**Data Structures using C++**

**Module 2**

1. **Stack operations using array (push, pop, peek, display).**

#include <iostream>

using namespace std;

int stack[10], n, top=-1;

void display()

{

if(top>=0)

{

cout<<"Stack: ";

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

{

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

}

cout<<"\n";

}

else

{

cout<<"Stack is empty";

}

}

void push()

{

int val;

if(top>=n-1)

{

cout<<"Stack Overflow\n";

}

else

{

cout<<"Enter value to be pushed: ";

cin>>val;

top++;

stack[top]=val;

display();

}

}

void pop()

{

if(top<=-1)

{

cout<<"Stack Underflow\n";

}

else

{

cout<<"The popped element is "<< stack[top] <<"\n";

top--;

display();

}

}

int main()

{

int ch, val;

cout<<"Enter stack size: ";

cin>>n;

do

{

cout<<"\nMENU\n1. Push\n2. Pop\n3. Display\n4. Exit\n";

cout<<"Enter choice: ";

cin>>ch;

switch(ch)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

break;

default:

cout<<"Invalid Choice\n";

}

}while(ch!=4);

return 0;

}

1. **Infix to postfix conversion.**

#include<iostream>

#include<stack>

using namespace std;

bool isOperator(char c)

{

if(c=='+' || c=='-' ||c=='\*' || c=='/' || c=='^')

{

return true;

}

else

{

return false;

}

}

int precedence(char c)

{

if(c== '^')

return 3;

else if(c== '\*' || c== '/')

return 2;

else if(c == '+' || c== '-')

return 1;

else

return -1;

}

string infixtopostfix(stack<char> s, string infix)

{

string postfix;

for(int i=0;i<infix.length();i++)

{

if((infix[i]>='a' && infix[i]<='z') || (infix[i]>='A' && infix[i]<='Z'))

{

postfix+=infix[i];

}

else if(infix[i]=='(')

{

s.push(infix[i]);

}

else if(infix[i]==')')

{

while((s.top()!='(') && (!s.empty()))

{

postfix+=s.top();

s.pop();

}

s.pop();

}

else if(isOperator(infix[i]))

{

if(s.empty())

{

s.push(infix[i]);

}

else if(precedence(infix[i])>=precedence(s.top()))

{

s.push(infix[i]);

}

else if(precedence(infix[i])<precedence(s.top()))

{

postfix+=s.top();

s.pop();

s.push(infix[i]);

}

}

}

return postfix;

}

int main()

{

string infix\_exp,postfix\_exp;

stack <char> stack;

cout<<"Enter infix expression: ";

cin>>infix\_exp;

cout<<"Infix expression: "<<infix\_exp<<"\n";

postfix\_exp=infixtopostfix(stack,infix\_exp);

cout<<"Postfix expression: "<<postfix\_exp;

}

1. **Postfix evaluation.**

#include<iostream>

#include<stack>

#include<math.h>

using namespace std;

bool isOperator(char c)

{

if(c=='+' || c=='-' ||c=='\*' || c=='/' || c=='^')

{

return true;

}

else

{

return false;

}

}

int postfixeval(string postfix)

{

stack <int> s;

int num;

for(int i=0;i<postfix.length();i++)

{

if(isdigit(postfix[i]))

{

s.push(postfix[i]-'0');

}

else if(postfix[i]==' ')

{

continue;

}

else if(isOperator(postfix[i]))

{

int a=s.top();

s.pop();

int b=s.top();

s.pop();

switch(postfix[i])

{

case '+':

s.push(a+b);

break;

case '-':

s.push(b-a);

break;

case '\*':

s.push(a\*b);

break;

case '/':

s.push(b/a);

break;

case '^':

s.push(pow(a,b));

break;

}

}

}

return s.top();

}

int main()

{

string postfix\_exp;

cout<<"Enter postfix expression: ";

getline(cin,postfix\_exp);

cout<<"\nValue of expression: ";

cout<<postfixeval(postfix\_exp);

return 0;

}

1. **Queue operations using array (insertion, deletion, display).**

#include<iostream>

using namespace std;

class queue

{

int front;

int rear;

int q[10];

public:

queue()

{

front=-1;

rear=-1;

}

void enqueue(int n)

{

int item;

if(rear==n-1)

{

cout<<"Queue Full\n";

}

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

{

cout<<"Enter element: ";

cin>>item;

front=0;

rear=0;

q[rear]=item;

display();

}

else

{

cout<<"Enter element: ";

cin>>item;

rear++;

q[rear]=item;

display();

}

}

void dequeue()

{

if(front==-1 || front>rear)

{

cout<<"Queue Empty\n";

}

else

{

int num=q[front];

front++;

cout<<num<<" deleted\n";

display();

}

}

void display()

{

if(front==-1 || front>rear)

{

cout<<"Queue Empty\n";

}

else

{

cout<<"Queue: ";

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

{

cout<<q[i]<<" ";

}

cout<<"\n";

}

}

};

int main()

{

int n,c;

queue que;

cout<<"Enter queue size: ";

cin>>n;

do

{

cout<<"MENU\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n";

cout<<"Enter your choice: ";

cin>>c;

switch(c)

{

case 1:

que.enqueue(n);

break;

case 2:

que.dequeue();

break;

case 3:

que.display();

break;

case 4:

break;

default:

cout<<"Invalid choice\n";

}

}

while(c!=4);

return 0;

}

1. **Circular queue operations.**

#include <iostream>

using namespace std;

class queue

{

int front;

int rear;

int q[10];

public:

queue()

{

front = -1;

rear = -1;

}

void enqueue(int n)

{

int item;

if ((front == 0 && rear == n - 1) || (front == rear + 1))

{

cout << "Queue Full\n";

}

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

{

cout << "Enter element: ";

cin >> item;

front = 0;

rear = 0;

q[rear] = item;

display(n);

}

else

{

cout << "Enter element: ";

cin >> item;

rear = (rear + 1) % n;

q[rear] = item;

display(n);

}

}

void dequeue(int n)

{

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

{

cout << "Queue Empty\n";

}

else if (front == rear)

{

cout << q[front] << " deleted\n";

front = -1;

rear = -1;

}

else

{

cout << q[front] << " deleted\n";

front = (front + 1) % n;

}

display(n);

}

void display(int n)

{

int f = front, r = rear;

if (front == -1)

{

cout << "Queue Empty\n";

}

else if (front <= rear)

{

cout << "Queue: ";

while (f <= r)

{

cout << q[f] << " ";

f++;

}

}

else

{

cout << "Queue: ";

while (f < n)

{

cout << q[f] << " ";

f++;

}

f = 0;

while (f <= r)

{

cout << q[f] << " ";

f++;

}

}

cout << "\n";

}

};

int main()

{

int n,c;

queue que;

cout << "Enter queue size: ";

cin >> n;

do

{

cout << "MENU\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n";

cout << "Enter your choice: ";

cin >> c;

switch (c)

{

case 1:

que.enqueue(n);

break;

case 2:

que.dequeue(n);

break;

case 3:

que.display(n);

break;

case 4:

break;

default:

cout << "Invalid choice\n";

}

} while (c != 4);

return 0;

}

**Module – 3**

1. **Create a linked list which stores integer values and display it.**
2. **Implement singly linked list operations (insertion, deletion, display).**
3. **Sort a linked list.**
4. **Concatenate two linked lists.**
5. **Swap two data fields in a linked list.**
6. **Implement search in a linked list.**
7. **Create a student database using linked list (creation, display, search).**
8. **Implement circular linked list.**
9. **Implement basic operations of doubly linked list.**
10. **Implement circular doubly linked list.**
11. **Implement stack using linked list.**
12. **Implement queue using linked list.**