**Bahria University, Lahore Campus**

Department of Computer Science

Lab Journal 03

**(Spring 2023)**

|  |  |  |
| --- | --- | --- |
| Course: | **Data Structures and Algorithm - Lab** | Date: \_9-3-2023\_\_ |
| Course Code: | CSL-221 | Max Marks: 10 |
| Faculty’s Name: | Fatima Zulfiqar |  |

Name: \_AFFAN AHMAD\_ Enroll No: \_03-134221-003\_\_ Class: \_BS CS \_\_

Objective(s):

Upon completion of this lab session, learners will be able to:

* Implement stack data structure (push, pop, and display)
* Stack Application - Convert decimal to binary
* Stack Application - Convert infix expression to postfix expression
* Stack Application - Convert infix expression to prefix expression

## Lab Tasks:

**Task 1**

Implement stack data structure using arrays and perform following operations.

* Push ()
* Pop ()
* Display ()

**Note:** size of the array should be fixed. Don’t use dynamic array.

#include<iostream>

using namespace std;

int stack[100], size = 100,top=-1;

void push(int val)

{

if (top >= size - 1)

{

cout << "stack is overflow " << endl;

}

else

{

top++;

stack[top] = val;

}

}

void pop()

{

if (top <= -1)

{

cout << "stack underflow :" << endl;

}

else

{

cout << "your pop element is :" << stack[top];

top--;

}

}

void display()

{

cout << "your stack elements are :";

if (top > -1)

{

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

{

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

}

}

else

{

cout << "your stack is empty" << endl;

}

}

int main()

{

int ch, val;

cout << "1) Push in stack" << endl;

cout << "2) Pop from stack" << endl;

cout << "3) Display stack" << endl;

cout << "4) Exit" << endl;

do {

cout << "Enter choice: " << endl;

cin >> ch;

switch (ch) {

case 1: {

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

cin >> val;

push(val);

break;

}

case 2: {

pop();

cout << endl;

break;

}

case 3: {

display();

cout << endl;

break;

}

case 4: {

cout << "Exit" << endl;

break;

}

default: {

cout << "Invalid Choice" << endl;

}

}

} while (ch != 4);

system("pause");

return 0;

}

**Task 2**

Write a program to convert decimal number (taken from the user) to binary using stack data structure.

#include<iostream>

using namespace std;

int stack[100], size = 100,top=-1;

int num, rem;

void push(int val)

{

if (top >= size - 1)

{

cout << "stack is overflow " << endl;

}

else

{

top++;

stack[top] = val;

}

}

void pop()

{

if (top <= -1)

{

cout << "stack underflow :" << endl;

}

else

{

cout << "your pop element is :" << stack[top];

top--;

}

}

void display()

{

cout << "your stack elements are :";

if (top > -1)

{

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

{

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

}

}

else

{

cout << "your stack is empty" << endl;

}

}

int main()

{

int ch, val;

cout << "1) Push in stack" << endl;

cout << "2) Pop from stack" << endl;

cout << "3) Display stack" << endl;

cout << "4) convert decimal to binary " << endl;

cout << "4) Exit" << endl;

do {

cout << "Enter choice: " << endl;

cin >> ch;

switch (ch) {

case 1: {

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

cin >> val;

push(val);

break;

}

case 2: {

pop();

cout << endl;

break;

}

case 3: {

display();

cout << endl;

break;

}

case 4:{

cout << "enter decimal number :"; cin >> num;

while ((num != 0))

{

if (top>=size-1)

{

cout << "stack is overflow " << endl;

}

else

{

push(num % 2);

num = num / 2;

}

}

break;

}

case 5: {

cout << "Exit" << endl;

break;

}

default: {

cout << "Invalid Choice" << endl;

}

}

} while (ch != 5);

system("pause");

return 0;

}

**Task 3**

Write a program that takes infix expression from the user and convert it into postfix expression using stack.

**Example:**

**Infix Expression:** A^B\*C-D+E/F/(G+H)

**Postfix Expression:**  AB^C\*D-EF/GH+/+

**#include<iostream>**

**#include<stack>**

**using namespace std;**

**// defines the Boolean function for operator, operand, equalOrhigher precedence and the string conversion function.**

**bool IsOperator(char);**

**bool IsOperand(char);**

**bool eqlOrhigher(char, char);**

**string convert(string);**

**int s[100],size=100,top=-1;**

**void push(int val)**

**{**

**if(top>=size-1)**

**{**

**cout << "overflow "<< endl;**

**}**

**else**

**{**

**top++;**

**s[top]=val;**

**}**

**}**

**void pop()**

**{**

**if(top<=-1)**

**{**

**cout << "stack is underflow "<< endl;**

**}**

**else**

**{**

**cout << "your pop element is :"<< s[top]<< endl;**

**top--;**

**}**

**}**

**void display()**

**{**

**if (top>-1)**

**{**

**cout << "your stack elements are :";**

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

**{**

**cout << s[i]<<" ";**

**}**

**}**

**else**

**{**

**cout << "your stack is empty :"<< endl;**

**}**

**}**

**// define the IsOperator() function to validate whether any symbol is operator.**

**/\* If the symbol is operator, it returns true, otherwise false. \*/**

**bool IsOperator(char c)**

**{**

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

**return true;**

**return false;**

**}**

**// IsOperand() function is used to validate whether the character is operand.**

**bool IsOperand(char c)**

**{**

**if( c >= 'A' && c <= 'Z') /\* Define the character in between A to Z. If not, it returns False.\*/**

**return true;**

**if (c >= 'a' && c <= 'z') // Define the character in between a to z. If not, it returns False. \*/**

**return true;**

**if(c >= '0' && c <= '9') // Define the character in between 0 to 9. If not, it returns False. \*/**

**return true;**

**return false;**

**}**

**// here, precedence() function is used to define the precedence to the operator.**

**int precedence(char op)**

**{**

**if(op == '+' || op == '-') /\* it defines the lowest precedence \*/**

**return 1;**

**if (op == '\*' || op == '/')**

**return 2;**

**if(op == '^') /\* exponent operator has the highest precedence \*/**

**return 3;**

**return 0;**

**}**

**/\* The eqlOrhigher() function is used to check the higher or equal precedence of the two operators in infix expression. \*/**

**bool eqlOrhigher (char op1, char op2)**

**{**

**int p1 = precedence(op1);**

**int p2 = precedence(op2);**

**if (p1 == p2)**

**{**

**if (op1 == '^' )**

**{**

**return false;**

**}**

**else**

**{**

**return true;**

**}**

**}**

**return (p1>p2 ? true : false);**

**}**

**/\* string convert() function is used to convert the infix expression to the postfix expression of the Stack \*/**

**string convert(string infix)**

**{**

**stack <char> s;**

**string postfix ="";**

**char ch;**

**s.push( '(' );**

**infix += ')';**

**for(int i = 0; i<20; i++)**

**{**

**ch = infix[i];**

**if(ch == ' ')**

**continue;**

**else if(ch == '(')**

**push(ch);**

**else if(IsOperand(ch))**

**postfix += ch;**

**else if(IsOperator(ch))**

**{**

**while(!s.empty() && eqlOrhigher(s.top(), ch))**

**{**

**postfix += s.top();**

**s.pop();**

**}**

**s.push(ch);**

**}**

**else if(ch == ')')**

**{**

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

**{**

**postfix += s.top();**

**s.pop();**

**}**

**s.pop();**

**}**

**}**

**return postfix;**

**}**

**int main ()**

**{**

**string infix\_expression, postfix\_expression;**

**int ch, val;**

**cout<<"1) Push in stack"<<endl;**

**cout<<"2) Pop from stack"<<endl;**

**cout<<"3) Display stack"<<endl;**

**cout<<"4) infix to postfix"<< endl;**

**cout<<"5) Exit"<<endl;**

**do {**

**cout<<"Enter choice: "<<endl;**

**cin>>ch;**

**switch(ch) {**

**case 1: {**

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

**cin>>val;**

**push(val);**

**break;**

**}**

**case 2: {**

**pop();**

**break;**

**}**

**case 3: {**

**display();**

**cout << endl;**

**break;**

**}**

**case 4:**

**{**

**do**

**{**

**cout << " Enter an infix expression: ";**

**cin >> infix\_expression;**

**postfix\_expression = convert(infix\_expression);**

**cout << " Your Infix expression is: " << infix\_expression<< endl;**

**cout << " Postfix expression is: " << postfix\_expression<< endl;**

**} while(ch == 1);**

**}**

**case 5: {**

**cout<<"Exit"<<endl;**

**break;**

**}**

**default: {**

**cout<<"Invalid Choice"<<endl;**

**}**

**}**

**}while(ch!=5);**

**return 0;**

**}**

**Task 4**

Write a program that takes infix expression from the user and convert it into prefix expression using stack.

**Example:**

**Infix Expression:** A^B\*C-D+E/F/(G+H)

**Prefix Expression:**  -\*^ABC+D/E/F+GH

**Hint:**

Reverse given infix expression, remaining logic as it is with the difference of brackets only. Reverse expression in prefix string again before displaying. Strrev()

**Lab Grading Sheet :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Max Marks** | **Obtained Marks** | **Comments(*if any*)** |
| 1. | 02 |  |  |
| 2. | 02 |  |  |
| 3. | 03 |  |  |
| 4. | 03 |  |  |
| **Total** | **10** |  | **Signature** |

**Note : Attempt all tasks and get them checked by your Lab Instructor.**