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 _____ Песчатский С. Д.

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 6

Расстановка тегов.

Требуется написать программу, получающую на вход строку, (без кириллических символов и не более 3000 символов) представляющую собой код "простой" html-страницы и проверяющую ее на валидность. Программа должна вывести correct если страница валидна или wrong.

html-страница, состоит из тегов и их содержимого, заключенного в эти теги.

Теги представляют собой некоторые ключевые слова, заданные в треугольных скобках. Например, <tag> (где tag - имя тега). Область действия данного тега распространяется до соответствующего закрывающего тега </tag> который отличается символом /. Теги могут иметь вложенный характер, но не могут пересекаться.

<tag1><tag2></tag2></tag1> - верно

<tag1><tag2></tag1></tag2> - не верно

Существуют теги, не требующие закрывающего тега.

Валидной является html-страница, в коде которой всякому открывающему тегу соответствует закрывающий (за исключением тегов, которым закрывающий тег не требуется).

Во входной строке могут встречаться любые парные теги, но гарантируется, что в тексте, кроме обозначения тегов, символы < и > не встречаются. аттрибутов у тегов также нет.
Теги, которые не требуют закрывающего тега:
, <hr>.

Стек (который потребуется для алгоритма проверки парности тегов) требуется реализовать самостоятельно на базе массива. Для этого необходимо:

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

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
char** mData;
};
```

Перечень методов класса стека, которые должны быть реализованы:

void push(const char* val) - добавляет новый элемент в стек

void pop() - удаляет из стека последний элемент

char* 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

Примечания:

Указатель на массив должен быть protected.

Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено(<cstring> и <iostream>).

Предполагается, что пространство имен std уже доступно.

Использование ключевого слова using также не требуется.

Пример:

Входная строка:

<html><head><title>HTML Document</title></head><body> This text is bold,
this is bold and italics </body></html>

Результат: correct

Выполнение работы

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

Конструктор создаёт пустой стек, присваивает mSize значение 0, mCap — 0, выделяет память под массив целых чисел, приравнивая его указателю nullptr. Деструктор освобождает память из-под массива чисел. Метод push получает на вход const char*, добавляет его в массив, увеличивает счётчик элементов массива и, если требуется расширяет допустимый размер стека, делает это с помощью функции extend.

Метод pop удаляет верхний элемент стека, если элементов в массиве больше 0 (то есть они есть). Счётчик уменьшается, если удаление прошло успешно.

Метод size возвращает значение поля mSize, empty возвращает true, если элементов в стеке нет, иначе — false. Метод top возвращает верхний элемент стека.

В функции main объявляется переменные bool swit, string line, int size = 0, int curr = 0 и массив строк char ** tags для считывания строки и преобразования её в массив тэгов. При помощи цикла for, логических условий, функций класса CustomStack и объекта этого класса stack выполняется проверка данной html строки на корректность. Как только программа находит ошибку, она выводит “incorrect” и останавливает свою работу. Если не было обнаружено ни одной ошибки программа выводит “correct”.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	<html><head><title>HTML Document</title></head><body> This text is bold, this is bold and italics </body></html>	correct	
2.	<html><head><title>HTML Document</title></head><body> This text is bold, this is bold and italics </body><html>	incorrect	

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: main.cpp

```
void mem_error(){
     cout << "Memory error!" << endl;
     exit(1);
}
class CustomStack{
    public:
        CustomStack(){
            mData=nullptr;
            mSize=0;
            mCap=0;
        }
        void push(const char* val){
            if(mSize==mCap){
                extend(1);
            }
            mData[mSize++]=(char*)val;
        }
        void pop(){
            if(mSize!=0){
                mSize--;
            }
        }
        char * top(){
            return mData[mSize-1];
        }
        size_t size(){
            return mSize;
        }
        bool empty(){
            return mSize==0;
        }
        void extend(int n){
         char** newData;
            mCap=mCap+n;
            newData=new char* [mCap];
            if(newData==nullptr){mem_error;}
            for(int i=0; i<mSize; i++){
                newData[i]=mData[i];
            }
            delete [] mData;
            mData=newData;
        }
        ~CustomStack(){
           delete [] mData;
        }
    private:
    int mSize;
    int mCap;
    protected:
    char ** mData;
```

```
};
int main(){
    char * tmps;
    tmps = new char [51];
    if(tmps==nullptr){mem_error;}
    bool swit = false;
    CustomStack stack;
    string line;
    int size=0; int curr=0;
    getline(cin, line);
    char ** tags;
    tags = new char * [100];
    if(tags==nullptr){mem_error;}
    for(int i=0; i<3000; i++){
        if(line[i]=='<'){
            swit=true;
            i++;
            tags[size]=new char [51];
            if(tags[size]==nullptr){mem_error;}
            }
        if(line[i]=='>'){swit=false;size++;curr=0;}
        if(swit){tags[size][curr++]=line[i];}
    }
    for(int i=0; i<size; i++){
        if(tags[i][0]=='/'){
            char * tmpf;
            tmpf = new char [51];
            if(tmpf==nullptr){mem_error;}
            for(int j=0; tags[i][j+1]; j++){
                tmpf[j]=tags[i][j+1];
            }
            if(stack.empty()){
                cout << "incorrect";
                stack.push(tmpf);
                cout <<"a\n";
                break;
            }
            if(strcmp(stack.top(), tmpf)!=0){
                cout <<"wrong";
                break;
            }
            if(strcmp(stack.top(), tmpf)==0){
                stack.pop();
            }
            delete [] tmpf;
        }
        else{
            if(strcmp(tags[i],   "br")==0   ||   strcmp(tags[i],
"hr")==0){}
            else{
                stack.extend(1);
                stack.push(tags[i]);
            }
        }
    }
    if(stack.empty()){
```

```
            cout << "correct";
        }
        delete [] tags;
        delete [] tmps;
        return 0;
    }
```