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. 0382 _____ Тихонов С.В.

Преподаватель _____ Берленко Т.А..

Санкт-Петербург

2021

Цель работы.

Освоить работу с динамическим структурами в 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.

Язык C++ реализует объектно-ориентированную парадигму программирования, которая включает в себя реализацию механизма инкапсуляции данных. Инкапсуляция в C++ означает, что в одной языковой конструкции размещается как данные, так и функции для обработки этих данных.

Доступ к этим данным нельзя получить из вне ограничен. Пользователю предоставляется интерфейс из методов (API) с помощью которого он может влиять на состояние данных.

В я зыке C++ память выделяется при помощи функции new. Он динамически выделает память в куче. Для освобождения выделенной памяти используется функция delete.

Класс — это шаблон, по котором определяется форма объекта. В не есть данные (поля) и функции (методы) для обработки этих данных. Любой метод или поле имеет свой спецификатор доступа: public, privaete или rotceted.

Выполнение работы.

Реализация класса CustomStack:

Конструктор CustomStack() для инициализации начальных значений

Деструктор ~CustomStack()Б в котором происходит освобождение памяти.

Метод void push(const char* val) в нем происходит пере выделение памяти, если это нужно, а так же добавление элемента в двумерный массив.

Метод void pop(), в котором удаляется верхний элемент стека. Если элементов нет, то происходит выход из метода.

Метод char*top(), в котором, если стек не пуст, возвращает адрес верхнего элемента.

Метод sie_t size(), в котором возвращается количество элементов стека.

Метод bool empty(), в котором возвращаемое значение равно единице, если стек пуст, и нулю, если в стеке есть элементы.

Метод extend(), в котором происходит выделение дополнительной памяти.

В функции main() происходит посимвольное считывание и проверка тегов на валидность в цикле while. Таким образом, если стек оказывается пуст после всех проверок, то выводится сообщение о том, что теги валидны. Если это не так, выводится сообщение о том, что теги не валидны.

Тестирование.

Результаты тестирования представлены в табл.1.

Таблица 1-Результаты тестирования

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

Вывод.

Были исследованы методы работы с динамическими структурами на зыке C++, а также динамическое выделение и очищение памяти. Разработана программа, выполняющая считывание с клавиатуры кода html - страницы и проверяющая его на валидность.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

using namespace std;

class CustomStack {
public:
    explicit CustomStack(int initialSize = 10) {
        size_d = 0;
        size_max = initialSize;
        mData = new char* [size_max];
    };

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

    void push(const char* val) {
        if (size_d >= size_max) {
            extend(size_plus);
        }
        mData[size_d] = new char;
        strcpy(mData[size_d], val);
        size_d++;
    }

    void pop() {
        if (size_d == 0) {
            return;
        }
        delete [] mData[size_d-1];
        size_d--;
```

```
    }

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

    size_t size() {
        return size_d;
    }

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

    void extend(int n) {
        size_max += n;
        mData = new char* [size_max];
    }

private:
    size_t size_d;
    size_t size_max;

protected:
    const size_t size_plus = 20;
    char** mData;
};

int main() {
    CustomStack stack(100);
    char* elem = new char [100];
    size_t index = 0;
    bool flag = false;
    char c = cin.get();

    while (c != '\n') {
        if (c == '<') {
            flag = true;
```

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

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