МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Программирование»
Тема: ВВЕДЕНИЕ В ЯЗЫК C++

Студент гр. 3341 _____ Костромитин М.М

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью работы является изучение основных механизмов языка C++ путем разработки структур данных стека и очереди на основе динамической памяти.

Для достижения поставленной цели требуется решить следующие задачи:

– ознакомиться со структурами данных стека и очереди, особенностями их реализации;

– изучить и использовать базовые механизмы языка C++, необходимые для реализации стека и очереди;

– реализовать индивидуальный вариант стека в виде C++ класса, его операции в виде функций этого класса, ввод и вывод данных программы.

Задание

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:

Указатель на массив 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.

Выполнение работы

Ниже представлен ход выполнения лабораторной работы.

1. Объявим класс CustomStack с указанными методами и полями.

2. Опишем конструктор CustomStack и метод расширения стека extend, который увеличивает емкость стека на заданное количество элементов.

3. Реализуем метод push, который добавляет новый элемент в стек, и метод pop, который удаляет последний элемент из стека.

4. Реализуем метод top, который позволяет получить доступ к верхнему элементу стека.

5. Реализуем метод size, который возвращает количество элементов в стеке, и метод empty, который проверяет, пуст ли стек.

6. Опишем деструктор CustomStack, который освобождает память, выделенную под элементы стека.

7. Напишем функцию isValidHtml для проверки валидности html-страницы в соответствии с заданными условиями, в которая работает по следующему принципу:

7.1. Просмотрим входную строку посимвольно.

7.2. Если встречаем открывающий тег, добавляем его в стек.

7.3. Если встречаем закрывающий тег, проверяем его на соответствие последнему открывающему тегу в стеке. Если теги совпадают, удаляем из стека открывающий тег.

7.4. В конце проверяем, остались ли открытые теги в стеке. Если да, то функция возвращает 0.

8. В функции main создадим экземпляр класса CustomStack и передадим в нее строку с html-кодом. После вызова функции isValidHtml выведем результат проверки на экран.

Код программы – см. Приложение А.

Тестирование

Результаты тестирования представлены в табл. 1

Табл. 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	Тест с moevm

Выводы

В ходе работы были изучены и применены основные механизмы языка C++ для разработки структур данных стека и очереди на основе динамической памяти. Реализован класс CustomStack для работы со стеком, включающий операции push, pop, top, size, empty и метод extend. Был также разработан алгоритм проверки валидности HTML-страницы с помощью стека. Этот опыт позволил глубже понять принципы работы стека, освоить базовые механизмы C++ и их применение для создания сложных структур данных.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
bool isValidHtml(char* html);
void gettag(char* input, char* dest);

class CustomStack {
public:
    CustomStack() {
        mData = new char *[10];
        mCapacity = 10;
        mSize = 0;
    }

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

    void
    push(const char *val) {
        if (mSize >= mCapacity) {
            extend(10);
        }

        mData[mSize] = new char[strlen(val) + 1];
        strcpy(mData[mSize], val);
        mSize++;
    }

    void
    pop() {
        if (mSize > 0) {
            delete[] mData[mSize - 1];
            mSize--;
        }
    }

    char *
    top() {
        if (mSize > 0) {
            return mData[mSize - 1];
        }
        return nullptr;
    }

    size_t
    size(){
        return mSize;
    }
```

```
    bool
    empty(){
        return (mSize == 0);
    }

    void
    extend(int n){
        mCapacity += n;
        char** tmp = new char*[mCapacity];
        for (int i = 0; i < mSize; i++){
            tmp[i] = mData[i];
        }
        delete[] mData;
        mData = tmp;
    }

private:
    int mSize;
    int mCapacity;

protected:
    char** mData;
};

int main() {
    char* input = new char[3001];
    fgets(input, 3000, stdin);
    if (isValidHtml(input)){
        cout << "correct";
    } else {
        cout << "wrong";
    }
    return 0;
}


void
gettag(char* input, char* dest){
    int size = 0;
    for (int i = 0; *input != '>'; input++ and i++){
        dest[i] = *input;
        size++;
    }
    dest[size] = '\0';
}

bool
isValidHtml(char* html) {
    CustomStack stack;
    char* tag = nullptr;
    char tmp[3001];
    while (*html) {
        if (*html == '<') {
            if (*(html + 1) == '/') {
                html++; // skip '/'
                html++; // move to tag name
                tag = stack.top();
                gettag(html, tmp);
```

```
                if (tag && strcmp(tag, tmp) == 0) {
                    stack.pop();
                } else {
                    return false;
                }
            } else {
                html++; // move to tag name
                gettag(html, tmp);
                if (strcmp(tmp, "br") and strcmp(tmp, "hr")) {
                    stack.push(tmp);
                }
            }
        }

        html++;
    }

    return stack.empty();
}
```