Министерство образования и науки Российской Федерации

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**Разработка структуры данных стек и применение её для решения практической задачи**

**Выполнил**:студент группы 381606-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимакин Н.Е.

Подпись

**Проверил**: к.ф.-м.н., доц.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Баркалов К.А.

Подпись

Нижний Новгород 2017

**Введение**

Стек – это абстрактный тип данных, представляющий собой список элементов и организованный по принципу «последний пришёл – первым ушёл», то есть добавлять или удалять элементы можно только с одного конца списка. Стек не только часто встречается в повседневной жизни (стопка тарелок или книг), но ещё с помощью него решаются многие прикладные задачи, например в теории графов. Одна из таких задач и будет рассмотрена как пример использования этого типа данных.

**Постановка задачи**

Реализовать тип данных стек как шаблонный класс, написать к нему тесты, подтверждающие его работоспособность, и тестовую программу – калькулятор, в двух вариантах: в виде консольного приложения и с графическим интерфейсом.

**Описание структуры программы**

Сам стек будет реализован как класс TStack в одном .h-файле, и для калькулятора понадобится написать отдельный класс TCalculator, который будет проводить все нужные операции для вычисления результата и использовать для этого поля типа TStack.

Программа содержит в себе 3 проекта:

1. Stack-lab – реализация классов TStack и TCalculator, тестовая программа (консольное приложение)
   * stack.h – объявление и реализация класса TStack
   * TCalculator.h – объявление класса TCalculator
   * TCalculator.cpp – реализация класса TCalculator
   * main.cpp – реализация тестовой программы
2. Test – тесты для стека и калькулятора
   * gtest-all.cc – библиотека с реализацией тестов
   * TCalculator.cpp (добавлен для работы с TCalculator)
   * test\_calculator.cpp – тесты для класса TCalculator
   * test\_main.cpp – файл, запускающий все тесты
   * test\_stack.cpp – тесты для класса TStack
3. Visual calculator – тестовая программа (приложение с графическим интерфейсом)
   * MyForm.h – объявление и реализация формы для графического приложения
   * MyForm.cpp – файл для запуска приложения
   * TCalculator.cpp (добавлен для работы с TCalculator)

Т - шаблон

**Класс TStack**

Поля:

T\* arr - память для хранения элементов

int size - текущий размер

int maxsize - максимальный размер

Методы:

TStack(int \_maxsize = 10) – конструктор по умолчанию

TStack(const TStack& st) – конструктор копирования

~TStack() - деструктор

TStack& operator=(const TStack& st) – перегрузка оператора присваивания

int isfull() – проверка на полноту

int isempty() – проверка на пустоту

T top() – возвращение вершины стека

T pop() – удаление элемента

void push(const T& el) – добавление элемента

int getsize() – возвращение текущего размера

int getmaxsize() – возвращение максимального размера

int operator==(const TStack& st) const – перегрузка операции сравнения

int operator!=(const TStack& st) const – перегрузка операции не равно

void clear() – очистка стека

**Класс TCalculator**

Выражение, которое необходимо посчитать, будет поступать в программу в виде строки. Чтобы эту строку символов превратить в последовательность математических операций, а затем и в вычисление результата, воспользуемся постфиксной записью, то есть вместо привычной для нас инфиксной строки <операнд 1> <оператор> <операнд 2> будем использовать такую: <операнд 1> <операнд 2> <оператор>. Например, строка “2+4” превратится в “2 4+”, или “5-3\*6” – “5 3 6\*-“, или “(1.1+2.7/3)^2” – “1.1 2.7 3/+ 2 ^”. Это позволит нам реализовать калькулятор именно с помощью стека.

Поля:

string infix – инфиксная строка

string postfix – постфиксная строка

TStack<char> stc – стек для знаков

TStack<double> StD – стек для чисел

Методы:

TCalculator(string \_infix = "") – конструктор по умолчанию

void setinfix(string \_infix) – задание инфиксной строки

string getinfix() – получение инфиксной строки

string getpostfix() – получение постфиксной строки

int check() – проверка на корректность инфиксной строки

int priority(char sym) - определение приоритета символа

void topostfix() – перевод инфиксной строки в постфиксную

double calc() – подсчёт результата

**Описание алгоритмов**

**Класс TStack**

TStack(int \_maxsize = 10) {если \_maxsize <= 0, то выбросить исключение. Максимальному размеру присвоить \_maxsize, текущему размеру – 0. Выделить память объёмом максимального размера}

TStack(const TStack& st) {максимальному размеру присвоить максимальный размер st. Текущему размеру присвоить текущий размер st. Выделить память объёмом максимального размера. Каждому элементу присвоить значение соответствующего из st}

~TStack() {освободить память}

TStack& operator=(const TStack& st) {если this не указывает на st, то если максимальные размеры не равны, то максимальному размеру присвоить максимальный размер st, освободить память, выделить новую; текущему размеру присвоить текущий размер st, каждому элементу присвоить значение соответствующего из st. Вернуть \*this}

int isfull() {если текущий размер совпадает с максимальным, то вернуть 1, иначе – 0}

int isempty() {если текущий размер равен 0, то вернуть – 1, иначе – 0}

T top() {если стек пустой, то выбросить исключение. Вернуть элемент с номером (текущий размер – 1)}

T pop() {если стек пустой, то выбросить исключение. Уменьшить текущий размер на 1. Вернуть элемент с номером (текущий размер)}

void push(const T& el) { если стек полный, то выбросить исключение. Записать в элемент с номером (текущий размер) значение el. Увеличить размер на 1}

int getsize() {вернуть текущий размер}

int getmaxsize() {вернуть максимальный размер}

int operator==(const TStack& st) const {Если максимальные размеры не совпадают, вернуть 0. Если текущие размеры не совпадают, вернуть 0. Если хотя бы один элемент не совпадает, вернуть 0. Вернуть 1}

int operator!=(const TStack& st) const {вернуть отрицание от проверки на равенство \*this и st}

void clear() {присвоить текущему размеру 0}

**Класс TCalculator**

TCalculator(string \_infix = "") {вызвать конструкторы для знакового и числового стеков с параметром (\_infix + 1). Инфиксной строке присвоить \_infix}

void setinfix(string \_infix) {Инфиксной строке присвоить \_infix}

string getinfix() {Вернуть инфиксную строку}

string getpostfix() {Вернуть постфиксную строку}

int check() {Очистить стек. В цикле от 0 до длины инфиксной строки: если i-ый символ инфиксной строки ‘(‘, то добавить ‘(‘ в знаковый стек; если i-ый элемент ‘)‘, то, если стек пустой – вернуть 0, удалить элемент из стека. Если стек пустой, то вернуть 1, иначе – 0}

int priority(char sym) {если sym это ‘(’, вернуть 0. Если sym это ‘+’ или ‘-‘, вернуть 1. Если sym это ‘\*’ или ‘/’, вернуть 2. Если sym это ‘^’, вернуть 3. Иначе вернуть -1}

void topostfix() {

если инфиксная строка корректна{

постфиксу присвоить пустую строку

очистить знаковый стек

создать строку-буфер, такую же как инфикс только со скобками в начале и в конце

цикл от 0 до длины буфера{

если i-ый символ буфера ‘(’{

добавить ‘(’ в стек

}

если i-ый символ буфера – цифра или ‘.’{

прибавить i-ый символ к постфиксу

}

если i-ый символ буфера ‘)’{

создать временный символ, равный вершине стека (применить pop())

пока временный символ не равен ‘)’{

прибавить к постфиксу временный символ

извлечь элемент из стека во временный символ

}

}

если i-ый символ буфера ‘+’, ‘-’, ‘\*’, ‘/’ или ‘^’{

прибавить к постфиксу пробел

пока приоритет i-ого символа меньше или равен приоритету вершины стека{

прибавить к постфиксу вершину стека (применить pop())

}

}

}

}

иначе{

выбросить исключение

}

}

double calc() {

перевести инфиксную строку в постфиксную

очистить числовой стек

в цикле от 0 до длины постфикса{

если i-ый элемент постфикса ‘+’, ‘-’, ‘\*’, ‘/’ или ‘^’{

создать два числа – операнд1 и операнд2

операнду2 присвоить вершину стека (pop())

операнду1 присвоить вершину стека (pop())

если i-ый символ:

‘+’: добавить в стек сумму операндов

‘-’: добавить в стек разность операндов

‘\*’: добавить в стек произведение операндов

‘/’: добавить в стек частное операндов

‘^’: добавить в стек оп1 в степени оп2

}

если i-ый элемент постфикса – цифра или ‘.’{

создать указатель на символ (tmp) – указатель на конец строки

создать вещественный результат = strtod(&postfix[i], &tmp);

создать целочисленную переменную di = tmp - &postfix[i];

i += di -1;

результат добавить в стек

}

}

создать результат равный вершине стека

если стек не пустой, то выбросить исключение

вернуть результат

}

**Консольное приложение**

**Приложение с графическим интерфейсом**

**Результаты**

**Вывод**

**Литература**

Брайан Керниган, Деннис Ритчи «Язык программирования Си»

Брюс Эккель «Философия С++. Введение в стандартный С++»

Стивен Прата «Язык программирования С++. Лекции и упражнения»

**Приложение**