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. 2384 _____ Валеева А.А.

Преподаватель _____ Гаврилов А.В.

Санкт-Петербург

2023

Цель работы

Изучить классы на языке C++, реализовать программу которая с помощью базовых методов класса проверяет на валидность поданную в виде строки html-страницу.

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. аттрибутов 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:

1. Указатель на голову 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.
5. 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

Для выполнения данной лабораторной работы включили библиотеки: *iostream, cstring, cstdlib*.

Функция *void push(const char* tag)* добавляет новый элемент в стек. Новый элемент становится на место головы списка. Описание работы: метод принимает на вход указатель на массив *arr*, который содержит данные для добавляемого элемента. Сначала создается новый элемент стека *ListNode*, для которого выделяется память с помощью оператора *new*. Затем в поле *mData* этого элемента копируется содержимое массива *arr* с помощью функции *strcpy()*. Далее указатель на следующий элемент *mNext* нового элемента устанавливается на текущую вершину стека *mHead*, а затем указатель на вершину стека *mHead* перенаправляется на новый элемент, таким образом, новый элемент добавляется в начало стека.

После выполнения метода *push()* стек содержит новый элемент с данными *arr* в вершине.

Функция *void pop()* удаляет из стека последний элемент. Описание работы: сначала проверяется, если стек пустой, то выводится ***wrong*** и программа завершает работу. Иначе указатель на вершину стека *mHead* переходит на следующий элемент, а затем текущий элемент удаляется из памяти с помощью оператора *delete*. Также из памяти удаляется массив символов *mData*, содержащий данные элемента.

После выполнения метода *pop()* стек содержит на вершине следующий элемент.

Функция *char* top()* возвращает данные, хранящиеся в поле *mData* головы списка. Если стек не имеет элементов, выводится ***wrong***, иначе

выводится содержимое *mData*.

После выполнения метода *top()* стек остается неизменным.

Функция *size_t size()* возвращает количество элементов стека. Если стек пуст, то метод возвращает 0. Метод проходит по элементам стека, начиная с вершины и увеличивая счетчик *size_1* на 1 на каждом элементе. Цикл продолжается до тех пор, пока не будет достигнут конец стека (*mNext = nullptr*).

Функция *bool empty()* проверяет стек на пустоту. Если функция *size()* возвращает 0, то данная функция возвращает *true*. Иначе – *false*. Далее объявляется *CustomStack stack*.

Считывание производится в переменную *text* с помощью функции *fgets()*. Потом происходит поиск слов, заключенных в «<>». Слово запоминается в переменную *buff*. Если первый символ не является «/», то слово добавляется в стек *stack.push(buff)*. Иначе – сравнивается содержимое *buff+1* (т.е. так мы сравниваем слово без «/») и *stack.top()*. Если слова сходятся, то удаляем элемент стека *stack.pop()*. Иначе выводим ***wrong*** и завершаем работу. Когда алгоритм проверит всю строку, выполняем проверку стека на пустоту *stack.empty()*. Если пустой – то выводим ***correct***. Иначе - ***wrong***.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

Изучен принцип работы с классами на языке C++. Реализована программа, выполняющая действия над введённой строкой с помощью методов класса. Выполнены функции на добавление, удаление и подсчёт элементов в стеке.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#define N 3000
#define BUF_SIZE 30

class CustomStack {
public:
    // конструктор
    CustomStack() {
        mHead = nullptr;
    }
    // деструктор
    ~CustomStack() {
        ListNode *cur = mHead;
        ListNode *before;
        while (cur != nullptr) {
            before = cur->mNext;
            delete cur;
            cur = before;
        }
    }
```

```
void push(const char* arr){
    auto * new_el = new ListNode;
    new_el -> mData = new char[strlen(arr)];
    strcpy(new_el -> mData,arr);
    new_el -> mNext = mHead;
    mHead = new_el;
}

void pop(){
    if(!size()){
        cout << "wrong" << endl;
        exit(0);
    }

    ListNode* tmp = mHead;
    delete mHead -> mData;
    mHead = mHead -> mNext;
    delete tmp;
}

char* top(){
    if(!size()){
        cout << "wrong" << endl;
        exit(0);
    }
    return mHead -> mData;
```

```
    }

    size_t size(){
        size_t size_1 = 0;
        ListNode* tmp = mHead;

        while(tmp != nullptr){
            size_1++;
            tmp = tmp -> mNext;
        }
        return size_1;
    }

    bool empty(){
        return size() == 0;
    }
protected:
    ListNode* mHead;
};

int main() {
    char* text = new char[N];
    fgets(text, N, stdin);
    CustomStack stack;
    int i = 0;
    int help = 0;
```

```
while(help <= strlen(text)){

   if(text[help] == '<'){
      char* buff = new char[BUF_SIZE];

      while(text[help] != '>'){
         i++;
         help++;
      }
      strncpy(buff, text+help - i +1, i-1);
      if(strcmp(buff, "br") != 0 && strcmp(buff, "hr") != 0){
         if(buff[0]!='/'){
            stack.push(buff);
         } else{
            if(!strcmp(stack.top(), buff+1)) stack.pop();
            else{
               cout << "wrong" << endl;
               return 0;
            }
         }
      }
      i = 0;
   }
   help++;
}

if (stack.empty()){
```

```
        cout << "correct" << endl;
    }
    else{
        cout << "wrong" << endl;
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ Б.

ТЕСТИРОВАНИЕ

Таблица Б.1 – результаты тестирования

№ п/п	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	Верный ответ.