**STACKS AND QUEUES**

**LAB # 08**



**Data Structures & Algorithms**

Submitted by: **Shah Raza**

Registration No: **18PWCSE1658**

Class Section: **B**

“On my honor, as a student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to: **Dr. Khurram Shehzad Khattak**

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**Lab Objectives:**

Objectives of this lab are as follows:

* Implement basic operations on Stacks.
* Implement basic operations on Queues.

**Task # 1:**

Implement Stack using Arrays.

**Code:**

#include <iostream>

using namespace std;

typedef struct

{

int top=-1;

int \*Array;

}Stack;

int SIZE;

Stack Create(Stack S, int Size)

{

SIZE=Size;

S.Array = new int[SIZE];

return S;

}

Stack Pop(Stack S)

{

if(S.top<0)

{

cout<<"Stack is Empty.\n";

}

else

{

int R=S.Array[S.top];

S.top-=1;

cout<<"Element "<<R<<" is Popped from the Stack.\n";

}

return S;

}

Stack Push(int N,Stack S)

{

if(S.top>=SIZE-1)

{

cout<<"Stack is Full.\n";

}

else

{

S.top+=1;

S.Array[S.top]=N;

cout<<"Element Inserted.\n";

}

return S;

}

void Top(Stack S)

{

cout<<"Element on Top is: "<<S.Array[S.top]<<endl;

}

void Size(Stack S)

{

cout<<"The size of the Stack is: "<<S.top+1<<endl;

}

Stack Empty(Stack S)

{

delete[] S.Array;

S.top=-1;

cout<<"Stack has been Emptied.\n";

return S;

}

void Display(Stack S)

{

if(S.top>=0)

{

cout<<"Elements of the Stack are:\n";

for(int i=S.top;i>=0;i--)

cout<<S.Array[i]<<" ";

cout<<endl;

}

else

cout<<"Stack is Empty.\n";

}

int main()

{

int choice, item,S;

Stack s;

while (true)

{

cout<<"\n-------------"<<endl;

cout<<"Operations on Stack"<<endl;

cout<<"\n-------------"<<endl;

cout<<"1.Create Stack"<<endl;

cout<<"2.Push Element into the Stack"<<endl;

cout<<"3.Pop Element from the Stack"<<endl;

cout<<"4.Display the Stack"<<endl;

cout<<"5.Empty the Stack"<<endl;

cout<<"6.Size of Stack"<<endl;

cout<<"7.Top item"<<endl;

cout<<"8.Quit"<<endl;

cout<<"Enter your Choice: ";

cin>>choice;

switch(choice)

{

case 1:

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

cin>>S;

s = Create(s,S);

break;

case 2:

cout<<"Enter value to be pushed into the stack: ";

cin>>item;

s = Push(item,s);

break;

case 3:

s = Pop(s);

break;

case 4:

Display(s);

break;

case 5:

s=Empty(s);

break;

case 6:

Size(s);

break;

case 7:

Top(s);

break;

case 8:

return 0;

break;

default:

cout<<"Wrong Choice"<<endl;

}

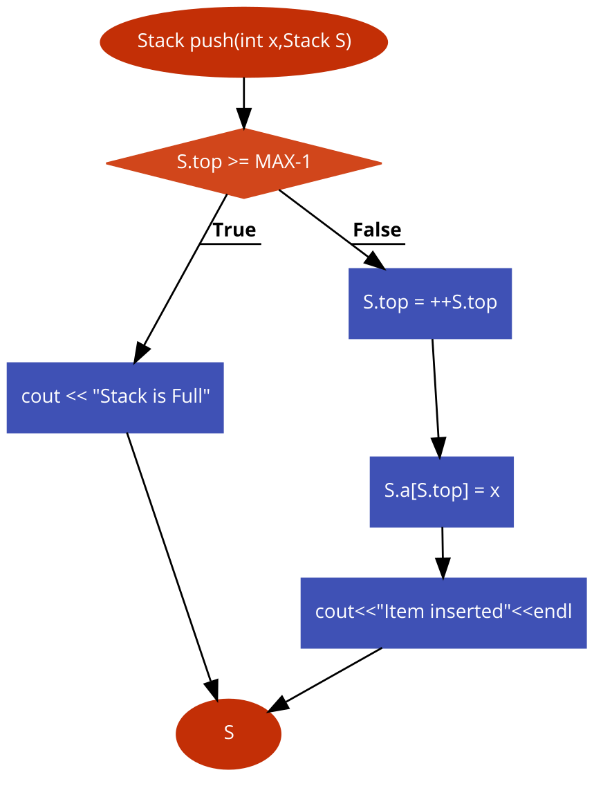
}

return 0;

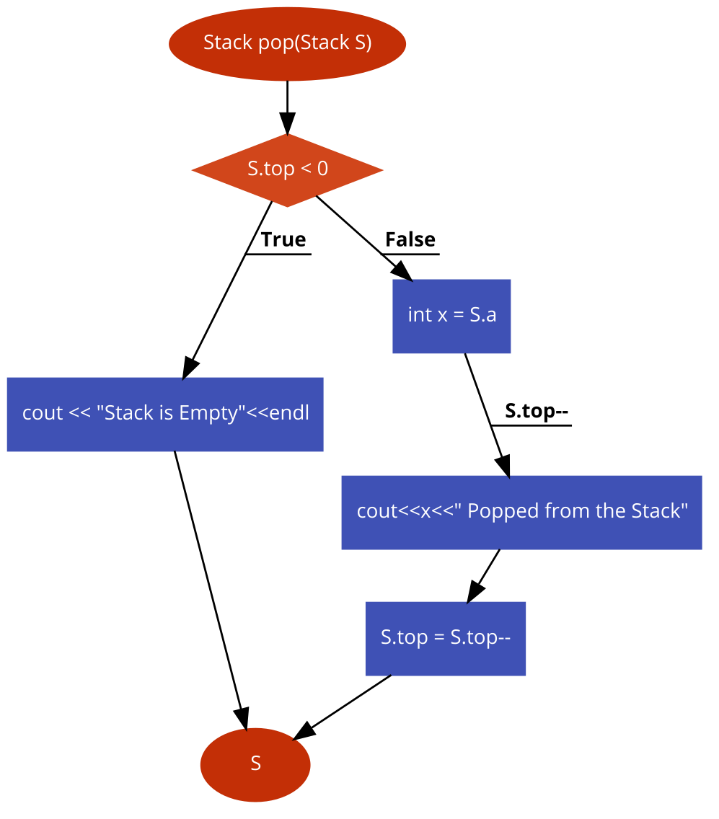
}

**Flow Chart:**

* **Push Function:**

****

* **Pop Function:**

****

**Pseudo-Code:**

* Push function:
  + if top is greater than or equal to MAX-1
    - Then print “Stack is Full”
  + else
    - Increment top
    - Insert the entered item, Array[top] = X
    - Print “Item inserted”
  + return the updated Stack
* Pop function:
  + if top is less than 0
    - Then print “Stack is Empty”
  + else
    - X = array[top]
    - Print “X Popped from the Stack”
    - Decrement top
* return the updated Stack

**Task # 2:**

Implement Stack using Linked List.

**Code:**

#include <iostream>

using namespace std;

typedef struct Node

{

int data;

Node \*next;

}Node;

class Stack

{

private:

Node \*top;

int MAX\_SIZE;

int Size=0;

public:

void Create(int Size)

{

MAX\_SIZE=Size;

top=NULL;

cout<<"Stack Created.\n";

}

void Push(int data)

{

if(Size>=MAX\_SIZE)

{

cout<<"Stack is Full.\n";

}

else

{

Node \*temp;

temp= new Node();

temp->data=data;

temp->next=top;

top=temp;

Size++;

cout<<"Element Pushed into the Stack.\n";

}

}

void Pop()

{

if(Size==0)

cout<<"Stack is Empty.\n";

else

{

Node \*temp;

temp=top;

top=top->next;

temp->next=NULL;

delete temp;

Size--;

cout<<"Element Popped from the stack.\n";

}

}

void Top()

{

if(Size==0)

cout<<"Stack is Empty.\n";

else

cout<<"Top Element is: "<<top->data<<endl;

}

void size()

{

cout<<"Size of the Stack is: "<<Size<<endl;

}

void Empty()

{

Node \*temp;

temp=top;

delete temp;

top=NULL;

Size=0;

cout<<"Stack Emptied.\n";

}

void Display()

{

if(Size==0)

cout<<"Stack is Empty.\n";

else

{

Node \*temp;

temp=top;

while(temp!=NULL)

{

cout<<temp->data<<" ";

temp=temp->next;

}

}

}

};

int main()

{

int choice, item,S;

Stack s;

while (true)

{

cout<<"\n-------------"<<endl;

cout<<"Operations on Stack"<<endl;

cout<<"\n-------------"<<endl;

cout<<"1.Create Stack"<<endl;

cout<<"2.Push Element into the Stack"<<endl;

cout<<"3.Pop Element from the Stack"<<endl;

cout<<"4.Display the Stack"<<endl;

cout<<"5.Empty the Stack"<<endl;

cout<<"6.Size of Stack"<<endl;

cout<<"7.Top item"<<endl;

cout<<"8.Quit"<<endl;

cout<<"Enter your Choice: ";

cin>>choice;

switch(choice)

{

case 1:

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

cin>>S;

s.Create(S);

break;

case 2:

cout<<"Enter value to be pushed into the stack: ";

cin>>item;

s.Push(item);

break;

case 3:

s.Pop();

break;

case 4:

s.Display();

break;

case 5:

s.Empty();

break;

case 6:

s.size();

break;

case 7:

s.Top();

break;

case 8:

return 0;

break;

default:

cout<<"Wrong Choice"<<endl;

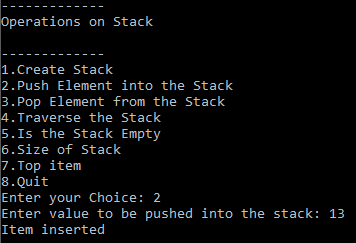
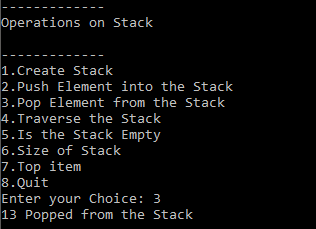
}

}

return 0;

}

**Output of Task 1 and 2 :**

** **

**Task # 3:**

Implement Queues using Arrays.

**Code:**

#include <iostream>

using namespace std;

typedef struct

{

int Front,Rear,Capacity;

int \*Array;

}Queue;

Queue Create(Queue Q,int S)

{

Q.Front=Q.Rear=0;

Q.Capacity=S;

Q.Array=new int;

return Q;

}

Queue Enqueue(int data, Queue Q)

{

if(Q.Rear==Q.Capacity)

{

cout<<"Queue is Full.\n";

}

else

{

Q.Array[Q.Rear]=data;

Q.Rear+=1;

cout<<"Element Enqueued.\n";

}

return Q;

}

Queue Dequeue(Queue Q)

{

if(Q.Front==Q.Rear)

cout<<"Queue is Empty.\n";

else

{

for(int i=0;i<Q.Rear-1;i++)

{

Q.Array[i]=Q.Array[i+1];

}

Q.Rear--;

cout<<"Element Dequeued.\n";

}

return Q;

}

void Front(Queue Q)

{

if(Q.Front==Q.Rear)

cout<<"Queue is Empty.\n";

else

cout<<"Front: "<<Q.Array[Q.Front]<<endl;

}

void Rear(Queue Q)

{

if(Q.Front==Q.Rear)

cout<<"Queue is Empty.\n";

else

cout<<"Rear: "<<Q.Array[Q.Rear-1]<<endl;

}

void Size(Queue Q)

{

if(Q.Front==Q.Rear)

cout<<"Queue is Empty.\n";

else

cout<<"Size of the Queue: "<<Q.Rear<<endl;

}

Queue Empty(Queue Q)

{

if(Q.Front==Q.Rear)

cout<<"Queue is Empty.\n";

else

{

Q.Front=Q.Rear=0;

delete[] Q.Array;

cout<<"Queue Emptied.\n";

}

return Q;

}

void Display(Queue Q)

{

if(Q.Front==Q.Rear)

cout<<"Queue is Empty.\n";

else

{

for(int i=Q.Rear-1;i>0;i--)

cout<<Q.Array[i]<<"-->";

cout<<Q.Array[0]<<endl;

}

}

int main()

{

int choice, item,S;

Queue q;

while (1)

{

cout<<"\n-------------"<<endl;

cout<<"Operations on Queue"<<endl;

cout<<"\n-------------"<<endl;

cout<<"1.Create Queue"<<endl;

cout<<"2.Enqueue"<<endl;

cout<<"3.Dequeue"<<endl;

cout<<"4.Empty the Queue"<<endl;

cout<<"5.Size of Queue"<<endl;

cout<<"6.Front"<<endl;

cout<<"7.Rear"<<endl;

cout<<"8.Display the Queue"<<endl;

cout<<"9.Quit"<<endl;

cout<<"Enter your Choice: ";

cin>>choice;

switch(choice)

{

case 1:

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

cin>>S;

q = Create(q,S);

break;

case 2:

cout<<"Enter the value to be Enqueued: ";

cin>>item;

q=Enqueue(item,q);

break;

case 3:

q=Dequeue(q);

break;

case 4:

q=Empty(q);

break;

case 5:

Size(q);

break;

case 6:

Front(q);

break;

case 7:

Rear(q);

break;

case 8:

Display(q);

break;

case 9:

return 0;

break;

default:

cout<<"Wrong Choice"<<endl;

}

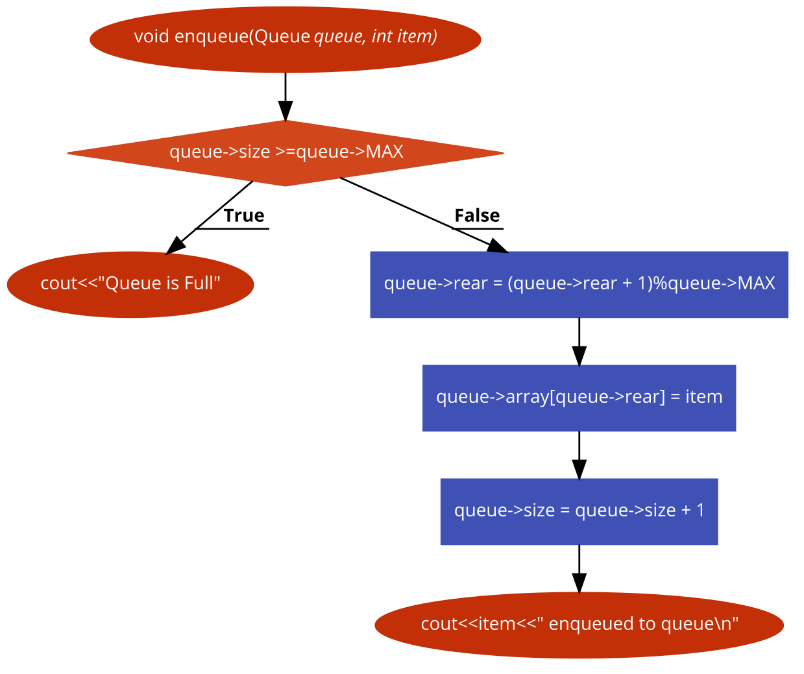
}

return 0;

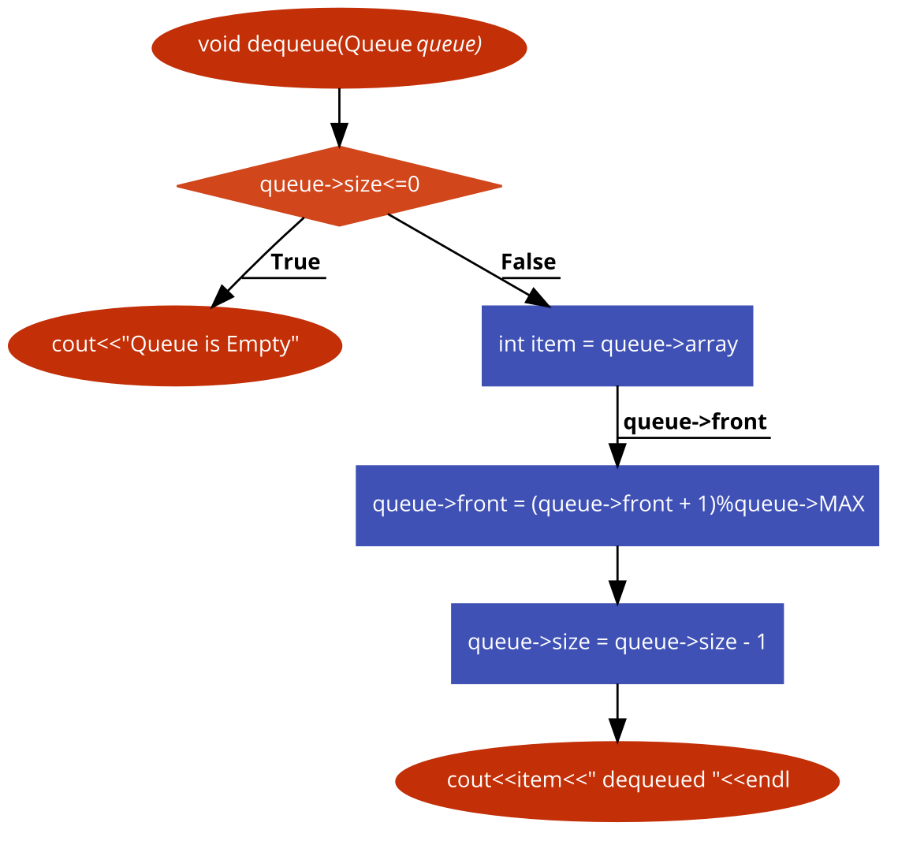
}

**Flow Chart:**

* **Enqueue Function:**

****

* **Dequeue Function:**

****

**Task # 4:**

Implement Queues using Linked List.

**Code:**

#include <iostream>

using namespace std;

typedef struct Node

{

int data;

Node \*next=NULL;

}Node;

class Queue

{

private:

Node \*Front;

Node \*Rear;

int Capacity;

int SIZE=0;

public:

void Create(int C)

{

Front=Rear=NULL;

Capacity=C;

cout<<"Queue Created.\n";

}

void Enqueue(int x)

{

if(SIZE>=Capacity)

cout<<"Queue is Full.\n";

else

{

Node \*temp;

temp= new Node;

temp->data=x;

if(Rear==NULL)

{

Front=Rear=temp;

}

else

{

Rear->next=temp;

Rear=temp;

}

cout<<"Element Enqueued.\n";

SIZE++;

}

}

void Dequeue()

{

if(Front==NULL)

cout<<"Queue is Empty.\n";

else

{

Node \*temp=Front;

Front=Front->next;

if(Front==NULL)

{

Rear=NULL;

}

delete temp;

SIZE--;

cout<<"Element Dequeued.\n";

}

}

void FRONT()

{

if(Front==NULL)

cout<<"Queue is Empty.\n";

else

cout<<"Front: "<<Front->data<<endl;

}

void REAR()

{

if(Rear==NULL)

cout<<"Queue is Empty.\n";

else

cout<<"Rear: "<<Rear->data<<endl;

}

void Size()

{

if(Front==NULL)

cout<<"Queue is Empty.\n";

else

cout<<"Size of the Queue: "<<SIZE<<endl;

}

void Empty()

{

if(Front==NULL)

cout<<"Queue is Empty.\n";

else

{

Node \*temp=Front;

Node \*N;

while(temp!=NULL)

{

N=temp;

temp=temp->next;

delete N;

}

Front=Rear=NULL;

cout<<"Queue Emptied.\n";

SIZE=0;

}

}

void Display()

{

if(Front==NULL)

cout<<"Queue is Empty.\n";

else

{

Node \*temp=Front;

int Array[SIZE];

int i=0;

while(temp!=NULL)

{

Array[i]=temp->data;

i++;

temp=temp->next;

}

for(int j=SIZE-1;j>0;j--)

cout<<Array[j]<<"-->";

cout<<Array[0]<<endl;

}

}

};

int main()

{

int choice, item,S;

Queue q;

while (1)

{

cout<<"\n-------------"<<endl;

cout<<"Operations on Queue"<<endl;

cout<<"\n-------------"<<endl;

cout<<"1.Create Queue"<<endl;

cout<<"2.Enqueue"<<endl;

cout<<"3.Dequeue"<<endl;

cout<<"4.Empty the Queue"<<endl;

cout<<"5.Size of Queue"<<endl;

cout<<"6.Front"<<endl;

cout<<"7.Rear"<<endl;

cout<<"8.Display the Queue"<<endl;

cout<<"9.Quit"<<endl;

cout<<"Enter your Choice: ";

cin>>choice;

switch(choice)

{

case 1:

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

cin>>S;

q.Create(S);

break;

case 2:

cout<<"Enter the value to be Enqueued: ";

cin>>item;

q.Enqueue(item);

break;

case 3:

q.Dequeue();

break;

case 4:

q.Empty();

break;

case 5:

q.Size();

break;

case 6:

q.FRONT();

break;

case 7:

q.REAR();

break;

case 8:

q.Display();

break;

case 9:

return 0;

break;

default:

cout<<"Wrong Choice"<<endl;

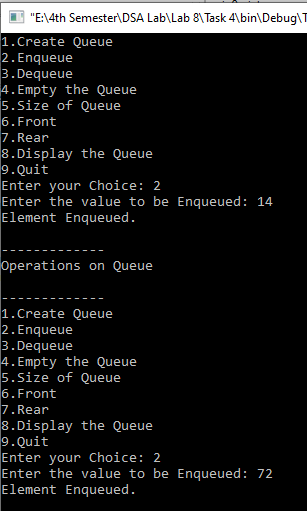
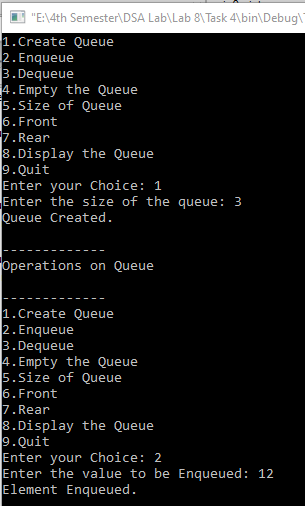
}

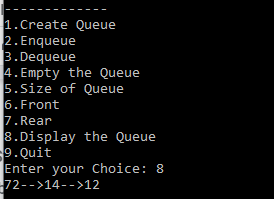
}

return 0;

}

**Output of Task 3 and 4:**

****

****