**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 7

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Спас А.А. |
| 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.**

В заданном текстовом файле *F* записана формула вида

< формула > ::= < цифра > | *M* ( < формула >, < формула > ) |

*m* ( < формула >, < формула > )

< цифра > ::= 0 | 1 | … | 9

где *M* обозначает функцию *max*, a *m –* функцию *min*. Вычислить (как целое число) значение данной формулы. Например, *M* (5, *m*(6, 8)) = 6.

**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 полученной на входе формулы. Программа считывает входные данные из файла, либо с консоли. В начале работы программы пользователю предлагается сделать выбор, откуда именно считывать информацию.

Также существует проверка на неверно введенную команду, таким образом, в случае ошибки будет выведен соответствующий текст. Кроме этого, в случае считывания из файла осуществляется проверка на корректность открытия данного файла, если происходит ошибка, пользователь будет уведомлен о ней.

После успешного осуществления пользователем выбора, откуда следует производить считывание данных, путем передачи управления функции bool read\_expr(istream& input, Stack \*op, Stack \*dig);

Это рекурсивная функция, которая осуществляет последовательное считывание символов, входящих в исходную формулу.

На вход по ссылке подается поток ввода, а так же указатели на стеки.

Данная функция работает как синтаксический анализатор: вместе со считыванием происходит и анализ, является ли введенная формула корректной, т.е. соответствует ли она рекурсивному определению

< формула > ::= < цифра > | *M* ( < формула >, < формула > ) |

*m* ( < формула >, < формула > )

< цифра > ::= 0 | 1 | … | 9

Для отслеживания корректности используется булева переменная flag, принимающая значение 1, если текущий символ вписывается в определение, и 0 – в противном случае. При возникновении ошибок выводятся сообщения о них и их характере.

Требуемая задача вычисления значения формулы осуществляется при помощи двух стеков: первый (operation) используется для хранения того, какая функция должна быть вычислена (максимум M или минимум m), а второй (digit) – для хранения значений, которые используются в вычислениях.

При считывании символа происходит следующее:

1. Проверяется, является ли он цифрой. Если да, то при помощи функции push значение элемента добавляется во 2 стек – digit.
2. Или проверяется, является ли считанный символ названием функции: M или m. Если является, то таким же образом при помощи push в первый стек - operation отправляется 1 или 2 ()
3. Если же символ не подходит по пунктам 1 и 2, то он никуда не сохраняется, и программа переходит к следующему символу, контролируя при этом корректность введенной формулы.

Как только составная часть формулы идентифицирована как корректная, вызывается функция void calc(Stack\* op, Stack\* dig), осуществляющая частичное вычисление по текущей формуле.

Вычисления происходят таким образом:

1. Из стека digit берется значение головного элемента, а затем он удаляется (Используется функция pop2). Затем то же самое повторяется. Таким образом, мы имеем 2 последних необходимых значения.

2. Из стека operation берется головной элемент, который затем удаляется (Используется функция pop2).

3. По полученным значениям, с учетом того, какая функция взята из стека operation осуществляется частичное вычисление.

1. Полученный частичный результат с помощью функции push добавляется в стек digit.

Затем управление возвращается функции read\_expr.

Когда работу завершает функция read\_expr, необходимо удостовериться в том, что формула корректна. Если это действительно так, то выводится итоговый результат, в противном случае – сообщение о возникшей ошибке.

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

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

Рассмотрим функции класса Stack:

1. base 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. base 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 base &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.**

Приведем для одного из тестов более подробные результаты:

На вход подается формула: M( 5, m( 1 , 4 ) )

В файл с выходными данными и протоколом работы программы было выведено:

|  |
| --- |
| Calculation process.  Command:  1  Enter the expression:  M(5,m(1,4))  m( 4 , 1 ) = 1  ------------------------------------  M( 1 , 5 ) = 5  ------------------------------------  formula is correct  Result of calculating: 5  End! |

Тест успешен.

Таблица 1 - Тестирование

|  |  |
| --- | --- |
| Формула | Результат |
| 1 | 1 |
| M(1,5) | 5 |
| m(1,5) | 1 |
| M(2, 2) | 2, the arguments are equal |
| M(m(3,4), M(2,8)) | 8 |
| M(1) | Error |
| m 2,3) | Error |

**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**

Файл interf.h

#include <iostream>

#include <fstream>

#include "interface.h"

using namespace std;

using namespace stack;

void menu();

//считывание формулы

bool read\_expr(istream& input, Stack \*op, Stack \*dig);

//вычисление по формуле

void calc(Stack\* op, Stack\* dig);

ifstream fin("input.txt", ios\_base::in);

ofstream fout("output.txt", ios\_base::out);

//вспомогательная функция - меню

void menu()

{

cout << "----------------MENU----------------\n";

cout << "1. Read expression from the keyboard\n";

cout << "2. Upload expression from file\n";

cout << "------------------------------------\n";

}

//чтение выражения с клавиатуры

bool read\_expr(istream& input, Stack \*op, Stack \*dig)

{

bool flag = false;

char ch;

do input >> ch; while (ch == ' ');

if ((ch >= '0') && (ch <= '9')) //является ли цифрой?

{ //является цифрой

dig->push(ch - '0'); //записываем в стек digit

flag = true;

}

else if ((ch == 'M') || (ch == 'm')) //является ли обозначением операции min или Max?

{ //является обозначением операции

if (ch == 'M') //является MAX

{

op->push(1); // 1 эквивалентно M, записываем в стек operation

}

else if (ch == 'm') //является min

{

op->push(2); //2 эквивалентно m, записываем в стек operation

}

do input >> ch; while (ch == ' ');

if (ch == '(')

{//очередной символ - окрывающая скобка

flag = read\_expr(input, op, dig); //рекурсивный вызов

if (flag)

{//предыдущее выражение - формула

do input >> ch; while (ch == ' ');

if (ch == ',')

{//очередной символ - запятая

flag = read\_expr(input, op, dig);//рекурсивный вызов

if (flag)

{//предыдущее выражение - формула

do input >> ch; while (ch == ' ');

if (ch == ')')

{//очередной символ - закрывающая скобка

flag = true;

calc(op, dig); //вычисление по крайней считанной формуле

}

else

{

cout << "! - Error\n";

fout << "! - Error\n";

exit(EXIT\_FAILURE);

}

}

else

{

cout << "\n! - Error\n";

fout << "\n! - Error\n";

system("pause");

exit(EXIT\_FAILURE);

}

}

else

{

cout << "\n! - Error\n";

fout << "\n! - Error\n";

system("pause");

exit(EXIT\_FAILURE);

}

}

else

{

cout << "\n! - Error\n";

fout << "\n! - Error\n";

system("pause");

exit(EXIT\_FAILURE);

}

}

else

{

cout << "\n! - Error\n";

fout << "\n! - Error\n";

system("pause");

exit(EXIT\_FAILURE);

}

}

else

{

cout << "\n! - Error\n";

fout << "\n! - Error\n";

system("pause");

exit(EXIT\_FAILURE);

}

return flag; //для идентификации, является ли считываемое выражение формулой

}

void calc(Stack\* op, Stack\* dig)

{//вычисление последней считанной формулы

int fun = op->pop2(); //берем из стека operation последнюю

base a = dig->pop2(); //берем из стека символ и меняем указатель верхушки стека

base b = dig->pop2(); //берем из стека символ и меняем указатель верхушки стека

base res;//для хранения результата вычисления

if (a == b) //аргументы равны

{

cout << "the arguments are equal\n";

fout << "the arguments are equal\n";

res = a; //результат - любой из аргументов

}

else

{

if (fun == 1)//определяем максимум

{

(a > b) ? res = a : res = b; //если a>b, то резуьтат - a, иначе - b

cout << "M( " << a << " , " << b << " ) = " << res << endl;

fout << "M( " << a << " , " << b << " ) = " << res << endl;

}

else if (fun == 2)//определяем минимум

{

(a < b) ? res = a : res = b; //если a<b, то результат - a, иначе - b

cout << "m( " << a << " , " << b << " ) = " << res << endl;

fout << "m( " << a << " , " << b << " ) = " << res << endl;

}

}

dig->push(res); //записываем в стек результат вычисления

cout << "------------------------------------\n\n";

fout << "------------------------------------\n\n";

}

Файл interface.h

namespace stack

{

//-------------------------------------

typedef int base;

class Stack {

private:

struct node;

node \*topOfStack;

public:

Stack()

{

topOfStack = NULL;

};

base top(void);

void pop(void);

base pop2(void);

void push(const base &x);

bool isNull(void);

void destroy(void);

};

}

Файл implementation.cpp

#include <fstream>

#include <iostream>

#include <cstdlib>

#include "interface.h"

using namespace std;

namespace stack

{

struct Stack::node {//

base \*hd;

node \*tl;

// constructor

node()

{

hd = NULL; tl = NULL;

}

};// end node

//-------------------------------------

base Stack::top(void)

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: top(null) \n"; exit(1); }

else return \*topOfStack->hd;

}

//-------------------------------------

void Stack::pop(void)

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }

else

{

node \*oldTop = topOfStack;

topOfStack = topOfStack->tl;

delete oldTop->hd;

delete oldTop;

}

}

//-------------------------------------

base Stack::pop2(void)

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }

else

{

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;

if (topOfStack != NULL) {

topOfStack->hd = new base;

\*topOfStack->hd = x;

// cout << "push -> " << x << endl; // Demo

topOfStack->tl = p;

}

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

}

//-------------------------------------

bool Stack::isNull(void)

{

return (topOfStack == NULL);

}

//-------------------------------------

void Stack::destroy(void)

{

while (topOfStack != NULL) {

pop();

}

}

} // end of namespace stack

Файл main1.cpp

#include <iostream>

#include <fstream>

#include "interf.h"

using namespace std;

using namespace stack;

int main()

{

setlocale(LC\_ALL, "");

if (!fout.is\_open()) //проверка на корректность открытия

{

cout << "! - Error: file was not opened\n"; //ошибка

system("pause");

exit(EXIT\_FAILURE);

}

if (!fin.is\_open()) //проверка корректности открытия

{

cout << "! - Error: file is not opened." << endl; //ошибка

system("pause");

exit(EXIT\_FAILURE);

}

int cmd;

Stack operation; //стек для операций M(max) или m(min) из формулы

Stack digit; //стек для цифр из формулы

bool tag;

cout << "Загрузка данных из файла" << endl;

tag = read\_expr(fin, &operation, &digit); //начинаем считывание из файла

if (tag && !fout.eof())

{//формула корректна

cout << "\nformula is correct\n";

fout << "\nformula is correct\n";

}

else

{ //ошибка: слишком много символов

cout << "\n! - Error: invalid expression- too much symbols!\n";

fout << "\n! - Error: invalid expression- too much symbols!\n";

exit(EXIT\_FAILURE);

}

base result = digit.top(); // получение результата из стека.

cout << "Result of calculating: " << result << endl;

fout << "Result of calculating: " << result << endl;

cout << "End!\n";

fout << "End!\n";

system("pause");

fin.close();

fout.close();

operation.destroy();

digit.destroy();

return 0;

}