**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. 3384 _____ Рудаков А.Л.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

**Цель работы.**

Изучить теорию работы с динамическими структурами данных в языке C++. Получить практические навыки путем написания программного кода.

**Задание.**

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) беспечить 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.

Пример:

*Исходная последовательность:* 1 -10 - 2 *

*Результат:* 22

**Выполнение работы.**

В начале работы была написана функция *void errors()*, которая выводит информацию об ошибке и завершает работу программы.

Далее был создан класс *class CustomStack()*. Он имеет поле *ListNode* mHead*, являющееся *protected*, в нем хранится указатель на последний элемент стека. Также класс имеет *private* метод *void errors()*, для возможности вызывать ошибки внутри класса. Внутри *public* располагаются методы класса, конструктор и деструктор.

Конструктор *CustomStack(ListNode* Head = NULL)* присваивает полю класса *mHead* указатель на поданный элемент, либо на *NULL*, если элемент не подан.

Деструктор класса *~CustomStack()* проходит по всему стеку, стирая указатели на верхние элементы, пока в стеке не останется элементов.

Методы класса. *void push(int val)* — метод, благодаря которому в стек добавляется элемент. Внутри него создается указатель на *ListNode*, выделяется под него память, записывается поданное значение и указатель на последний элемент стека, после чего полю *mHead* присваивается указатель на данный новый элемент. Метод *void pop()* удаляет элемент из стека, если стек не пустой, проверяя это методом *empty()*, посредством того, что присваивает полю *mHead* указатель на предыдущий элемент стека, а память из под удаленного элемента очищается, если стек пустой, вызывается *errors()*. Метод *int top()* возвращает значение верхнего элемента в стеке, если стек не пустой, иначе вызывает *errors()*. Метод *size_t size()* считает длину стека, посредством прохождения по всем элементам стека, пока указатель на элемент не станет равным *NULL*, после этого возвращает посчитанное количество. Метод *bool empty()* проверяет пустой ли стек, если поле *mHead* указывает на *NULL*, то стек пустой и возвращается *true*, в противном случае возвращается *false*.

Далее была написана функция *void action(CustomStack* stack, char sign)*, которая выполняет полученное заданное арифметическое действие. На вход ей подается указатель на стек и символ операции. Вначале инициализируются переменные *first* и *second* типа *int*, которые инициализируются значениями верхних двух элементов стека при помощи метода *top()*, при элементы из стека удаляются благодаря методу *pop()*. Далее, если знак сложения, вычитания,

умножения или деления, то выполняется сложение, вычитание, умножение или деление соответственно, результат записывается в стек при помощи метода *push()*. Если знак не является знаком операции, то вызывается ошибка.

Далее написана функция *main()*. В ней инициализируется строка *data*, в которую при помощи *fgets()* считывается вводимая строка. Создается стек *stack* и *int flag*. Далее идет цикл по длине введенной строки, внутри которой идет проверка текущего символа на равенство символу пробела или переноса строки. Если это так, то определяется длина до ближайшего предыдущего пробела или начала строки, если длина больше одного, то это точно число, поэтому при помощи функции *stoi()* строка переводится в число и добавляется в стек. Если длина равна 1, то проверяется то, является ли символ цифрой при помощи функции *isdigit()*, если да, то символ преобразуется в цифру и добавляется в стек. Если не один вариант не подошел, то это знак операции, следовательно вызывается функция *action()* для обработки данной операции. После цикла значению *ans* присваивается значение последнего элемента в стеке, значению *len* — размер стека. Если размер стека не равен 1, то вызывается *errors()*, в противном полученное значение ans выводится на экран.

Разработанный программный код см. в приложении А.

**Выводы.**

Была изучена теория работы с динамическими структурами данных в языке C++. В работе был создан и использован класс CustomStack, внутри которого были созданы методы и поля.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main_lb4.cpp

```
void errors()
{
    cout << "error";
    exit(0);
}

class CustomStack {

    public:

        CustomStack(ListNode* Head = NULL)
        {
            mHead = Head;
        }

        ~CustomStack()
        {
            ListNode* temp;
            while (mHead != NULL)
            {
                temp = mHead;
                mHead = mHead->mNext;
                delete temp;
            }
        }

        void push(int val)
        {
            ListNode* new_el = new ListNode;
            new_el->mNext = mHead;
            new_el->mData = val;
            mHead = new_el;
        }

        void pop()
        {
            if (not empty())
            {
                ListNode* last_el = mHead;
                mHead = mHead->mNext;
                delete last_el;
            }
            else
            {
                errors();
            }
        }

        int top()
        {
            if (not empty())
            {
```

```
            return mHead->mData;
        }
        else
        {
            errors();
        }
    }

    size_t size()
    {
        size_t count = 0;
        ListNode* temp = mHead;
        while (temp != NULL)
        {
            count++;
            temp = temp->mNext;
        }
        return count;
    }

    bool empty()
    {
        if (mHead == NULL)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
// методы push, pop, size, empty, top + конструкторы, деструктор

private:

// поля класса, к которым не должно быть доступа извне
    void errors()
    {
        cout << "error";
        exit(0);
    }

protected:  // в этом блоке должен быть указатель на голову

    ListNode* mHead;
};

void action(CustomStack* stack, char sign)
{
    int first, second;
    first = (*stack).top();
    (*stack).pop();
    second = (*stack).top();
    (*stack).pop();

    if (sign == '+')
    {
        int sum = first + second;
```

```
        (*stack).push(sum);
    }
    else if (sign == '-')
    {
        int diff = second - first;
        (*stack).push(diff);
    }
    else if (sign == '*')
    {
        int compos = first * second;
        (*stack).push(compos);
    }
    else if (sign == '/')
    {
        int part = second / first;
        (*stack).push(part);
    }
    else
    {
        errors();
    }
}

int main()
{
    char data[101];
    fgets(data, 100, stdin);

    int flag = 0;
    CustomStack stack;

    for (size_t i = 0; i < strlen(data); i++)
    {
        if (data[i] == ' ' || data[i] == '\n')
        {
            int size = i - flag;
            int val;
            char value = '1';
            if (flag == -1) flag = 0;

            char* temp = new char[size+1];
            strncpy(temp, data + flag, size);

            if (size > 1)
            {
                val = stoi(temp);
            }
            else if (isdigit(temp[0]))
            {
                val = stoi(temp);
            }
            else
            {
                value = temp[0];
            }

            if (value == '1')
            {
```

```
                stack.push(val);
                // znak
            }
            else
            {
                action(&stack, value);
                //chislo
            }

            flag = i + 1;
            delete temp;
        }
    }

    int ans, len;
    ans = stack.top();
    len = stack.size();

    if (len != 1)
    {
        errors();
    }
    else
    {
        cout << ans;
    }
}
```