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.

Студентка гр. 0382 _____ Михайлова О.Д.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Научиться работать с динамическими структурами данных на языке C++.

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:

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

Основные теоретические положения.

Стек – это структура данных, в которой хранятся элементы в виде последовательности, организованной по принципу LIFO (Last In — First Out). Такую структуру данных можно сравнить со стопкой тарелок или магазином автомата. Стек не предполагает прямого доступа к элементам и список основных операций ограничивается операциями помещения элемента в стек и извлечения элемента из стека. Их принято называть PUSH и POP соответственно. Также, обычно есть возможность посмотреть на верхний элемент стека, не извлекая его (TOP) и несколько других функций, таких как проверка на пустоту стека и некоторые другие.

Класс – это абстрактный тип данных, который может включать в себя не только данные, но и программный код в виде функций. Они реализуют в себе оба принципа, описанных выше следующим образом:

- В классе могут размещаться как данные (их называют полями), так и функции (их называют методы) для обработки этих данных.
- Любой метод или поле класса имеет свой спецификатор доступа: public, private или protected.

Выполнение работы.

В программе реализован класс CustomStack. В конструкторе класса выделяется память для массива данных char** mData. В деконструкторе происходит очистка памяти массива данных. В классе определены следующие методы:

1. void push(const char* val) - добавляет новый элемент в стек. Сначала в этом методе происходит проверка на то, достаточно ли выделенной памяти для добавления нового элемента. Если нет, то вызывается метод extend для расширения массива данных. Далее выделяется память и добавляется элемент в стек.

2. void pop() - удаляет из стека последний элемент, перед этим проверяя, не пуст ли стек.

3. char* top() - доступ к верхнему элементу. Если стек пуст, то функция возвращает нулевой указатель, если нет – указатель на верхний элемент стека.

4. size_t size() - возвращает количество элементов в стеке.

5. bool empty() - проверяет отсутствие элементов в стеке.

6. void extend(int n) — расширяет исходный массив на n ячеек, перевыделяя память для массива данных mData с помощью функции realloc.

Функция main()

С помощью функции cin.get() и цикла while посимвольно считывается входная строка. Считанный символ записывается в переменную с типа char. Если введенный символ равен “<”, то переменой int flag присваивается значение 1. Если символ не равен “<” и не равен “>” и значение переменной flag равно 1, то символы записываются в массив char* tag до тех пор, пока не будет введен символ, равный “>”. Как только последнее условие выполняется, в конец массив tag записывается символ конца строки ‘\0’. Если тег, записанный в символьный массив tag, не “hr” и не “br”, то далее сравниваются верхний элемент в стеке и строка tag, не учитывая ее первый символ. Если они равны и стек не пуст, то верхний элемент извлекается из стека. В ином случае tag добавляется в стек.

После завершения цикла while программа выводит на экран “correct”, если стек пуст, и “wrong”, если в стеке остались элементы.

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><tag2></tag2></tag1>	correct	Результат верный.
2.	<html><head><title>HTML Document</title></ head><body><p>This text is bold, this is bold and italics</ p></body></html>	correct	Результат верный.
3.	<tag1><tag2>asdfg</ tag2></tag1>	wrong	Результат верный.

Выводы.

Были изучены методы работы с динамическими структурами данных на языке C++.

Разработана программа, в которой реализован стек на базе массива. Программа принимает на вход код «простой» html-страницы и проверяет ее на валидность.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: pr_lb_4.cpp

```
#include <iostream>
#include <cstring>
using namespace std;

class CustomStack {
public:
    CustomStack(){
        mDataSize = 0;
        mDataMaxSize = 10;
        mData = (char**)malloc(mDataMaxSize*sizeof(char *));
    };

    ~CustomStack(){
        for (size_t i=0; i<mDataSize; i++){
            free(mData[i]);
        }
        free(mData);
    }

    void push(const char* val){
        if (mDataSize >= mDataMaxSize){
            extend(mDataMaxSize);
        }
        mData[mDataSize] = (char*)malloc(strlen(val)*sizeof(char));
        strcpy(mData[mDataSize], val);
        mDataSize++;
    }

    void pop(){
        if (empty()){
            return;
        }
        free(mData[mDataSize-1]);
        mDataSize--;
    }

    char* top(){
        if (empty()){
            return nullptr;
        }
        return mData[mDataSize-1];
    }

    size_t size(){
        return mDataSize;
    }

    bool empty(){
        return (mDataSize == 0);
    }

    void extend(int n){
        mDataMaxSize += n;
        mData = (char**)realloc(mData, mDataMaxSize*sizeof(char*));
```

```
    }

private:
    size_t mDataSize;
    int mDataMaxSize;

protected:
    char** mData;
};

int main() {
    CustomStack stack;
    char* tag = (char*)malloc(100*sizeof(char));
    size_t index = 0;
    size_t flag = 0;
    char c = cin.get();
    while(c != '\n'){
        if (c == '<') {
            flag = 1;
        }
        else if(c == '>') {
            tag[index] = '\0';
            index = 0;
            flag = 0;

            if (strcmp(tag, "br")!=0 && strcmp(tag, "hr")!=0){
                if (!stack.empty() && strcmp(tag+1, stack.top())==0){
                    stack.pop();
                }
                else{
                    stack.push(tag);
                }
            }
        }
        else if(flag){
            tag[index++] = c;
        }
        c = cin.get();
    }

    if (stack.empty()){
        cout << "correct" << endl;
    } else {
        cout << "wrong" << endl;
    }
    return 0;
}
```