# 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. 3343 |  | Добрякова А.А.. |
| Преподаватель |  | Государкин Я.С. |

Санкт-Петербург 2024

# Цель работы

Цель работы заключается в изучении способов реализации динамических структур данных на языке C++ и их применении для решения конкретной задачи. В рамках данной работы необходимо написать программу, которая использует одну из динамических структур данных — стек, для решения задачи валидации тегов html документа.

# 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](https://ru.wikipedia.org/wiki/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: <br>, <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.

# Пример:

*Входная строка:*

<html><head><title>HTML Document</title></head><body><p><b>This text is bold,<br><i>this is bold and italics</i></b></p></body></html>

*Результат:*

***correct***

# Выполнение работы

В рамках выполнения работы была разработана программа на языке C++, включающая в себя класс *CustomStack*, функцию *isBalanced*, предназначенную для проверки корректности html-тегов, а также функцию *main*, которая является точкой входа в программу.

## Класс CustomStack

Класс *CustomStack* реализует структуру данных стек на основе массива.

Экземпляры класса содержат защищенные поля *mData* для хранения массива отдельных тегов, *topIndex* для обращения к верхнему элементу стека и размер текущей памяти, зарезервированной под стек. Класс предоставляет следующие методы:

* *push*: добавляет новый элемент на верх стека;
* *top*: возвращает значение верхнего элемента;
* *pop*: вытаскивает значение верхнего элемента, удаляя его из стека;
* *empty*: возвращает *true*, если стек пуст, иначе *false*;
* *size*: возвращает текущее количество элементов в стеке;
* *extend*: расширяет память, выделенную под стек, на указанное количество элементов.

## Функция isBalanced

Функция *isBalanced* принимает на вход строку.

Возвращает *true* или *false* в зависимости от того, имеет ли каждый открывающий тег соответствующий ему закрывающий.

Для этого каждый открывающий тег добавляется в стек. Когда встречается закрывающий тег, проверяется, лежит ли соответствующий открывающий тег в стеке. Если его там не оказывается, то последовательность тегов не верна.

Разработанный программный код см. в приложении А.

# Тестирование

Результаты тестирования представлены в Таблице 1.

Таблица 1 – Проверка на корректность поиска

|  |  |  |  |
| --- | --- | --- | --- |
| ` | Входные данные | Выходные данные | Комментарии |
| 1. | <tag1><tag2></tag2></tag1> | correct | OK |
| 2. | <tag1><tag2></tag1></tag2> | wrong | OK |
| 3. | <html><head><title>HTML Document</title></head><bo dy><p><b>This text is bold,<br><i>this  is bold and italics</i></b></p></body>  </html> | correct | OK |

# Выводы

В результате выполнения данной работы были изучены принципы реализации динамических структур данных на языке C++ и их применение для решения задачи валидации тегов html документа. Была разработана программа, использующая стек в качестве структуры данных для проверки корректности расстановки html-тегов.

# ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: *main.cpp*

class CustomStack {

public:

CustomStack()

{

this->mData = new char\*[mDataSize];

topIndex = -1;

}

~CustomStack()

{

delete[] mData;

}

void push(const char\* val)

{

topIndex++;

if (topIndex > mDataSize) {

extend(mDataSize);

}

mData[topIndex] = new char[strlen(val) + 1];

strcpy(mData[topIndex], val);

}

char\* top()

{

return mData[topIndex];

}

char\* pop()

{

return mData[topIndex--];

}

bool empty()

{

return topIndex == -1;

}

size\_t size()

{

return topIndex + 1;

}

void extend(int n)

{

char\*\* newData = new char\*[mDataSize + n];

for (size\_t i = 0; i <= topIndex; ++i) {

newData[i] = mData[i];

}

delete[] mData;

mDataSize += n;

mData = newData;

}

protected:

char\*\* mData;

int topIndex;

size\_t mDataSize = 10;

};

bool isBalanced(string line)

{

int startIndex = line.find('<');

int endIndex = line.find('>');

CustomStack\* stack = new CustomStack;

while (startIndex != string::npos || endIndex != string::npos) {

string tag = line.substr(startIndex + 1, endIndex - startIndex - 1);

if (tag == "br" || tag == "hr") {

line = line.substr(endIndex + 1);

startIndex = line.find('<');

endIndex = line.find('>');

continue;

}

if (tag[0] != '/') {

stack->push(tag.c\_str());

line = line.substr(endIndex + 1);

startIndex = line.find('<');

endIndex = line.find('>');

continue;

}

if (stack->empty()) {

return false;

}

char\* previous\_tag = stack->pop();

if (strcmp(previous\_tag, tag.c\_str() + 1) != 0) {

return false;

}

line = line.substr(endIndex + 1);

startIndex = line.find('<');

endIndex = line.find('>');

}

}

int main()

{

string line;

getline(cin, line);

if (isBalanced(line)) {

cout << "correct";

} else {

cout << "wrong";

}

return 0;

}