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. 3341 Шаповаленко Е.В

Преподаватель Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Изучить динамические структуры данных. Применить знания на практике написав программу на языке C++, использующую стек для решения задачи.

Задание

Вариант 5

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, атрибутов у тегов также нет.

Теги, которые не требуют закрывающего тега:
, <hr>.

Класс стека (который потребуется для алгоритма проверки парности тегов) требуется реализовать самостоятельно на базе списка. Для этого необходимо реализовать класс CustomStack, который будет содержать перечисленные ниже методы. Стек должен иметь возможность хранить и работать с типом данных char*.

Структура класса узла списка:

```
struct ListNode {
    ListNode *mNext;
    char *mData;
};
```

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 голову
    ListNode *mHead;
};
```

Перечень методов класса стека, которые должны быть реализованы:

- void push(const char* tag) - 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

Примечания:

- Указатель на голову 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.
- Структуру ListNode реализовывать самому не надо, она уже реализована.

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

Объявляются глобальные константы для тегов.

Реализуется класс *CustomStack*. Методы класса:

- Конструктор: полю *mHead* задается значение *nullptr*;
- *Push*: добавляется новый элемент в начало списка (в голову стека), выделяет память под поле *mData* элемента;
- *Pop*: удаляется первый элемент из списка (голова стека), очищает память, выделенную под поле *mData*;
- *Size*: возвращает количество элементов в стеке;
- *Empty*: возвращает *true,* если стек пустой, иначе *false*;
- *Top*: возвращает значение поля *mData* первого элемента списка (головы стека);
- Деструктор: удаляет все элементы стека, очищает выделенную память;

В функции *main* считывается строка в *buffer*. Создается объект класса *CustomStack*. Цикл итерируется по *buffer*. Если находится тег (находятся символы *"<"* и *">"*), он добавляется в стек. Если тег не имеет соответствующего ему закрывающего тега (*"<hr>"* и *"
"*), он удаляется из стека. Для остальных тегов, когда размер стека не меньше двух, проверяется, соответствует ли тег в голове стека предыдущему (например, *"</tag>"* и *"<tag>"*)? Если да, они оба удаляются из стека. Иначе они оба остаются.

Таким образом, если теги написаны корректно, стек опустеет. Иначе в нем останутся незакрытые теги. Программа выводит *"correct"* если стек пустой и *"wrong"* в противном случае.

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.	<tag1>Some text more text</tag1><tag2><tag3> even more text<hr></tag3></tag2>	correct	Тест на правильных входных данных
2.	<tag1> a <tag2> b </tag1> c </tag2>	wrong	Тест на неправильных входных данных

Выводы

Были изучены динамические структуры данных. Полученные знания применены на практике: написана программа на языке C++, использующая стек для решения задачи.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#define HR_TAG "hr"
#define BR_TAG "br"
#define OPEN_TAG '<'
#define CLOSE_TAG '>'
#define CLOSING_TAG_SLASH '/'
#define TAG_OFFSET 1
#define NULL_CH_BUFFER_SIZE 1
#define CMP_OFFSET 1

const char *correct_answer = "correct";
const char *wrong_answer = "wrong";

class CustomStack {
public:
    CustomStack() {
        this->mHead = nullptr;
    }

    void push(const char *data) {
        ListNode* newNode = new ListNode;

        newNode->mData = new char[strlen(data) + NULL_CH_BUFFER_SIZE];
        strcpy(newNode->mData, data);

        newNode->mNext = this->mHead;
        this->mHead = newNode;
    }

    void pop() {
        if (!this->empty()) {
                ListNode* tmp = this->mHead;
                this->mHead = this->mHead->mNext;

                delete[] tmp->mData;
                delete tmp;
           }
    }

    size_t size() {
        ListNode* tmp = this->mHead;
        size_t counter = 0;

        while (tmp) {
            counter++;
            tmp = tmp->mNext;
        }

        return counter;
    }
```

```
        bool empty() {
            return this->mHead == nullptr;
        }

        char* top() {
            if (this->empty())
                return nullptr;

            return this->mHead->mData;
        }

        ~CustomStack() {
            while (!this->empty())
                    this->pop();
        }

    protected:
        ListNode* mHead;
    };

    int main() {
        std::string buffer;
        getline(cin, buffer);

        CustomStack stack;

        for (size_t i = 0; i < buffer.size(); i++) {
            if (buffer[i] == OPEN_TAG) {
                size_t len = 1;

                while (buffer[i + len + TAG_OFFSET] != CLOSE_TAG)
                    len++;

                stack.push(buffer.substr(i + TAG_OFFSET, len).c_str());

                char    *tag    =    new    char[strlen(stack.top())    +
NULL_CH_BUFFER_SIZE];
                    strcpy(tag, stack.top());


                if (strcmp(tag, BR_TAG) == 0 || strcmp(tag, HR_TAG) ==
0)
                    stack.pop();
                else {
                    if (stack.size() >= 2) {
                        stack.pop();

                        if (strcmp(tag + CMP_OFFSET, stack.top()) == 0
&& tag[0] == CLOSING_TAG_SLASH)
                            stack.pop();
                        else
                            stack.push(tag);
                    }
                }

                    delete[] tag;
            }
```

```
    }

    if (stack.empty())
        std::cout << correct_answer << endl;
    else
        std::cout << wrong_answer << endl;

     return 0;
}
```