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. 3342 _____ Иванов С.С.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью данной лабораторной работы является изучение работы с динамическими структурами данных и их создание. Также одна из целей – изучение основ работы с языком C++. Требуется написать программу, моделирующую работу стека на базе **массива.**

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:

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

Выполнение работы

Реализован класс CustomStack, который имеет следующие методы: push, pop, size, empty, top, extend. Он имеет приватные поля, содержащие размер и вместительность стека. В защищенном поле mData находятся данные стека. Реализован main() в котором считываются и выполняются поиск результата последовательности. Есть проверка на пустоту массива, при вызове pop и top.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	-1 2 - 5 7 * +	32

Выводы

Была разработана программа на языке C++, которая создаёт динамическую структуру данных – стек на базе массива. Реализованы методы для работы с созданной структурой и считывание последовательности и поиск её результата.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#define BLOCK 10



class Thrower {

public:
    template <typename T>
    static inline void info(const T &_x)
    { std::cout << _x << std::endl; }

    static inline void error()
    {
        std::cout << "error";
        std::exit(0);

    }

};


class CustomStack
{
public:
    CustomStack(const std::size_t &_size)
    {
        this->_len = 0;
        this->_size = _size;

        if (_size)
            this->mData = new int[_size];

    }

    CustomStack() : CustomStack(2) {}


    void
    push(int val)
    {
        if (this->_size < this->_len+1)
            this->__alloc_new_memory();

        this->mData[this->_len++] = val;

    }

    void
    pop()
```

```
    {
        if (!this->_len)
            Thrower::error();

        --this->_len;
    }

    int
    top()
    {
        if (!this->_len)
            Thrower::error();

        return this->mData[this->_len-1];

    }

    int
    get()
    {
        int val = this->top();
        this->pop();
        return val;

    }

    std::size_t
    size() { return this->_len; }

    bool
    empty() { return (bool)this->_len; }

    void
    extend(int n)
    {
        this->__alloc_new_memory(n);

    }


    ~CustomStack()
    {
        delete [] this->mData;

    }

private:
    std::size_t _size;
    std::size_t _len;

protected:
    int *mData = nullptr;

private:
    void
    __alloc_new_memory(const std::size_t &_block = BLOCK)
    {
        this->_size += _block;
```

```
        int *new_memory_ptr = new int[this->_size];

        for (size_t i = 0; i < this->_len; ++i)
            new_memory_ptr[i] = this->mData[i];

        delete [] this->mData;
        this->mData = new_memory_ptr;

    }

};


bool
is_number(const std::string &str)
{
    if (str.size() == 1)
    {
        return std::isdigit(str[0]);

    }

    bool negative = false;

    if (str[0] == '-')
        negative = true;

    for (char ch : str)
    {
        if (negative && ch == '-')
        {
            continue;

        }

        if (!std::isdigit(ch))
            return false;

    }
    return true;

}


int main()
{
    std::string sequence;
    std::getline(std::cin, sequence);

    std::stringstream ss(sequence);
    std::string token;

    CustomStack stack;
    int a, b;

    while (std::getline(ss, token, ' '))
    {
        if (is_number(token))
```

```
            {
                stack.push(
                    std::stoi(token)
                );

                continue;
            }

            a = stack.get();
            b = stack.get();

            switch (*token.data())
            {
            case '+':
                stack.push(b + a);
                break;

            case '-':
                stack.push(b - a);
                break;

            case '*':
                stack.push(b * a);
                break;

            case '/':
                stack.push(b / a);
                break;

            }

        }

        if (stack.size() != 1)
            Thrower::error();

        Thrower::info(stack.top());

        return 0;

    }
```