**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №3**

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

**Тема: Стек**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7383 |  | Кирсанов А.Я. |
| Преподаватель |  | Размочева Н.В. |

Санкт-Петербург

2018

**Содержание**

[Цель работы 3](#__RefHeading__2318_1303384392)

[Реализация задачи](#__RefHeading__2320_1303384392) 3

[Тестирование](#__RefHeading__2322_1303384392) 4

[Вывод](#__RefHeading__2324_1303384392) 5

[Приложение A. Тестовые случаи](#__RefHeading__4504_1303384392) 6

[Приложение Б. Исходный код программы](#__RefHeading__4506_1303384392) 7

Цель работы

Цель работы: научиться реализовывать стек, освоить базовые функции работы с ним, научиться писать свои функции на основе рекурсии для работы со стеком.

Формулировка задачи: в заданном текстовом файле *F* записано логическое выражение (ЛВ) в следующей форме:

<ЛВ>::= *true* | *false* | (¬< ЛВ > ) | ( < ЛВ > ∧ < ЛВ > ) | ( < ЛВ > ∨ < ЛВ > )

где знаки ¬, ∧ и ∨ обозначают соответственно отрицание, конъюнкцию и дизъюнкцию. Вычислить (как *Boolean*) значение этого выражения.   
(Вариант 8-д).

Реализация задачи

Программа состоит головной функции, функций для работы со стеком и функций обработки выражения. В

В головной функции пользователю предлагается выбрать способ ввода выражения: из файла или с клавиатуры. После ввода выражения производится его обработка, затем выводится результат подсчета выражения и пользователю снова предлагается выбрать способ ввода нового выражения.

Обработка выражения производится с помощью стека. Стек хранит указатели на значения типа Boolean. Для стека написаны функция взятия верхнего значения, функция, позволяющая записать указатель на Boolean наверх стека, а также функция проверки пустоты стека и функция удаления стека.

Для обработки введенного выражения сначала вызывается функция Space, которая добавляет пробелы до и после круглых скобок в введенной строке, а также между знаками ‘+’ и ‘\*’. Вместо знаков ¬, ∧, ∨, для удобства ввода были использованы знаки ‘!’ , ‘+’ и ‘\*’ соответственно. После вызова функции Space вызывается функция expression, которая получает строку из функции readStr, проверяет, не является ли она знаком отрицания ‘!’ (знак отрицания должен быть в скобках). Если он присутствует, функция возвращает в main исключение " '!' must be in brackets.". Затем функция expression вызывает функцию read, которая кладет в стек значение 1, если первое слово в выражение – “true”, 0, если “false” или значение в скобках (для этого вызывается функция Bracket). После этого функция expression возвращает значение функции Mark.

Функция Mark производит действия между двумя соседними выражениями если между ними стоит знак ‘\*’. Для этого она вызывает функцию readOne, которая возвращает значение соседнего выражения (0, 1, или значение в скобках). Полученное при умножении значение записывается в стек и оно же будет использоваться в дальнейших операциях, так как приоритет умножения выше. Если между выражениями стоит знак ‘+’, функция main откладывает операцию сложения и в качестве правого аргумента вызывает функцию expression.

Подсчет значения в скобках производится с помощью функции Bracket. В скобках может быть либо отрицание выражения в скобках, либо само выражение. В обоих случаях Bracket вызывает readOne, которая, в свою очередь, в первом случае вызывает функцию Not (тем самым она возвращает отрицание Bracket). Во втором случае функция Bracket проверяет соответствие выражения модели < ЛВ > <операция> < ЛВ > . Если выражение в скобках не соответствует модели, функция передаст соответствующее исключение в main.

Конец выражения (при его соответствии модели) может быть обработан либо в функции read, либо в функции Mark. В обоих случаях функции возвращают в main значение Boolean. Все остальные функции передают функции main исключения, которые обрабатываются функцией класса ExpExpression – what(), которая выводит переданную строку в качестве ошибки.

Тестирование

Сборка и тестирование программы производилось в среде разработки QT на Linux Ubuntu 16.04 LTS.

В ходе тестирование были использованы различные выражения, заведомо правильные или неправильные. Результаты тестирования представлены в приложении А.

В ходе тестирования была обнаружена следующая ошибка: если выражение заканчивалось на функции read, она возвращала в main лишь верхнее значение стека (даже если функция read была вызвана функцией mark, производящую операцию между двумя логическими выражениями). Для исправления этой ошибки, обработка конца выражения была добавлена также в функцию mark.

Вывод

В ходе работы был изучен стек. Получен опыт работы с базовыми функциями стека. Реализована рекурсивная функция, вычисляющая значение логического выражения. Найдены и исправлены ошибки в работе функций.

ПРИЛОЖЕНИЕ А.

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Тестовые случаи

|  |  |
| --- | --- |
| Входные данные | Вывод программы |
| 1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program.  2  Enter expression:  (!(false\*(true+(true+false))))  ( ! ( false \* ( true + ( true + false ) ) ) ) | Answer: 1 |
| 1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program.  2  Enter expression:  false\*(true+(true+false))  false \* ( true + ( true + false ) ) | Answer: 0 |
| 1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program.  2  Enter expression:  false\*(true+(true+false)  false \* ( true + ( true + false ) | Missing brackets. |
| 1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program.  2  Enter expression:  asd  asd | Wrong logical. |
| 1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program.  1  Enter file name:  test.txt  true + false | Answer: 1 |
| 1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program.  1  Enter file name:  asd | Input file not open. |

**ПРИЛОЖЕНИЕ Б.**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**main.cpp:**

#include "st\_interf1.h"

#include "error.h"

#define N 1000

using namespace std;

using namespace st\_modul1;

int **main**()

{

char str0[N];

int k = 0;

bool b;

while(k != 3){

string str;

strstream x;

Stack s;

cout << endl << "1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program." << endl;

cin >> k;

switch (k) {

case 1:{

cout << "Enter file name:" << endl;

cin >> str;

ifstream outfile(str);

try {

if(!outfile) throw ExpException("Input file not open.\n");

} catch (ExpException &e) {

cout << e.*what*();

continue;

}

outfile.read(str0, N);

outfile.close();

str = Space(str0);

x << str;

break;

}

case 2:{

cout << "Enter expression: " << endl;

cin.get();

cin.getline(str0, N);

str = Space(str0);

x << str;

break;

}

case 3:{ cout << "Press Enter\n"; return 0; }

}

try {

b = expression(s, *x*, str);

}

catch(bool a){

b = a;

}

catch (ExpException &e){

cout << endl;

cout << e.*what*();

continue;

}

cout << endl << "Answer: " << b << endl;

}

return 0;

}

**st\_impl1.cpp**

#include "st\_interf1.h"

#include "error.h"

using namespace std ;

namespace st\_modul1

{

struct Stack::node {

base \*hd;

node \*tl;

node (){

hd = nullptr;

tl = nullptr;

}

};

base Stack::pop(void)

{

node \*oldTop = topOfStack;

base r = \*topOfStack->hd;

topOfStack = topOfStack->tl;

delete oldTop->hd;

delete oldTop;

return r;

}

void Stack::push (const base &x)

{ node \*p;

p = topOfStack;

topOfStack = new node;

topOfStack->hd = new base;

\*topOfStack->hd = x;

topOfStack->tl = p;

}

bool Stack::isNull(void)

{ return (topOfStack == nullptr) ;

}

void Stack::destroy (void)

{ while ( topOfStack != nullptr) {

pop();

}

}

}

**st\_interf1.h**

#include <iostream>

#include <strstream>

#include <cstdlib>

#include <fstream>

namespace st\_modul1{

typedef bool base;

class Stack{

private:

struct node;

node \*topOfStack;

public:

Stack ()

{ topOfStack = nullptr; }

base pop (void);

void push (const base &x);

bool isNull(void);

void destroy (void);

};

}

using namespace st\_modul1;

using namespace std;

bool expression(Stack s, strstream &x, string str);

bool Bracket(Stack s, strstream &x, string str);

bool readOne(Stack s, strstream &x, string str);

bool Mark(Stack s, strstream &x, string str);

bool oneMark(Stack s, strstream &x, string str);

bool Not(Stack s, strstream &x, string str);

string Space(char str0[]);

Stack read(Stack s, strstream &x, string str);

string readStr(Stack s, strstream &x, string str);

**error.h**

#include <exception>

#include <string>

class ExpException : std::exception

{

public:

ExpException(std::string &&whatStr) noexcept : whatStr(std::move(whatStr)) { }

ExpException(const std::string &whatStr) noexcept : whatStr(whatStr) { }

~ExpException() noexcept = default;

const char\* what() const noexcept override;

private:

std::string whatStr;

};

const char\* what();

**func.cpp**

#include "st\_interf1.h"

#include "error.h"

const char\* ExpException::*what*() const noexcept{

return whatStr.c\_str();

}

Stack read(Stack s, strstream &x, string str){

if(str == "(") s.push(Bracket(s, *x*, str));

else

if(str == "true") s.push(1);

else

if(str == "false") s.push(0);

else throw ExpException("Wrong logical.");

return s;

}

string readStr(Stack s, strstream &x, string str){

if(!(x >> str)){

if(!(s.isNull())){

bool a = s.pop();

s.destroy();

throw a;

}

else throw ExpException("Stack is null.");

}

cout << str << " ";

return str;

}

bool expression(Stack s, strstream &x, string str){

str = readStr(s, *x* ,str);

if(str == "!") throw ExpException("'!' must be in brackets.");

s = read(s, *x*, str);

return Mark(s, *x*, str);

}

bool readOne(Stack s, strstream &x, string str){

str = readStr(s, *x*, str);

if(str == "!") return Not(s, *x*, str);

s = read(s, *x*, str);

return s.pop();

}

bool Not(Stack s, strstream &x, string str){

str = readStr(s, *x*, str);

s = read(s, *x*, str);

return !(s.pop());

}

bool Mark(Stack s, strstream &x, string str){

if(!(x >> str)){

if(!(s.isNull())){

bool a = s.pop();

s.destroy();

return a;

}

else throw ExpException("Stack is null.");

}

cout << str << " ";

if(str == "+"){

s.push(s.pop() + expression(s, *x*, str));

}

if(str == "\*"){

s.push(s.pop() \* readOne(s, *x*, str));

return oneMark(s, *x*, str);

}

return expression(s, *x*, str);

}

bool oneMark(Stack s, strstream &x, string str){

if(!(x >> str)) return s.pop();

cout << str << " ";

if(str == "+"){

s.push(s.pop() + readOne(s, *x*, str));

}

if(str == "\*"){

s.push(s.pop() \* readOne(s, *x*, str));

}

return readOne(s, *x*, str);

}

bool Bracket(Stack s, strstream &x, string str){

bool b = readOne(s, *x*, str);

if(!(x >> str)){

throw ExpException("Empty mark");

}

cout << str << " ";

if(str == ")") return b;

if(str == "+") s.push(b + readOne(s, *x*, str));

else

if(str == "\*") s.push(b \* readOne(s, *x*, str));

else throw ExpException("Wrong mark.");

if(!(x >> str)) throw ExpException("Missing brackets.");

cout << str << " ";

if(str == ")") return s.pop();

else throw ExpException("Missing brackets.");

}

string Space(char str0[]){

int i = 0, k = 0;

string str1;

char str[1000];

while(str0[i] != '\0'){

if(str0[i] == '(' || str0[i] == ')' || str0[i] == '+' || str0[i] == '\*' || str0[i] == '!'|| str0[i] == '\0'){

str[k] = ' ';

str[k+1] = str0[i];

str[k+2] = ' ';

k += 3;

i++;

}

else{

if(str0[i] == '\n') break;

str[k] = str0[i];

i++;

k++;

}

}

str[k] = '\0';

str1 = str;

return str1;

}