МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Программирование»
Тема: «Динамические структуры данных»

Студент гр. 9303 _____ Максимов Е.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2020

Цель работы.

Изучить основные функции для работы с файловой системой.

Задание.

Вариант лабораторной работы №1.

Требуется написать программу, которая последовательно выполняет подаваемые ей на вход арифметические операции над числами с помощью стека на базе массива.

Стек реализуется на базе класса *CustomStack*, который будет содержать перечисленные ниже методы. Стек должен иметь возможность хранить и работать с типом данных *int.*

Перечень методов класса стека, которые должны быть реализованы:

void push(int val) - добавляет новый элемент в стек;

void pop() - удаляет из стека последний элемент;

int top() - доступ к верхнему элементу;

size_t size() - возвращает количество элементов в стеке;

bool empty() - проверяет отсутствие элементов в стеке;

extend(int n) - расширяет исходный массив на *n* ячеек.

Также необходимо беспечить в программе считывание из потока *stdin* последовательности (не более 100 элементов) из чисел и арифметических операций (+, -, *, /) разделённых пробелом, которые программа должна интерпретировать и выполнить по следующим правилам:

1) если очередной элемент входной последовательности - число, то положить его в стек;

2) если очередной элемент - знак операции, то применить эту операцию над двумя верхними элементами стека, а результат положить обратно в стек (следует считать, что левый операнд выражения лежит в стеке глубже);

3) Если входная последовательность закончилась, то вывести результат, по завершении работы программы в стеке должен быть более только 1 элемент.

4) Если в процессе вычисления возникает ошибка - вызов метода pop или top при пустом стеке, для операции в стеке не хватает

аргументов или в конце работы программы в стеке более 1 элемента, то программа должна вывести "error" и завершиться.

Основные теоретические положения.

1) Стек - это структура данных, в которой хранятся элементы в виде последовательности, организованной по принципу LIFO (Last In — First Out).

Стек не предполагает прямого доступа к элементам. Список основных операций ограничивается операциями помещения элемента в стек и извлечения элемента из стека (*push* и *push* соответственно). Обычно есть возможность посмотреть на верхний элемент стека не извлекая его (*top*) и несколько других функций, таких как проверка на пустоту стека и другие.

2) Очередь - эта структура данных, в которой хранятся элементы в виде последовательности, организованной по принципу FIFO (First In — First Out).

Также как и стек, очередь не предполагает прямого доступа к элементам. Основные операции - добавление и извлечение (*ENQ, enqueue* и *DEQ, dequeue* соответственно). Также обычно есть функции получения первого элемента без его извлечения (*top*), определения размера очереди, проверки на пустоту и другие.

Выполнение работы.

В программе используется класс *CustomStack*, содержащий:

1. Поля типа *private*:
   a. *int count* – количество элементов в стеке;
   b. *int sizemData* – количество выделенной для данных памяти.
2. Поля типа protected:
   a. *int* mData* – указатель на динамический массив данных.
3. Набор методов:
   a. Конструктор класса инициализирует количество элементов count равное 0 и первичный объём выделяемой памяти (определён в *SIZE_UNIT*) и выделяет необходимое количество памяти для стека.

b. Деструктор класса освобождает выделенную память.

c. *void extend(int n)* увеличивает размер стека для хранения дополнительных *n* элементов.

d. *void push(int val)* помещает значение *val* в конец стека; в случае, если памяти недостаточно, вызывает метод *extend* для выделения дополнительной памяти.

e. *void pop()* удаляет последний элемент стека; если элементов в массиве 0, то вызывает метод *stop*.

f. *int top()* возвращает верхний элемент стека; если элементов в массиве 0, то вызывает метод *stop*.

g. *size_t size()* возвращает количество элементов в массиве.

h. *bool IsEmpty()* возвращает 1, если массив пуст, и 0, если не пуст.

i. void print() печатает на экран оставшийся в стеке элемент; если элемент не 1, то вызывает метод *stop*.

j. *void stop()* вызывает деструктор класса, печатает на экран сообщение «error» и останавливает работу программы.

В программе реализованы следующие функции:

1. Функция *char* readStringToSpace()* считывает набор символов с потока ввода до тех пор, пока не встретится символ пробела или маркер конца потока (*EOF*). Функция возвращает NULL, если был считан только маркер конца потока, или динамически выделенный массив символов в обратном случае.

Программа работает следующим образом:

1. Инициализируется стек *stack* и указатель на массив символов *str*.

2. До тех пор, пока в потоке ввода не закончатся символы, по очереди обрабатываются входящие элементы посредством методов класса, функции *int atoi()* и оператора *switch* для определения одного из 4 доступных арифметических действий.

Разработанный программный код см. в приложении А.

Результаты тестирования.

Программный код был протестирован на онлайн-платформе «Stepik».

Результаты тестирования: 1 of 1 test(s) passed.

Выводы.

В ходе лабораторной работы были изучены основные типы динамических структур данных.

Была разработана программа, которая рекурсивно считывает поступающие в программу элементы, согласно определённым правилам выполняет операции с данными на основе стека и печатает результат работы на экран.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#define SIZE_UNIT 8
#define OPERATORS "+-*/"

class CustomStack {
private:
    int count;
    int sizemData;
protected:
    int* mData;
public:
    CustomStack(){
        count=0;
        sizemData = SIZE_UNIT;
        mData = new int[sizemData];
    }
    ~CustomStack(){
        delete [] mData;
    }
    void extend(int n){
        int* newmData = new int[sizemData+n];
        for(int i=0; i<sizemData; i++) newmData[i] = mData[i];
        delete [] mData;
        mData = newmData;
        sizemData+=n;
        return;
    }
    void push(int val){
        if(count+1==sizemData) this->extend(SIZE_UNIT);
        mData[count] = val;
        count++;
        return;
    }
    void pop(){
        if(this->IsEmpty()) this->stop();
        count--;
        return;
    }
    int top(){
        if(this->IsEmpty()) this->stop();
        return mData[count-1];
    }
    size_t size(){
        return count;
    }
    bool IsEmpty(){
        if(count==0) return 1;
        return 0;
    }
    void print(){
        if(this->size()!=1) this->stop();
        else cout << mData[0];
        return;
```

```
    }
    void stop(){
        this->~CustomStack();
        printf("error");
        exit(0);
    }
};


char* readStringToSpace(){
    char* str = new char[16];
    char c;
    int i = 0;
    while(((c=getchar())!=EOF)&&(c!=' ')&&(c!='\n')){
        str[i] = c;
        i++;
    }
    if((c==EOF)&&(i==0)) return NULL;
    str[i] = '\0';
    return str;
}

char* strchr(const char* s, char c){
    while (*s != c) if (!*s++) return 0;
    return (char*)s;
}

int main(){
    CustomStack stack;
    char* str;
    while((str=readStringToSpace())!=NULL){
        if((strchr(OPERATORS, *str))&&(str[1]=='\0')){
            int b = stack.top();
            stack.pop();
            int a = stack.top();
            stack.pop();
            char c = *str;
            switch(c){
                case '+':
                    stack.push(a+b);
                    break;
                case '-':
                    stack.push(a-b);
                    break;
                case '*':
                    stack.push(a*b);
                    break;
                case '/':
                    if(b==0) stack.stop();
                    stack.push(a/b);
                    break;
            }
        } else stack.push(atoi(str));
           delete [] str;
    }
    stack.print();
    return 0;
}
```