МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Программирование»

Тема: Динамические структуры данных.

Студент гр. 1304 _____ Стародубов М.В.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2022

Цель работы.

Научиться создавать динамические структуры данных на базе классов испозьзуя язык программирования C++.

Задание.

Вариант – 6.

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:

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

Выполнение работы.

Для выполнения поставленной задачи необходимо написать класс *CustomStack,* который будет предоставлять доступ к следующим *public*-методам: *push, pop, top, size, empty*. Также реализованный класс имеет *private*-метод *extend*, *private*-поля *bufeer_size* (размер области памяти, выделенной под хранение элементов стека) и *elements_count* (количество элементов в стеке). Поле, в котором хранится указатель на массив, в котором хранятся элементы стека является *protected.* Элементами данного стека являются массивы символов (строки в стиле Си).

В конструкторе класса *CustomStack* происходит инициализация полей *buffer_size, elements_count, mData*.

В деструкторе класса *CustomStack* происходит освобождение памяти, выделенной под оставшиеся в стеке элементы, а также освобождение памяти под хранение массива элементов.

Метод *push* добавляет в стек элемент *val*, т. е. строку, которая находиться по данному указателю. В случае, если в стеке не хватает памяти для добавления нового элемента, то происходит расширение выделенной области памяти с помощью метода *extend*. Добавление элемента происходит следующим образом: сначала происходит выделение памяти под хранение строки, указатель на которую содержиться в переменной *val*, далее происходит копирование содержимого данной строки в выделенную область памяти.

Метод *pop* удаляет из стека последний элемент. Если метод был вызван для пустого стека, то выбрасывается исключение. В процессе выполнения данного метода происходит освобождение области памяти, выделенной под хранение последнего элемента.

Метод *top* возвращает последний элемент стека, т. е. указатель на строку, хранящуюся по последнему указателю в массиве элементов. Если метод был

вызван для пустого стека, то выбрасывается исключение.

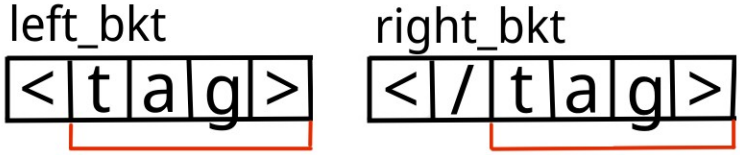
Метод *size* возвращает количество элементов в стеке.

Метод *empty* возвращает истину, если стек пуст, и ложь в противном случае.

Метод *extend* производит расширение массива элементов на *n* элементов. С помощью оператора *new* происходит выделение памяти под массив, в котором на *n* элементов больше, чем в старом, далее в новый массив происходит копирование всех данных из старого, память под старый массив освобождается.

Функция *isPair* проверяет, является ли строка, хранящаяся по указателю *right_bkt*, закрывающим тегом для тега, хранящегося по указателю *left_bkt*. С помощью функции *strcmp* происходит сравнение частей, которые должны быть одинаковыми у открывающего и у закрывающего тегов так, как показано на рисунке 1 (сравниваемые части подчеркнуты красным).

Рисунок 1. Сравнение открывающего и закрывающего тегов.



В функции *main* происходит считывание входного потока данных в массив *input*. Так как все теги заключены между угловыми скобками («<» и «>»), то будет производиться поиск пары таких скобок и дальнейшие действия будут выполняться только с последовательностями символов, заключенными между ними. Для хранения открывающих тегов создается экземпляр описанного выше стека. Далее в цикле будет происходить перемещение по всем тегам, как по парам открывающих и закрывающих угловых скобок. В переменную *cur_bkt* копируется последовательность символов, заключенная между найденной парой открывающей и закрывающей угловых скобок (включая угловые скобки). Далее

происходит проверка, что найденный тег не является тегом, не требующим закрывающего тега. Если символ, идущий после открывающей угловой скобки тега не является слешем, то найденный тег является открывающим, поэтому происходит его добавление в стек. В противном случае найденных тег является закрывающим. Если стек пуст или найденный тег не является закрывающим для последнего добавленного в стек открывающего тега, то расстановка тегов во входном потоке данных является неверной, программа печатает на экран строку «wrong» и завершает работу. Иначе, когда последний открывающий тег в стеке является парным для найденного закрывающего, то из стека извлекается последний открывающий тег. Далее происходит поиск следующего тега во входном потоке данных. Если после завершения работы цикла в стеке остаються элементы, то расстановка тегов также является неверной, и на экрн печатается строка «wrong», иначе печатается строка «correct».

Выводы.

В ходе выполнения данной лабораторной работы была написана программа на языке программированя C++, проверяющая правильность расстановки тегов в простой html-странице. Для выполнения данной задачи в программе был реализован класс динамической структуры данных «стек».

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <cstring>

using namespace std;

class CustomStack
{
public:
	CustomStack()
	{
		buffer_size = 0;
		elements_count = 0;
		mData = NULL;
	}

	~CustomStack()
	{
		for (int i = 0; i < elements_count; i++)
			delete[] mData[i];
		delete[] mData;
	}

	void push(const char *val)
	{
		if (elements_count == buffer_size)
			extend(100);
		mData[elements_count] = new char [strlen(val)+1];
		strcpy(mData[elements_count++], val);
	}

	void pop()
	{
		if (empty())
		{
			throw 1;
			return;
		}
		elements_count--;
		delete[] mData[elements_count];
	}

	char *top()
	{
		if (empty())
		{
			throw 1;
			return NULL;
		}
		return mData[elements_count-1];
	}
```

```
        size_t size()
        {
                return elements_count;
        }

        bool empty()
        {
                if (!elements_count) return true;
                return false;
        }

private:
        void extend(int n)
        {
                char **tmp = new char* [buffer_size + n];
                memcpy(tmp, mData, sizeof(char*)*elements_count);
                delete[] mData;
                mData = tmp;
                buffer_size += n;
        }

        int buffer_size;
        int elements_count;

protected:
        char **mData;

};

bool isPair(char *left_bkt, char *right_bkt)
{
        if (!strcmp(left_bkt+1, right_bkt+2)) return true;
        return false;
}

int main()
{
        char input[3000];
        cin.getline(input, 3000);

        char *left_bkt = strstr(input, "<");
        char *right_bkt = strstr(left_bkt, ">");

        CustomStack stack;

        char *cur_bkt;
        while (left_bkt && right_bkt)
        {
                cur_bkt = new char [right_bkt - left_bkt + 2];
                memcpy(cur_bkt, left_bkt, right_bkt - left_bkt + 1);
                cur_bkt[right_bkt - left_bkt + 1] = '\0';

                if (strcmp(cur_bkt, "<br>") && strcmp(cur_bkt, "<hr>"))
                {
                        if (cur_bkt[1] != '/')
                        {
```

```
                                        stack.push(cur_bkt);
                                } else
                                {
                                        if (stack.empty() || !isPair(stack.top(),
cur_bkt))
                                        {
                                                cout << "wrong";
                                                delete[] cur_bkt;
                                                return 0;
                                        }

                                        stack.pop();
                                }
                        }

                        delete[] cur_bkt;
                        left_bkt = strstr(right_bkt, "<");
                        if (left_bkt)
                                right_bkt = strstr(left_bkt, ">");
                }

                if (stack.empty()) cout << "correct";
                else cout << "wrong";

                return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Данные тестирования представлены в таблице 1.

Таблица 1 – Результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	<tag1><tag2></tag2></tag1>	correct	Результат корректен
2.	<tag1><tag2></tag1></tag2>	wrong	Результат корректен
3.	<html><head><title>HTML Document</title></head><body> This text is bold, this is bold and italics </body></html>	correct	Результат корректен
4.	<html><head><title>Some title</title></head><body> Some text </body></html>	correct	Результат коррекрен
5.	<html><head><title>Some title</title></head><body> Some text w/ wrong tags </body></html>	wrong	Результат корректен
6.	<html><head><title>Some title</title></head><body> Some text w/ more wrong tags </body></html>	wrong	Результат корректен
7.	<html><head><title>Some title</title></head><body> Some text w/ even more wrong tags </body></html>	wrong	Результат корректен