**Лабораторна робота № 4**

**Прохоренко Ярослав ФБ-95**

**«Array Queue»**

#include <iostream>

#include <string>

#include <limits>

using namespace std;

template<typename T>

class ArrayQueue

{

private:

int size;

int head;

int tail;

int count;

T\* arr;

public:

ArrayQueue(int size\_);

bool enQueue(T value);

T deQueue();

void display();

bool isEmpty()

{

return count == 0;

}

};

template<typename T>

ArrayQueue<T>::ArrayQueue(int size\_)

{

size = size\_;

head = tail = -1;

count = 0;

arr = new T[size\_];

}

template<typename T>

bool ArrayQueue<T>::enQueue(T value)

{

try {

if ((head == 0 && tail == size - 1) || head == tail + 1)

{

throw -1;

}

else if (head == -1)

{

head = 0;

tail = 0;

arr[tail] = value;

count++;

}

else if (tail == size - 1 && head != 0)

{

tail = 0;

arr[tail] = value;

count++;

}

else

{

tail++;

arr[tail] = value;

count++;

}

return 1;

}

catch (...)

{

return 0;

}

}

template<typename T>

T ArrayQueue<T>::deQueue()

{

T val = arr[head];

if (head == tail)

{

head = -1;

tail = -1;

}

else if (head == size - 1)

head = 0;

else

head++;

count--;

return val;

}

template<typename T>

void ArrayQueue<T>::display()

{

if (head == -1)

{

cout << "[ERROR] Queue is empty\n";

return;

}

cout << "Queue: ";

if (tail >= head)

{

for (int i = head; i <= tail; i++)

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

cout << "\n";

}

else

{

for (int i = head; i < size; i++)

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

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

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

}

}

template<typename T>

void choice(ArrayQueue<T>\* arrQ)

{

int a = NULL;

while (a != 4)

{

system("clear");

cout << "---- Lab4 ----\n\nChoose operation: \n| 1. EnQueue\n| 2. DeQueue\n| 3. Display\n| 4. Exit\n\nlab4@asd:~$ ";

cin >> a;

while (cin.fail())

{

cin.clear();

cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

cout << "[ERROR] Incorrect input! Try again...\n\nlab4@asd:~$ ";

cin >> a;

}

switch (a)

{

case 1:

{

cout << "\nEnter value: ";

T b;

cin >> b;

arrQ->enQueue(b) ? cout << "Value '" << b << "' successfully added!\n" : cout << "Failed!\n";

cin.get();

} break;

case 2:

{

if (arrQ->isEmpty())

{

cout << "[ERROR] Queue is empty!\n";

}

else {

T temp = arrQ->deQueue();

cout << "Value '" << temp << "' successfully deleted!\n";

}

cin.get();

}break;

case 3:

arrQ->display();

cin.get();

break;

case 4:

break;

default:

cout << "[ERROR] Incorrect operation! Try again...\n";

cin.get();

break;

}

cout << "Press any key...";

cin.get();

}

}

int main()

{

ArrayQueue<int> aq(5);

choice(&aq);

return 0;

}

**«List Queue»**

#include <iostream>

#include <limits>

using namespace std;

struct Node

{

int data;

Node\* prev;

Node\* next;

};

class ListQueue

{

private:

int size;

Node\* head;

Node\* tail;

public:

ListQueue();

bool isEmpty()

{

return size == 0;

}

void enQueue(int data\_);

int deQueue();

void display();

};

ListQueue::ListQueue()

{

size = 0;

head = tail = NULL;

}

void ListQueue::enQueue(int data\_)

{

Node\* tempNode = new Node;

tempNode->next = NULL;

tempNode->data = data\_;

if(head != NULL)

{

tempNode->prev = tail;

tail->next = tempNode;

tail = tempNode;

size++;

}else

{

tempNode->prev = NULL;

head = tail = tempNode;

size++;

}

}

int ListQueue::deQueue()

{

int value = 0;

value = head->data;

if(size > 0)

{

if(size == 1)

{

delete head;

head = NULL;

size--;

}else

{

Node\* tempNode = head;

tempNode = head->next;

head->next->prev = NULL;

delete head;

head = tempNode;

size--;

}

}

return value;

}

void ListQueue::display()

{

Node\* tempNode = head;

while(tempNode != NULL)

{

cout << tempNode->data << " ";

tempNode = tempNode->next;

}

cout << "\n";

}

void choice(ListQueue\* listQ)

{

int a = 0;

while(a != 4)

{

system("clear");

cout << "---- Lab4 ----\n\nChoose operation: \n| 1. EnQueue\n| 2. DeQueue\n| 3. Display\n| 4. Exit\n\nlab4@asd:~$ ";

cin >> a;

while (cin.fail())

{

cin.clear();

cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

cout << "[ERROR] Incorrect input! Try again...\n\nlab4@asd:~$ ";

cin >> a;

}

switch(a)

{

case 1:

cout << "\nEnter value: ";

int val;

cin >> val;

listQ->enQueue(val);

cin.get();

break;

case 2:

if(listQ->isEmpty())

{

cout << "[ERROR] Queue is empty!\n";

}else

{

int temp = listQ->deQueue();

}

cin.get();

break;

case 3:

listQ->display();

cin.get();

break;

case 4:

break;

default:

cout << "[ERROR] Incorrect operation! Try again...\n";

cin.get();

break;

}

cout << "Press any key...\n";

cin.get();

}

}

int main()

{

ListQueue lq;

choice(&lq);

return 0;

}

**«Postfix using stack»**

#include <iostream>

#include <string>

using namespace std;

struct Node

{

char ch;

Node\* next;

};

class Stack

{

private:

int size;

Node\* head;

public:

Stack();

void push(char ch\_);

char pop();

void display();

bool isEmpty()

{

return size == 0;

}

char top()

{

return head->ch;

}

};

Stack::Stack()

{

size = 0;

head = NULL;

}

void Stack::push(char ch\_)

{

Node\* tempNode = new Node;

tempNode->ch = ch\_;

if(head != NULL)

{

tempNode->next = head;

head = tempNode;

size++;

}else

{

tempNode->next = NULL;

head = tempNode;

size++;

}

}

char Stack::pop()

{

char ch\_;

if(head != NULL)

{

if(size > 1)

{

ch\_ = head->ch;

Node\* tempNode = head->next;

delete head;

head = tempNode;

size--;

}else

{

ch\_ = head->ch;

delete head;

head = NULL;

size--;

}

return ch\_;

}

return 0;

}

void Stack::display()

{

Node\* tempNode = head;

while(tempNode != NULL)

{

cout << tempNode->ch << endl;

tempNode = tempNode->next;

}

}

bool isOperand(char ch\_)

{

if(ch\_ >= 'A' && ch\_ <= 'Z' || ch\_ >= 'a' && ch\_ <= 'z') return true;

return false;

}

bool isOperator(char ch\_)

{

if(ch\_ == '+' || ch\_ == '-' || ch\_ == '/' || ch\_ == '\*') return true;

return false;

}

int getPriority(char ch\_)

{

if(ch\_ == '+' || ch\_ == '-') return 1;

if(ch\_ == '\*' || ch\_ == '/') return 2;

return -1;

}

string infxToPostfx(string infix)

{

string postfix = "";

Stack st;

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

{

if(isOperator(infix[i]))

{

while(!st.isEmpty() && getPriority(infix[i]) <= getPriority(st.top()))

{

postfix += st.pop();

}

st.push(infix[i]);

}

else if(isOperand(infix[i]))

{

postfix += infix[i];

}

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

{

st.push(infix[i]);

}

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

{

while(!st.isEmpty() && st.top() != '(')

{

postfix += st.pop();

}

st.pop();

}

}

while(!st.isEmpty())

{

postfix += st.pop();

}

st.display();

return postfix;

}

int main()

{

cout << infxToPostfx("(A-B-C)/D-E\*F") << endl;

cout << infxToPostfx("(A+B)\*C-(D+E)/F") << endl;

cout << infxToPostfx("A/(B-C)+D\*(E-F)") << endl;

cout << infxToPostfx("(A\*B+C)/D-F/E") << endl;

return 0;

}