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. 3344 _____ Клюкин А.В.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

Цель работы

Получить навыки работы с классами на языке c++ в реализации стека для выполнения задачи.

Задание.

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

Выполнение работы

В самом начале был описан класс CustomStack, который помогает моделировать работу стека на базе массива. Он имеет следующие основные функции: push – операция вставки нового элемента; pop – операция удаления элемента с вершины стека; empty – проверка стека на наличие в нем элементов; size – подсчет количества элементов; top – операция просмотра верхнего элемента; extend - для расширения массива на n ячеек. Так же были вынесены вспомогательные функции для оперирования подзадачами. После начинается основная часть, в которой создаются все необходимые переменные и считывается строка. Далее идет цикл, который перебирает каждый элемент этой строки и в случае, если символ - открывающий тег, то ставится флаг для дальнейшей записи содержимого тега в отдельную переменную tag. Так же в случае, если после открывающего символа идет парный тег “/”, то ставится иной флаг, но запись в переменную tag все равно будет. Когда нахождится закрывающий символ, то идет проверка на непарный тег. В случае, если это парный и не имеет символа “/” (что проверяется через флаг) - он добавляется в стек через p.push(tag). Потом обнуляются переменный с флагом и записанным тегом. Если же символ “/” присутствовал, то идет проверка на соответствие с последним тегом, добавленным в стек. Если не совпал, то очевидна ошибка в расставлении и тогда вся строка неверная, поэтому выводится “wrong” и завершается работа программы. Если все совпало, то из стека удаляется тег. В конце цикла - стек должен оставаться пустым, что является следствием валидной строки, поэтому выводится “correct”.

Тестирование

Результаты тестирования представлены в таблице 1.

Таблица 1 – Результаты тестирования

№	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	-

Выводы

Получены навыки работы с классами и реализацией стека на c++.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Klyukin_Aleksandr_lb4.c

```
class CustomStack{
public:
	CustomStack(size_t initialCapacity){
		this->mCapacity = initialCapacity;
		this->mData = new char *[initialCapacity];
		this->mIndex = -1;
	}
	CustomStack() : CustomStack(10)
	{
		// 10 -- начальный размер стека, вызов другого конструктора
	}

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

	void push(const char *val){
		this->ensureSpace();   //   проверка,   что   размера   массива
достаточно для нового элемента
		this->mIndex++;
		mData[this->mIndex] = new char[strlen(val) + 1];

		strcpy(mData[this->mIndex], val);
	}

	void pop(){
		if (this->empty()){
			throw logic_error("pop() called on empty stack");
		}
		this->mIndex--; // “удаление” элемента
	}

	char *top(){
		return this->mData[this->mIndex];
	}

	size_t size() const{
		return this->mIndex + 1;
	}

	bool empty() const{
		return this->mIndex == -1;
	}

	void extend(int n){
		if (n <= 0){
			throw  logic_error("extend()  called  with  a  nonpositive
argument");
		}
```

```
            this->resize(this->mCapacity + n);
      }

protected:
      size_t mCapacity;
      size_t mIndex;
      char **mData;

      size_t getNewCapacity() const{
            // получение нового размера
            return this->mCapacity * 3 / 2 + 1;
      }

      void ensureSpace()
      {
            if (this->mIndex + 1 == mCapacity){
                  // если достигнут максимальный размер
                  size_t newCapacity = this->getNewCapacity();
                  this->resize(newCapacity);
            }
      }

      void resize(size_t newCapacity)
      {
            if (newCapacity == mCapacity){
                  return;
            }
            char **newData = new char *[newCapacity];
            copy(this->mData, this->mData + this->mCapacity, newData); //
копирование данных при помощи функции из заголовочного файла<algorithm>
            delete[] this->mData;
            this->mData = newData;
            this->mCapacity = newCapacity;
      }
};

int main()
{
      CustomStack p;
      char *text, *tag;
      int flag = 0, tagLen = 0;
      text = new char[3001];
      tag = new char[3001];
      fgets(text, 3000, stdin);

      for (size_t i = 0; i < strlen(text); i++){
            if (text[i] == '<'){
                  flag = 1;
            }
            else if (flag == 1 && text[i] == '/'){
                  flag = 2;
            }
            else if (text[i] == '>'){
                  tag[tagLen] = '\0';
                  if  (flag  ==  1  &&  (strcmp("<hr",  tag)  !=  0  &&
strcmp("<br", tag) != 0)){
                        p.push(tag);
                  }
```

```
                else if (flag == 2){
                        for (size_t j = 1; j < strlen(p.top()); j++){
                                if (tag[j + 1] != p.top()[j]){
                                        cout << "wrong" << endl;
                                        return 0;
                                }
                        }
                        p.pop();
                }

                flag = 0;
                tagLen = 0;
        }

        if (flag != 0){
                tag[tagLen++] = text[i];
        }
    }
    cout << "correct" << endl;


    return 0;
}
```