**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Линейные структуры данных: стек, очередь, дек.

Вариант 8

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Карамышев А.О. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

**Цель работы.**

Ознакомиться с линейной структурой – стеком, способами его реализации. Освоить на практике и закрепить навыки использования стека для решения практических задач.

**Постановка задачи.**

8. В заданном текстовом файле F записано логическое выражение (ЛВ) в следующей форме:

<ЛВ>::= true | false | (Ø< ЛВ > ) | ( < ЛВ > Ù < ЛВ > ) | ( < ЛВ > Ú < ЛВ > )

где знаки Ø, Ù и Ú обозначают соответственно отрицание, конъюнкцию и дизъюнкцию. Вычислить (как Boolean) значение этого выражения.

**Основные теоретические положения.**

Стек (от англ. stack – «стопка») – абстрактный тип данных, представляющий собой разновидность линейного списка, организованного по принципу *LIFO* (англ. *last in – first out,* «последним пришел – первым вышел»).

Существует 2 способа реализации стека:

1. В виде однонаправленного списка.

2. В виде вектора- одномерного массива с упорядоченными адресами.

В данной работе стек реализован с помощью линейного однонаправленного списка.

Функциональная спецификация стека задается следующими определениями (множество непустых стеков обозначено как *Non*\_*null*\_*stack*):

1) *Create*: → *Stack* (α);

2) *Null*: *Stack* (α) → *Boolean*;

3) *Top*: *Non*\_*null*\_*stack* (α) → α;

4) *Pop*: *Non*\_*null*\_*stack* (α) → *Stack* (α);

5) *Push*: α ⊗ *Stack* (α) → *Stack* (α).

и набором аксиом (∀*p*: α; ∀*s*: *Stack* (α); ∀*t*: *Non*\_*null*\_*stack* (α)):

A1) *Null* (*Create*) = *true*;

A2) *Null* (*Push* (*p*, *s*)) = *false*;

A3) *Top* (*Push* (*p*, *s*)) = *p*;

A4) *Pop* (*Push* (*p*, *s*)) = *s*;

A5) *Push* (*Top* (*s*), *Pop* (*s*)) = *s*.

*Ссылочная реализация* стека и очереди в динамической памяти в основном аналогична ссылочной реализации линейных списков. Упрощение здесь связано с отсутствием необходимости работать с текущим («внутренним») элементом списка.

*Стек*:

*Верхушка стека*

*Начало*:

*Конец*:

*Очередь*:

**Спецификация программы.**

Данная программа позволяет вычислить значение полученного на входе логического выражения. Программа считывает входные данные из файла.

При считывании из файла осуществляется проверка на корректность открытия данного файла, если происходит ошибка, пользователь будет уведомлен о ней.

При запуске программы, она мгновенно считает входные данные из файла, затем при помощи функции string norm(const string str) логическое выражение приводится к более простому виду, для дальнейших операций с ним.

Для подсчета введённого логического выражения следует прописать алгоритм превращения инфиксной формы записи выражения, в префиксную или постфиксную, в моем случае постфиксную.

Реализовал этот алгоритм с помощью двух функций string postfix(string &str) и int oper(char a). Первая функция и есть реализация алгоритма превращения инфиксной формы в постфиксную, вторая функция нужна для того, чтобы определять приоритет логических операций, когда происходит push из стека.

Требуемая задача вычисления значения логического выражения осуществляется при помощи одного стека. В функции bool solut(const string str) представлена реализация алгоритма, она состоит из следующих действий:

1. Символы из нашей постфиксной записи заносим в стек, до тех пор, пока не наткнемся на символ какого-либо логической операции.

|  |
| --- |
| class Stack {  private:  struct node;  node \*topOfStack;  public:  Stack()  {  topOfStack = NULL;  };  char top(void);  void pop(void);  char pop2(void);  void push(const char &x);  bool isNull(void);  void destroy(void);  }; |

1. При считывании символа логической операции, производится pop из стека в зависимости того, сколько наших переменных будут учувствовать в логической операции (для отрицания одна, для остальных две) и производится вычисление, результат push в стек, затем переходим обратно в п.1 до тех пор, пока не закончатся символы из входного потока.

Стек на языке программирования C++ реализован с помощью класса. Интерфейс отделен от реализации: интерфейс расположен в файле stek\_inter.h, а реализация соответственно в stek.cpp. Интерфейс выглядит следующим образом:

Рассмотрим функции класса Stack:

1. char top(void);

Функция возвращает значение головного элемента стека.

1. void pop(void);

Функция осуществляет удаление элемента из стека.

При удалении элемента убирается первый, а головным становится последующий элемент. При этом значение убранного элемента не возвращается.

1. char pop2(void);

Аналогично функции pop, данная функция удаляет последний элемент стека, но при этом возвращается значение удаляемого элемента, т.е. совмещаются последовательно вызванные функции top и pop.

1. void push(const char &x);

Функция осуществляет добавление нового элемента в стек (проталкивание в стек). Новый элемент становится головным и будет указывать на тот, который был «головой» до него.

1. bool isNull(void);

Функция проверяет, является ли стек пустым: если да, то возвращает 1, в противном случае – 0.

1. void destroy(void);

Функция освобождает динамическую память, выделенную под стек.

**Тестирование.**

|  |  |
| --- | --- |
| Логическое выражение | Результат |
| ((!(false))\*(true)) | true |
| ((!(true))\*(true)) | false |
| ((!<true))\*(true))+((false)+!(false)) | true |
| !true | false |
| !false | true |

**Выводы.**

Таким образом, в ходе выполнения лабораторной работы были получены и закреплены теоретические знания о стеке, а также приобретены навыки реализации стека при помощи линейного однонаправленного списка, и навыки написания функций для работы со стеком на языке программирования C++.

**Приложение А. Исходный код**

**Sourse.cpp**

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <windows.h>

#include <string>

#include "stek\_inter.h"

using namespace std;

using namespace st\_modul;

string norm(const string str) {

string str1;

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

if (str[i] == '<') str1 += '(';

else if (str[i] == '>') str1 += ')';

else if (str[i] == 'f') {

str1 += '0';

i += 4;

}

else if (str[i] == 't') {

str1 += '1';

i += 3;

}

else str1 += str[i];

}

return str1;

}

int oper(char a) {

if (a == '(') return 0;

if (a == '+') return 1;

if (a == '\*') return 2;

if (a == '!') return 3;

return 4;

}

string postfix(string &str) {

Stack s;

string str1;

str = norm(str);

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

if (str[i] == '0' || str[i] == '1') str1 += str[i];

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

s.push(str[i]);

else if (str[i] == ')') {

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

str1 += s.pop2();

s.pop();

}

else {

while ((oper(str[i]) <= oper(s.top())) && !s.isNull()) str1 += s.pop2();

s.push(str[i]);

}

}

while(!s.isNull()) { str1 += s.pop2(); };

s.destroy();

return str1;

}

bool solut(const string str) {

Stack s;

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

if (str[i] == '0' || str[i] == '1')

s.push(str[i] - 48);

if (str[i] == '!')

s.push(!s.pop2());

if (str[i] == '\*')

s.push(s.pop2() & s.pop2());

if (str[i] == '+')

s.push(s.pop2() | s.pop2());

}

if (s.pop2() == 0) {

s.destroy();

return false;

}

else {

s.destroy();

return true;

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

string str;

ifstream fin("postfix.txt");

if (!fin) { cout << "File not open for reading!\n"; return 1; }

getline(fin, str, '\n');

cout <<"Заданное выражение: "<< str<< endl;

str = postfix(str);

if (solut(str)) cout << "Ответ: true";

else cout << "Ответ: false";

system("pause > nul");

return 0;

}

Stek\_inter.h

#pragma once

namespace st\_modul

{

typedef int base;

class Stack {

private:

struct node;

node \*top\_stak;

public:

Stack(node \*inptop = nullptr) : top\_stak(inptop) {};

char top(void);

void pop(void);

char pop2(void);

void push(const char &x);

bool isNull(void);

void destroy(void);

};

}

Stek.cpp

#include <iostream>

#include <cstdlib>

#include "stek\_inter.h"

using namespace std;

namespace st\_modul

{

struct Stack::node {

char \*hd;

node \*tl;

node(char \*inphd = nullptr, node\* inptl = nullptr) :hd(inphd), tl(inptl) {};

};

char Stack::top(void){

if (top\_stak == nullptr) { return 0; }

else return \*top\_stak->hd;

}

void Stack::pop(void){

if (top\_stak == nullptr) { cerr << "Error: pop(null) \n"; exit(1); }

else{

node \*oldTop = top\_stak;

top\_stak = top\_stak->tl;

delete oldTop->hd;

delete oldTop;

}

}

char Stack::pop2(void){

if (top\_stak == nullptr) { cerr << "Error: pop(null) \n"; exit(1); }

else{

node \*oldTop = top\_stak;

char r = \*top\_stak->hd;

top\_stak = top\_stak->tl;

delete oldTop->hd;

delete oldTop;

return r;

}

}

void Stack::push(const char &x)

{

node \*p;

p = top\_stak;

top\_stak = new node;

if (top\_stak != nullptr) {

top\_stak->hd = new char;

\*top\_stak->hd = x;

top\_stak->tl = p;

}

else { cerr << "Memory not enough\n"; exit(1); }

}

bool Stack::isNull(void){

return (top\_stak == nullptr);

}

void Stack::destroy(void){

while (top\_stak != nullptr) {

pop();

}

}

}