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

Студентка гр. 3342 _____ Антипина В.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью работы является изучение основных механизмов языка C++ путем разработки структур данных стека и очереди на основе динамической памяти.

Задание

Вариант 1

Стековая машина.

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

1) Реализовать класс CustomStack, который будет содержать перечисленные ниже методы. Стек должен иметь возможность хранить и работать с типом данных int.

Объявление класса стека:

```
class CustomStack {
public:
// методы push, pop, size, empty, top + конструкторы, деструктор
private:
// поля класса, к которым не должно быть доступа извне
protected:  // в этом блоке должен быть указатель на массив данных
   int* mData;
};
```

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

2) Обеспечить в программе считывание из потока stdin последовательности (не более 100 элементов) из чисел и арифметических

операций (+, -, *, / (деление нацело)) разделенных пробелом, которые программа должна интерпретировать и выполнить по следующим правилам:

Если очередной элемент входной последовательности - число, то положить его в стек,

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),

Если входная последовательность закончилась, то вывести результат (число в стеке).

Если в процессе вычисления возникает ошибка:

- например вызов метода pop или top при пустом стеке (для операции в стеке не хватает аргументов),
- по завершении работы программы в стеке более одного элемента,
- программа должна вывести "error" и завершиться.

Примечания:

- Указатель на массив должен быть protected.
- Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено.
- Предполагается, что пространство имен std уже доступно.
- Использование ключевого слова using также не требуется.

Пример:

Исходная последовательность: 1 -10 - 2 *

Результат: 22

Основные теоретические положения.

Объектно ориентированный язык программирования должен в обязательном порядке обеспечивать:

– возможность формирования сложных объектов, которые сочетают в себе данные и функции;

– механизм доступа/скрытия к данным и функциям со стороны внешних объектов.

Эти пункты в совокупности называют механизмом инкапсуляции (от лат. in capsula), т. е. и данные, и функции находятся в одной «капсуле».

В C++ такой «капсулой» для данных и функций является класс. Класс – это пользовательский тип данных, удовлетворяющий требованиям инкапсуляции:

– в классе могут размещаться как данные (их называют полями), так и функции (их называют методы) для обработки этих данных;

– любой метод и поле исходного класса имеют свой спецификатор доступа.

В данной лабораторной работе потребуются:

○ Public – доступен для всех, т. е. нет ограничений на взаимодействие с полем (считывание/запись) или методом объекта (вызов);

○ Private – доступен только для методов исходного класса.

Есть специальный метод класса, который будет заниматься инициализацией его начального состояния (в момент создания экземпляра класса):

– метод-конструктор всегда носит имя своего класса. (Например, Point());

– у класса всегда есть конструктор по умолчанию (конструктор без аргументов);

– конструкторов может быть много;

– для конструктора не указывается возвращаемое значение.

Перегрузка функций в C++ позволяет определять несколько функций с одинаковым названием при условии, что их аргументы отличаются. Это делает перегрузку удобным инструментом, если требуется решить с помощью функции задачу в разном контексте.

Выполнение работы

Был объявлен класс CustomStack с доступными методами push, pop, size, empty, top, extend, конструктором и деструктором, приватными полями count, capacity, а также защищённым полем mData (указателем на массив).

Конструктор создаёт пустой стек, присваивает count значение 0, capacity — 1, выделяет память под массив целых чисел. Деструктор освобождает память из-под массива чисел. Метод push получает на вход целый аргумент, добавляет его в массив, увеличивает счётчик элементов массива и, если требуется перевыделение памяти, создаёт массив в новым количеством элементов, копирует в него элементы из исходного, удаляет исходный и заменяет его. Если выделить память не удалось, программа выводит ошибку. Таким же образом реализовано расширение стека в функции extend.

Метод pop удаляет последний элемент массива, если элементов в массиве больше 0 (то есть они есть), иначе выводит ошибку и завершает выполнение программы. Счётчик уменьшается, если удаление прошло успешно.

Метод size возвращает значение поля count, empty возвращает true, если элементов в стеке нет, иначе — false. Метод top возвращает последний элемент массива с данными, если стек не пустой, в противном случае выводит ошибку и завершает программу.

В функции main объявляется переменная класса CustomStack — s. Считывается символ из потока ввода, выделяется память под массив символов — буфер, объявляются переменные-счётчики и операнды. Пока не считан символ конца файла или переноса строки в буфер до пробельного символа записываются считанные символы. Затем в буфер записывается символ конца строки. Если перед ним цифра, то в переменную new_el записывается конвертированная в число строка буфер, индекс буфера обнуляется. Если было считано не число, то последние два элемента стека записываются в переменные second, first с помощью метода s.top (чтобы обратиться к

следующему после верхнего элемента, требовалось вызвать s.pop). Оператор switch принимал элемент буфера до символа конца строки. Если это был знак «+», в стек записывался результат сложения first и second, «-» - результат вычитания (причём уменьшаемое — first), «*» - умножения, «/» - деления, где делимое — first. Если считан пробел, то считывается ещё один символ, а индекс буфера обнуляется.

По завершении цикла, если в стеке остался один элемент, он выводится на экран, в противном случае выводится ошибка.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 -10 - 2 *	22	Всё правильно!
2.	5 7 * 7 +	42	

Выводы

Были изучены основные механизмы языка C++ путём разработки структуры данных стека на основе массива. Была реализована программа, выполняющая считывание чисел и запись их в стек, считывание арифметических операций, выполнение их над последними двумя элементами стека и запись результата в стек.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Antipina_Veronika_lb4.cpp

```
#define BLOCK_SIZE 15
class CustomStack{

public:
        CustomStack(){
                this->count = 0;
                this->capacity = 1;
                this->mData = new int [BLOCK_SIZE];
                if(!mData){
                        printf("error\n");
                        exit(1);
                }
        }


        ~CustomStack(){
                delete[] mData;
        }

        void push(int elem){
                this->mData[count] = elem;
                this->count++;
                if(this->count>capacity*BLOCK_SIZE){
                        this->capacity++;
                        int*        new_data        =        new
int[capacity*BLOCK_SIZE*sizeof(int)];
                        if(!new_data){
                                printf("error\n");
                                exit(1);
                        }

                        for(int i = 0; i<count-1; i++){
                                new_data[i] = this->mData[i];
                        }
                        delete[] mData;
                        this->mData = new_data;
                        delete[] new_data;
                }

        }

        void pop(){
                if(this->count>0){
                        mData[this->count] = '\0';
                }else{
                        printf("error\n");
                        exit(1);
                }
                this->count--;
        }
```

```
        size_t size(){
                return this->count;
        }

        bool empty(){
                return this->count==0;
        }
        int top(){
                if(this->count==0){
                        printf("error");
                        exit(0);
                }
                return(mData[this->count-1]);
        }

        void extend(int n){
                int actual = this->capacity*BLOCK_SIZE;
                actual+=n;
                int* new_data = new int[actual*sizeof(int)];

                if(!new_data){
                        printf("error\n");
                        exit(1);
                }

                for(int i = 0; i<count; i++){
                        new_data[i] = this->mData[i];
                }
                delete[] mData;
                this->mData = new_data;
                delete[] new_data;
        }

private:
        size_t count;
        size_t capacity;

protected:
        int* mData;//тут должен быть обычный указатель
};

int main(){
        CustomStack s;
        char ch = getchar();
        int new_el;
        char buffer[50];
        int idx = 0;
        int first;
        int second;

        while(ch!=EOF&&ch!='\n'){
                while(ch!=' ' && ch!='\n' && ch!=EOF){
                        buffer[idx++] = ch;
                        ch = getchar();
                }
```

```
                buffer[idx] = '\0';
                if(isdigit(buffer[idx-1])){
                        new_el = atoi(buffer);
                        s.push(new_el);
                        idx = 0;
                }else{
                        second = s.top();
                        s.pop();
                        first = s.top();
                        s.pop();
                        switch(buffer[idx-1]){
                                case '+':
                                        s.push(first+second);
                                        break;
                                case '-':
                                        s.push(first-second);
                                        break;
                                case '*':
                                        s.push(first*second);
                                        break;
                                case '/':
                                        s.push(first/second);
                                        break;
                                default:
                                        break;
                                //      printf("error\n");
                                //      exit(0);
                        }
                }
                if(ch==' '){
                        ch = getchar();
                        idx = 0;
                }
        }
        if(s.size()==1)
            printf("%d\n",s.top());
        else
            printf("error");

        return 0;
}
```