МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического обеспечения электронно-вычислительных машин

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

ТЕМА: ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

ВАРИАНТ: 1

Студентка гр. 0382 _____ Рубежова Н.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучить динамические структуры данных, их реализацию в языке Си++, освоить принцип работы классов, используя их в программном коде.

Задание.

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

Основные теоретические положения.

Давайте рассмотрим способы реализации таких структур данных как стек и очередь. Фактически, обе структуры данных можно представлять в памяти либо в виде однонаправленного списка, либо в виде массива.

Представление в виде списка

При такой организации хранения элементов, операции добавления элемента в стек и операции удаления элемента из стека эквивалентны операциям над списком: добавление в голову и удаление из головы соответственно. Таким образом, каждый элемент имеет указатель на следующий, лежащий "ниже" него в стеке.

В случае очереди, добавление элемента эквивалентно вставке элемента в конец списка, а извлечение - удаление элемента из головы списка. Если вместе с указателем на голову списка хранить указатель на его последний элемент, операция вставки перестает быть затратной (ввиду отсутствия необходимости проходить список до конца каждый раз). Таким образом, каждый элемент имеет указатель на следующий в порядке очереди элемент.

Представление в виде массива

Стек можно легко реализовать на основе массива. Для этого достаточно хранить индекс "верхнего" элемента в стеке. Операция добавления сопровождается инкрементом этого индекса и записью в соответствующую ячейку нового значения. Операция извлечения сопровождается декрементом этого индекса. Дополнительно, может потребоваться реализовать возможность увеличения и уменьшения размера массива

Реализовать на основе массива очередь немного сложнее. В отличие от стека, потребуется хранить два индекса - индекс первого элемента и индекс последнего. Вставка сопровождается инкрементом индекса последнего элемента и записью нового значения, а извлечение инкрементом индекса первого. В случае равенства индексов - очередь пуста. Проблема возникает в

том случае, когда индекс последнего подбирается к границе массива, при этом начало массива уже не используется. Эту проблему можно решить, начав циклически использовать ячейки (в этой ситуации индекс последнего элемента может быть меньше индекса первого)

К недостаткам реализации в виде списка можно отнести более медленную работу и незначительное увеличение затрат памяти. К преимуществам - произвольный объем структуры данных.

Выполнение работы.

1. Опишем *class CustomStack.* Объекты этого класса будут являться стеком на основе массива. Он имеет *public* методы: *push, empty, pop, top, size, extend,* которые мы опишем чуть позже. Поля *protected: int* mData* – хранит указатель на наш целочисленный массив-стек, *int count* – хранит текущее число элементов в стеке, *int maxcount* – число элементов, под которое выделена память для массива в текущий момент, если понадобится «расширить память», это число будет изменяться.

2. Опишем конструктор класса *CustomStack(),* который будет вызываться при каждом создании объекта класса. Присвоим полю *maxcount* изначальное число элементов, под которое будет выделяться память. Выделим память с помощью функции *calloc().* А полю *count* присвоим текущее число элементов на стеке равное нулю.

3. Опишем метод *void push(int val).* Вызов метода позволяет положить значение *int val* на стек, предварительно при этом проверив, достаточно ли памяти для нового элемента. Если текущее число элементов на стеке(поле *count)* равно значению поля *maxcount,* то вызывается метод расширения памяти *extend().* Далее, когда памяти достаточно, обращаемся к элементу массива, в который можно положить значение, разыменовав указатель, и передаем туда значение *int val.* Изменяем текущее число элементов поле *count* на единицу. Таким образом, целочисленное значение «кладется на стек».

4. Опишем метод *bool empty()*. Вызов метода позволяет определить, является ли стек пустым на текущий момент. Для этого достаточно проверить поле *count* у объекта. Если *count==0,* то возвращаем логическое *true,* в ином случае – *false.*

5. Опишем метод *void pop()*. Вызов метода позволяет удалить элемент из стека. Проверим, является ли стек пустым, вызвав метод *empty()*. Если стек пустой, то удалять нечего, поэтому выводим *"error",* в ином случае – удаляем элемент из стека, для этого обращаемся к этому элементу и обнуляем данные, а также не забываем изменить поле *count* – текущее число элементов в стеке уменьшилось на единицу.

6. Опишем метод *int top()*. Вызов метода возвращает целочисленное значение верхнего элемента на стеке. Проверяем является ли стек пустым, если пустой – выводим ошибку, если нет – то получаем значение элемента, разыменовывая указатель на последний элемент массива, и возвращаем его.

7. Опишем метод *size_t size()*. Вызов метода возвращает текущее количество элементов на стеке. Для этого достаточно привести к типу size_t значение поля *count,* отвечающее за текущее количество элементов на стеке.

8. Опишем метод *void extend(int n)*. Вызов метода позволяет расширить память, выделяемую под стек, на *n* байт. Для этого увеличим число *maxcount* на *n* и расширим память с помощью функции *realloc()*.

9. Опишем основную функцию *int main()*. Создаем объект класса *CustomStack obj*. Выделяем память под строку, в которой содержатся числа, над которыми нужно произвести вычисления, и математические операции. Считываем строку с помощью *fgets()*. Далее нам нужно извлечь числа и математические операции из этой строки. Реализуем это с помощью функции *strtok(),* которая на каждой итерации цикла *while* будет возвращать указатель на строку-кусочек, то есть элемент-токен. Попробуем преобразовать этот токен в число с помощью функции *atoi()*, если она вернет не *NULL*, значит преобразование прошло успешно, значит, это число. Кладем его на стек с помощью метода *push()*, обратившись к объекту класса. Если вернулся *NULL,*

значит этот токен – это математическая операция, которую надо выполнить над элементами стека. Получаем и извлекаем верхние два числа из стека поочередно с помощью obj.*top()* и *obj.pop()*. Если наш токен – *'+'*, то складываем извлеченные из стека числа и кладем результат на стек с помощью *obj.push()*. Если наш токен – *'-'*, то вычитаем одно извлеченное из стека число из другого и кладем результат на стек с помощью *obj.push()*. Если наш токен – *'*'*, то перемножаем извлеченные из стека числа и кладем результат на стек с помощью *obj.push()*. Если наш токен – *'-'*, то делим второе извлеченное из стека число на второе и кладем результат на стек с помощью *obj.push()*. Как только исходная строка закончилась, токены закончились, нужно проверить, действительно, ли на стеке осталось одно число – результат всех операций, тогда выводим это число, получив его функцией *obj.top()*, если на стеке больше одного числа, то выводим ошибку *'error'*.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 -10 – 2 *	22	Вывод верный
2.	1 2 + 3 4 – 5 * +	-2	Вывод верный

Выводы.

Были изучены динамические структуры данных, их реализация на языке Си++, а также освоен принцип работы с классами посредством использования их в программном коде.

Разработана программа, которая последовательно выполняет подаваемые ей на вход арифметические операции над числами с помощью стека на базе массива.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack {
public:
    CustomStack(){
        maxcount=100;
        mData=(int*)calloc(maxcount,sizeof(int));
        count=0;
    }
    void push(int val){
        if(count==maxcount)
            extend(5);
        *(mData+count)=val;
        count++;
    }
    bool empty(){
        if(count==0)
            return true;
        else
            return false;
    }
    void pop(){
        if(!empty()){
            *(mData+count-1)=0;
            count--;
        }
        else {
            cout << "error";
        }
    }
    int top(){
        if(!empty()){
            int res=*(mData+count-1);
            return res;
        }
        else
            cout << "error";
        return 0;
    }
    size_t size(){
        size_t s=(size_t)count;
        return s;
    }
    void extend(int n){
        maxcount+=n;
        mData=(int*)realloc(mData,maxcount*sizeof(int));
    }

protected:
    int* mData;        //стек
    int count;
    int maxcount;
```

```
};

int main() {
    CustomStack obj;
    char* str=(char*)calloc(100,sizeof(char));
    int num,res=0;
    fgets(str,100,stdin);
    char* pch=strtok(str," ");
    while(pch!=NULL){
        num=atoi(pch);
        if(num)
            obj.push(num);
        else{
            int num_r=obj.top();
            if(!num_r)
                return 0;
            obj.pop();

            int num_l=obj.top();
            if(!num_l)
                return 0;
            obj.pop();

            if((*pch)=='+'){
                res=num_l+num_r;
                obj.push(res);
            }
            if((*pch)=='-'){
                res = num_l - num_r;
                obj.push(res);
            }
            if((*pch)=='*'){
                res = num_l * num_r;
                obj.push(res);
            }
            if((*pch)=='/'){
                res=num_l/num_r;
                obj.push(res);
            }
        }
        pch=strtok(NULL," ");
    }
    if(obj.size()==1)
        cout << obj.top();
    else
        cout << "error"<<endl;
    return 0;
}
```