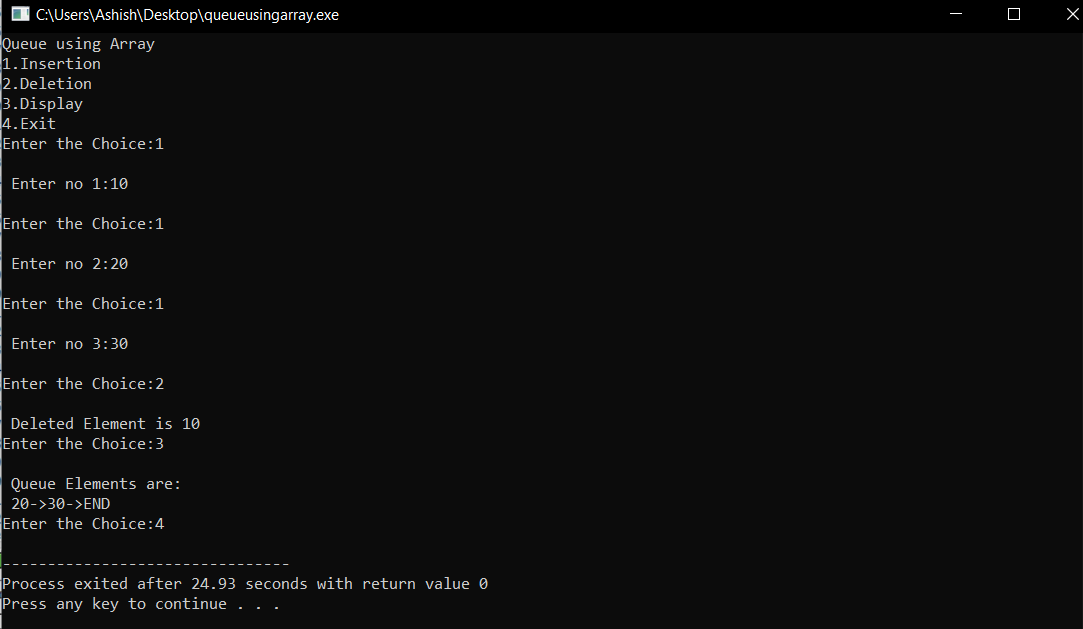
}

}

return 0;

}

**OUTPUT:-**

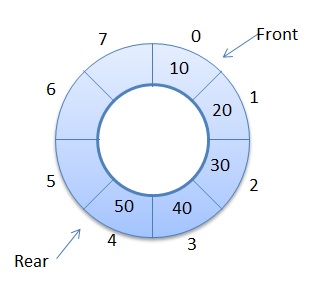


**Learning & Finding:-** Through C Programming, we can easily implement a Queue and can perform it’s various functions using Array, and also we found that queue follows FIFO approach.

**PROGRAM 6**

**Aim:-**To write a C program to implement a Circular Queue.

**Theory:-** A **circular queue** is a very important data structure because it can store data in a very practical way. **The circular queue is a linear data structure. It follows FIFO principle. In circular queue, the last node is connected back to the first node to make a circle**. Elements are added at the rear end and the elements are deleted at the front end of the queue.



**ALGORITHM:-**

In the Code below there are four parts. First three function to implement three different operations like **Insert a node**, **delete a node** and **display queue**. The Fourth part is the **main function**, in that a do while loop is implemented to keep the user engaged and provide him the all the given choices, according to the choice one of the three function get called.

The First function checks whether the queue is empty, rear is at last position of queue or Queue is full. If the queue is not full it adds up the item.

The Second function checks whether the list is empty, full or it has only one item. If any node exists, it will delete the node in FIFO order.

The Third function will simply print all the elements of the Queue if exist. If not, then it will say Queue is Empty.

**CODE:-**

#include<stdio.h>

#include<iostream.h>

#include<conio.h>

#define max 5

int q[10],front=0,rear=-1;

int main()

{

int ch;

void insert();

void delete();

void display();

//clrscr();

printf("\n Implementation of Circular Queue \n");

printf("1.insert\n2.delete\n3.display\n4.exit\n");

while(1)

{

printf("Enter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1: insert();

break;

case 2: delete();

break;

case 3:display();

break;

case 4:

exit(0);

default:printf("Invalid option\n");

}

}

return 0;

}

void insert()

{

int x;

if((front==0&&rear==max-1)||(front>0&&rear==front-1))

printf("Queue is overflow\n");

else

{

printf("Enter element to be insert:");

scanf("%d",&x);

if(rear==max-1&&front>0)

{

rear=0;

q[rear]=x;

}

else

{

if((front==0&&rear==-1)||(rear!=front-1))

q[++rear]=x;

}

}

}

void delete()

{

int a;

if((front==0)&&(rear==-1))

{

printf("Queue is underflow\n");

return;

}

if(front==rear)

{

a=q[front];

rear=-1;

front=0;

}

else

if(front==max-1)

{

a=q[front];

front=0;

}

else a=q[front++];

printf("Deleted element is:%d\n",a);

}

void display()

{

int i,j;

if(front==0&&rear==-1)

{

printf("Queue is underflow\n");

return;

}

if(front>rear)

{

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

printf("%d",q[i]);

printf("->");

for(j=front;j<=max-1;j++)

printf("%d",q[j]);

printf("->");

printf("\nrear is at %d\n",q[rear]);

printf("\nfront is at %d\n",q[front]);

}

else {

for(i=front;i<=rear;i++)

{

printf("%d",q[i]);

printf("->");

}

printf("END");

printf("\nrear is at %d\n",q[rear]);

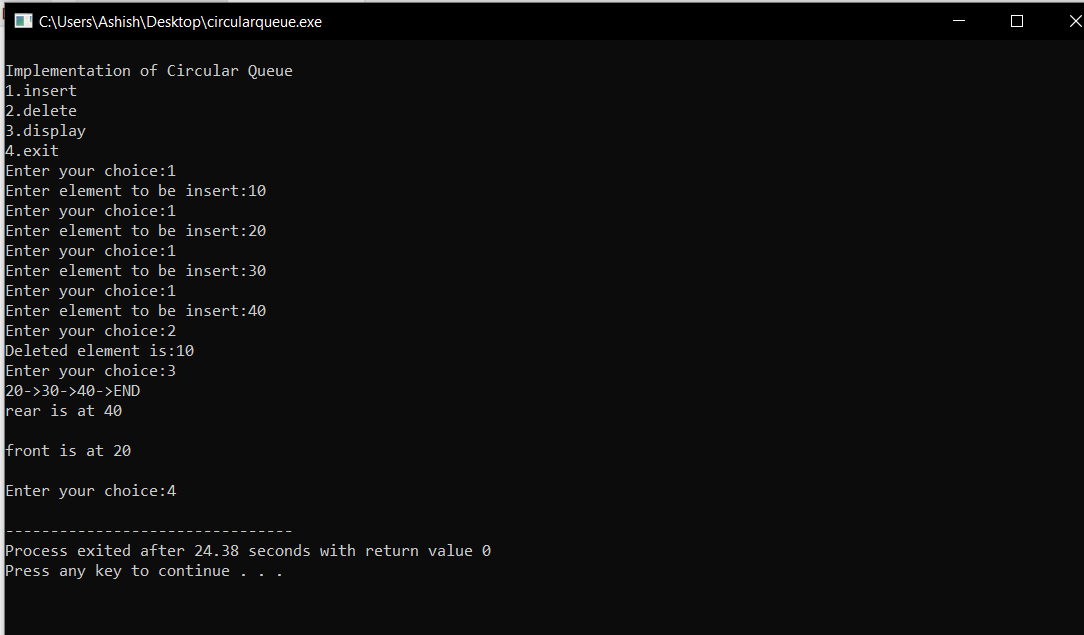
printf("\nfront is at %d\n",q[front]);

}

printf("\n");

}

**OUTPUT:-**



**Learning & Finding:-** Through C Programming, we can easily implement a circular Queue and can perform it’s various functions using Array.

**PROGRAM 7**

**Aim:-**To write a program to convert a valid prefix expression into postfix expression.

**Theory:-**

**Prefix expression** : An expression is called the prefix expression, if the operator appears in the expression before the operands.

Example : \*+AB-CD

**Postfix expression**: An expression is called the postfix expression if the operator appears in the expression after the operands.

Example : AB+CD-\*

**ALGORITHM:-**

Step 1 : Read the Prefix expression in reverse order (from right to left)

Step 2 : If the symbol is an operand, then push it onto the Stack

Step 3 : If the symbol is an operator, then pop two operands from the Stack  
Create a string by concatenating the two operands and the operator after them.  
**string = operand1 + operand2 + operator**  
And push the resultant string back to Stack

Step 4 : Repeat the above steps until end of Prefix expression

**CODE:-**

#include <iostream>

#include <stack>

using namespace std;

bool isOperator(char x) {

switch (x) {

case '+':

case '-':

case '/':

case '\*':

return true;

}

return false;

}

string preToPost(string pre\_exp) {

stack<string> s;

int length = pre\_exp.size();

for (int i = length - 1; i >= 0; i--) {

if (isOperator(pre\_exp[i])) {

string op1 = s.top(); s.pop();

string op2 = s.top(); s.pop();

string temp = op1 + op2 + pre\_exp[i];

s.push(temp);

}

else {

s.push(string(1, pre\_exp[i]));

}

}

return s.top();

}

int main()

{

string pre\_exp;

cout<<"Enter a valid prefix expression: ";

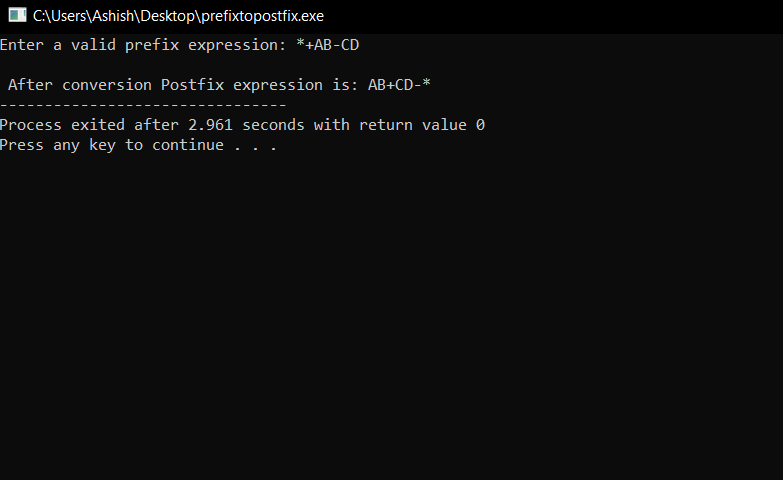
cin>>pre\_exp;

cout << "\n After conversion Postfix expression is: " << preToPost(pre\_exp);

return 0;

}

**OUTPUT:-**

****

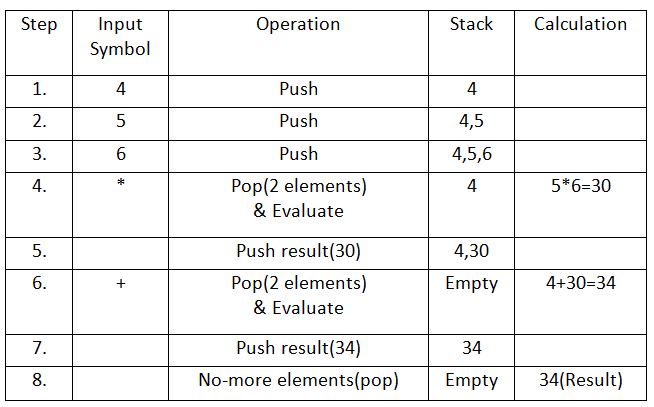
**Learning & Finding:-** Through C++ Programming, we can easily convert a valid prefix expression to postfix expression using Stack.

**PROGRAM 8**

**Aim:-**To write a program to evaluate a given postfix expression and its values for the variables.

**Theory:-**

**Given Expression: 456\*+**



**Evaluation rule of a Postfix Expression states:**

1. While reading the expression from left to right, push the element in the stack if it is an operand.
2. Pop the two operands from the stack, if the element is an operator and then evaluate it.
3. Push back the result of the evaluation. Repeat it till the end of the expression.

**ALGORITHM:-**

1. Create a stack to store operands (or values).  
   2) Scan the given expression and do following for every scanned element.  
   …..a) If the element is a number, push it into the stack  
   …..b) If the element is a operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack  
   3) When the expression is ended, the number in the stack is the final answer

**CODE:-**

#define SIZE 50 /\* Size of Stack \*/

#include <ctype.h>

#include<iostream>

using namespace std;

int s[SIZE];

int top=-1;

void push(int elem)

{

s[++top]=elem;

}

int pop()

{

return(s[top--]);

}

int main()

{

char pofx[50],ch;

int i=0,op1,op2;

cout<<"Enter a valid Postfix Expression : ";

cin>>pofx;

while( (ch=pofx[i++]) != '\0')

{

if(isdigit(ch)) push(ch-'0');

else

{

op2=pop();

op1=pop();

switch(ch)

{

case '+':push(op1+op2);break;

case '-':push(op1-op2);break;

case '\*':push(op1\*op2);break;

case '/':push(op1/op2);break;

}

}

}

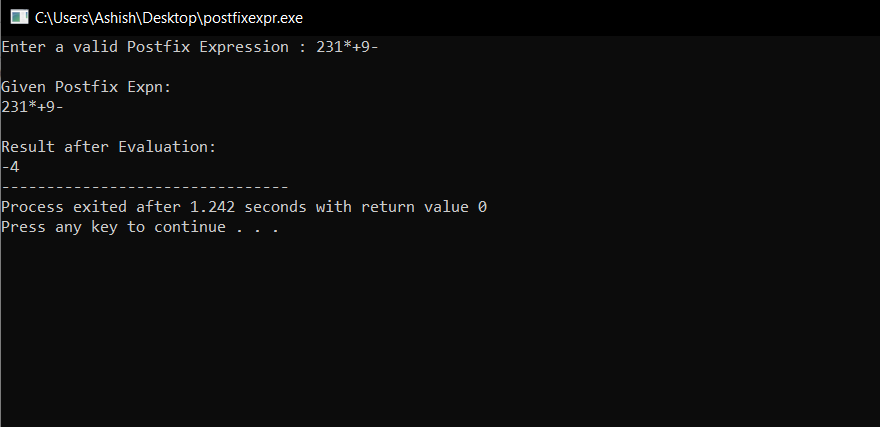
cout<<"\nGiven Postfix Expn: \n"<<pofx<<endl;

cout<<" \nResult after Evaluation: \n"<<s[top];

return 0;

}

**OUTPUT:-**

****

**Learning & Finding:-** Through C++ Programming, we can evaluate valid postfix expression using Stack.

**PROGRAM 9**

**Aim:-**To make a program on Quick sort using Divide and

conquer.

**Theory:-** Like [Merge Sort](http://quiz.geeksforgeeks.org/merge-sort/), QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

**ALGORITHM:-**

**Partition Algorithm**  
There can be many ways to do partition. The logic is simple, we start from the leftmost element and keep track of index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

/\* low --> Starting index, high --> Ending index \*/

quickSort(arr[], low, high)

{

if (low < high)

{

/\* pi is partitioning index, arr[pi] is now

at right place \*/

pi = partition(arr, low, high);

quickSort(arr, low, pi - 1); // Before pi

quickSort(arr, pi + 1, high); // After pi

}

}

**Pseudo code for partition()**

/\* This function takes last element as pivot, places

the pivot element at its correct position in sorted

array, and places all smaller (smaller than pivot)

to left of pivot and all greater elements to right

of pivot \*/

partition (arr[], low, high)

{

// pivot (Element to be placed at right position)

pivot = arr[high];

i = (low - 1) // Index of smaller element

for (j = low; j <= high- 1; j++)

{

// If current element is smaller than the pivot

if (arr[j] < pivot)

{

i++; // increment index of smaller element

swap arr[i] and arr[j]

}

}

swap arr[i + 1] and arr[high])

return (i + 1)

}

**CODE:-**

#include <iostream>

using namespace std;

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition (int arr[], int low, int high)

{

int pivot = arr[high]; // pivot

int i = (low - 1);

for (int j = low; j <= high- 1; j++)

{

if (arr[j] <= pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

int pivot = partition(arr, low, high);

quickSort(arr, low, pivot - 1);

quickSort(arr, pivot + 1, high);

}

}

void displayArray(int arr[], int size)

{

int i;

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

cout<<arr[i]<<"\t";

}

int main()

{

int size ;

cout<<"Enter size of array : ";

cin>>size;

int arr[size] ;

cout<<"\nEnter Input array elements : "<<endl;

for (int i=0; i < size; i++) {

cin>>arr[i];

}

cout<<endl;

quickSort(arr, 0, size-1);

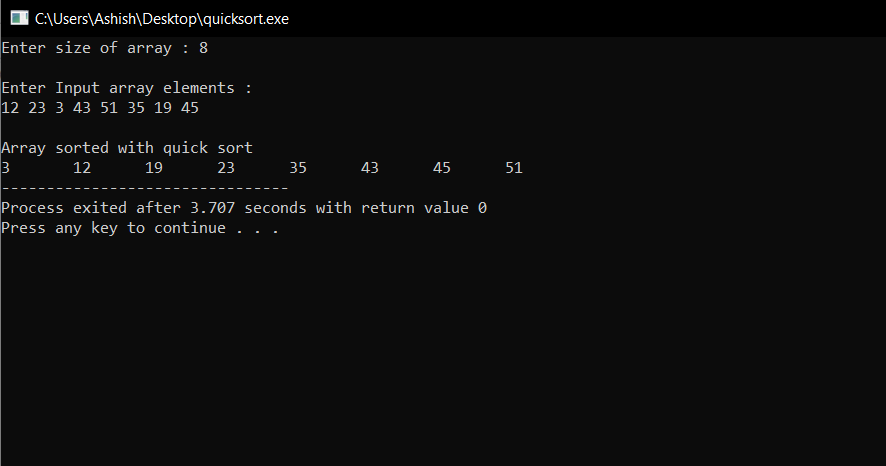
cout<<"Array sorted with quick sort"<<endl;

displayArray(arr,size);

return 0;

}

**OUTPUT:-**



**Learning & Finding:-** Through Quick sort ,we can sort the elements of an array in n\*log(n) time.

**PROGRAM 10**

**Aim:-**To sort an array of numbers using Heap Sort.

**Theory:-** Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

A [Binary Heap](https://www.geeksforgeeks.org/binary-heap/) is a Complete Binary Tree where items are stored in a special order such that value in a parent node is greater(or smaller) than the values in its two children nodes. The former is called as max heap and the latter is called min heap. The heap can be represented by binary tree or array.

**ALGORITHM:-**

Since a Binary Heap is a Complete Binary Tree, it can be easily represented as array and array based representation is space efficient. If the parent node is stored at index I, the left child can be calculated by 2 \* I + 1 and right child by 2 \* I + 2 (assuming the indexing starts at 0).

**Heap Sort Algorithm for sorting in increasing order:**  
**1.** Build a max heap from the input data.  
**2.** At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.  
**3.** Repeat above steps while size of heap is greater than 1.

**CODE:-**

#include <iostream>

using namespace std;

void heapify(int arr[], int n, int i)

{

int largest = i; // Initialize largest as root

int l = 2 \* i + 1; // left = 2\*i + 1

int r = 2 \* i + 2; // right = 2\*i + 2

if (l < n && arr[l] > arr[largest])

largest = l;

if (r < n && arr[r] > arr[largest])

largest = r;

if (largest != i) {

swap(arr[i], arr[largest]);

heapify(arr, n, largest);

}

}

void heapSort(int arr[], int n)

{

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i = n - 1; i >= 0; i--) {

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

void printArray(int arr[], int n)

{

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

cout << arr[i] << " ";

cout << "\n";

}

int main()

{

int size ;

cout<<"Enter size of array : ";

cin>>size;

int arr[size] ;

cout<<"\nEnter Input array elements : "<<endl;

int i;

for (i=0; i < size; i++){

cin>>arr[i];

}

heapSort(arr, size);

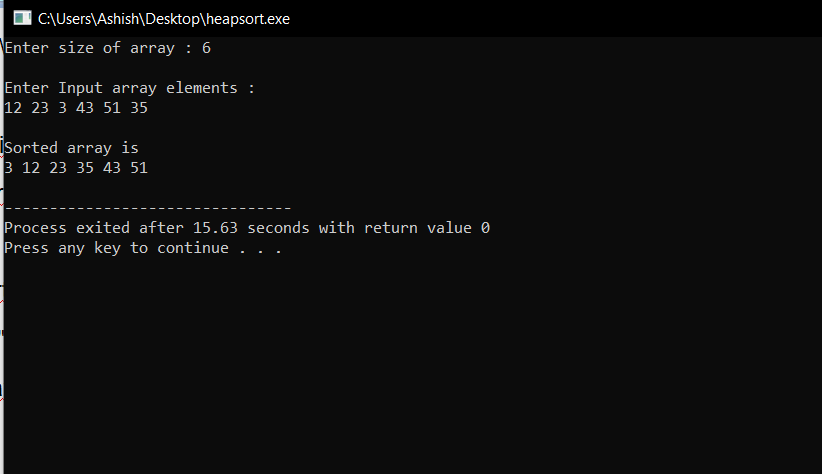
cout << "\nSorted array is \n";

printArray(arr, size);

return 0;

}

**OUTPUT:-**

****

**Learning & Finding:-** Through Heap sort , we can sort the elements of an array in n\*log(n) time.

**PROGRAM 11**

**Aim:-** Program to traverse all the nodes of a Binary Tree using the following traversals.

**Theory & ALGORITHM:-**

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree −

* In-order Traversal
* Pre-order Traversal
* Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

## In-order Traversal

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order.



We start from **A**, and following in-order traversal, we move to its left subtree **B**. **B** is also traversed in-order. The process goes on until all the nodes are visited. The output of inorder traversal of this tree will be −

***D → B → E → A → F → C → G***

### Algorithm

Until all nodes are traversed −

**Step 1** − Recursively traverse left subtree.

**Step 2** − Visit root node.

**Step 3** − Recursively traverse right subtree.

## Pre-order Traversal

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.



We start from **A**, and following pre-order traversal, we first visit **A** itself and then move to its left subtree **B**. **B** is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be −

***A → B → D → E → C → F → G***

### Algorithm

Until all nodes are traversed −

**Step 1** − Visit root node.

**Step 2** − Recursively traverse left subtree.

**Step 3** − Recursively traverse right subtree.

## Post-order Traversal

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.



We start from **A**, and following Post-order traversal, we first visit the left subtree **B**. **B** is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be −

***D → E → B → F → G → C → A***

### Algorithm

Until all nodes are traversed −

**Step 1** − Recursively traverse left subtree.

**Step 2** − Recursively traverse right subtree.

**Step 3** − Visit root node.

* A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

**CODE:-**

#include <iostream>

using namespace std;

struct Node

{

int data;

struct Node\* left, \*right;

Node(int data)

{

this->data = data;

left = right = NULL;

}

};

void printPostorder(struct Node\* node)

{

if (node == NULL)

return;

printPostorder(node->left);

printPostorder(node->right);

cout << node->data << " ";

}

void printInorder(struct Node\* node)

{

if (node == NULL)

return;

printInorder(node->left);

cout << node->data << " ";

printInorder(node->right);

}

void printPreorder(struct Node\* node)

{

if (node == NULL)

return;

cout << node->data << " ";

printPreorder(node->left);

printPreorder(node->right);

}

int main()

{

struct Node \*root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->left = new Node(6);

root->right->right = new Node(7);

root->right->left->left = new Node(8);

cout << "Preorder traversal of binary tree is :\n";

printPreorder(root);

cout << "\n\nInorder traversal of binary tree is :\n";

printInorder(root);

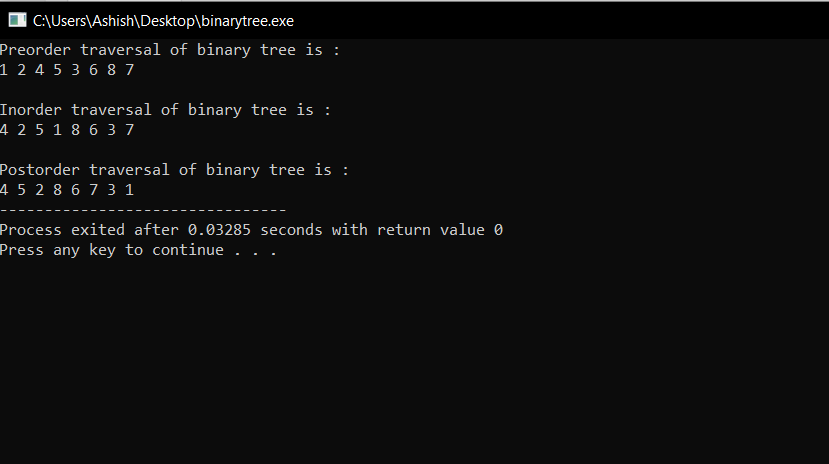
cout << "\n\nPostorder traversal of binary tree is :\n";

printPostorder(root);

return 0;

}

**OUTPUT:-**

****

**Learning & Finding:-** Through C++ Programming, we can traverse the binary tree in various order i.e., inorder,preorder,postorder in **Time Complexity of**O(n).