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.
Тестирование. C++ intro.

Студент гр. 1304 _____ Кривоченко Д. И.

Преподаватель _____ Чайка К. В.

Санкт-Петербург
2022

Цель работы.

Научиться работать с классами в C++. Реализовать класс, отвечающий условиям на языке C++.

Задание.

Вариант 5. 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*.

Выполнение работы.

Был реализован класс стека CustomStack на основе связного списка, а также стандартные методы, присущие стеку (*push()*, *pop()*, *top()*, *size()*, *empty()*). Помимо стандартных методов был написан приватный метод *tag_cmp(char* str1*, *char* str2)*, сравнивающий теги между собой. Также по заданию реализован метод *text_processing(char* text)*, принимающий на вход набор тегов и проверяющий строку на правильность (по заданию).

Выводы.

Написан класс на C++, проверяющий набор тегов в строке на корректность. Научились работать с классами в C++, повторили реализацию стека на основе связного списка.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
class CustomStack {
public:
    CustomStack() {
        mHead = (ListNode*)calloc(1, sizeof(ListNode));
    }
    bool empty() {
        return (size_count == 0);
    }
    size_t size() {
        return size_count;
    }

    ListNode* createNode(const char* data, ListNode* next) {
                    ListNode*  cur  =  (ListNode*)calloc(1,
sizeof(ListNode));
        char* dummy_data = (char*)malloc(strlen(data) + 1);
        strcpy(dummy_data, data);
        cur->mData = dummy_data;
        cur->mNext = next;
        return cur;
    }

    void push(const char* tag) {
        ListNode* tmp = mHead;
        ListNode* toPush = createNode(tag, NULL);

        if (mHead->mData == NULL) {
            mHead = toPush;
            size_count++;
            return;
```

```
    }

    else {
        while (tmp->mNext != NULL) {
            tmp = tmp->mNext;
        }
        tmp->mNext = toPush;
        size_count++;
    }
}


void print_list() {
    cout << "[ ";
    //  cout << mHead->mData;
    ListNode* tmp = mHead;
    while (tmp != NULL) {
        cout << tmp->mData << " ";
        tmp = tmp->mNext;
    }
    cout << "]\n";
}




char* top() {
    ListNode* tmp = mHead;

    if ((tmp->mData) == NULL) {
        cout << "error" << endl;
        exit(0);
    }
    else {
```

```
        while (tmp->mNext) {
            tmp = tmp->mNext;
        }
        return tmp->mData;
    }
}
void pop() {

    ListNode* tmp = mHead;
    if ((tmp->mData) == NULL) {
        cout << "error";
        exit(0);
    }
    if ((tmp->mNext) == NULL) {
        mHead = NULL;
        size_count--;
        return;
    }

    while (tmp->mNext->mNext) {
        tmp = tmp->mNext;
    }
    char dummy_str[1000];
    strcpy(dummy_str, tmp->mNext->mData);
    free(tmp->mNext);
    tmp->mNext = NULL;
    size_count--;

}



void text_processing(char* text) {
```

```
                //        char* tag_txt =(char*) calloc(3000,
sizeof(char));  <html> </html>
            int len = strlen(text);
            bool startRead = 0;
            char* dummy_str = NULL;
            int dummy_iter = 0;
            dummy_str = (char*)calloc(3000, sizeof(char));
            for (int i = 0; i < len; i++) {
                if ((text[i] == '<') && (!startRead)) {
                    startRead = 1;
                    dummy_str[dummy_iter++] = text[i];

                }
                  else if (((text[i] == '<') && (startRead)) ||
((text[i] == '>') && (startRead == 0))) {
                    cout << "error";
                    exit(0);
                }
                else if ((text[i] == '>') && (startRead)) {
                    startRead = 0;

                    dummy_str[dummy_iter++] = text[i];
                    dummy_str[dummy_iter++] = '|';
                    //      dummy_str[dummy_iter] = '\0';
                    //      dummy_iter = 0;
                    //      if ((strcmp(dummy_str, "<br>") != 0)
&& (strcmp(dummy_str, "<hr>") != 0)) {
                    //          push(dummy_str);
                    //          free(dummy_str);
                     //     }
                }
                else if (startRead) {
                    dummy_str[dummy_iter++] = text[i];
                }
```

```
            }
            dummy_str[dummy_iter] = '\0';
            //   cout << dummy_str<<endl;

        //     cout << dummy_str<<endl;
            char* pch = strtok(dummy_str, "|");
            char* to_cmp = NULL;
            char* inv_tag =(char*) calloc(100, sizeof(char));
            while (pch != NULL)
            {
                 if ((strcmp(pch, "<br>") != 0) && (strcmp(pch,
"<hr>") != 0)) {
                    if (strchr(pch, '/') != 0) {
                        if ((tag_cmp(top(), pch)) != true) {
                            cout << "wrong" << endl;
                            exit(0);
                        }
                        else {
                            pop();
                        }

                    }
                    else {
                        push(pch);
                    }

                }
                pch = strtok(NULL, "|");
            }
            if (!empty()) {
                cout << "error";
                exit(0);
              }
```

```
    else {
        cout << "correct";
        exit(0);
    }
}

protected:
    int size_count = 0;
    ListNode* mHead;


private:
    bool tag_cmp(char* str1, char* str2) {
        if ((strlen(str2) - strlen(str1)) == 1) {
            char dummy_str[100];
            int i_d = 0;
            int i = 0;
            while (i < strlen(str2)) {
                if (str2[i] != '/') {
                    dummy_str[i_d++] = str2[i++];
                }
                else {
                    dummy_str[i_d] = str2[i++];
                }
            }
            dummy_str[i_d] = '\0';
            // cout << str1 << "|" << dummy_str << endl;
            if (strcmp(str1, dummy_str) == 0) {
                return true;
            }
        }
        return false;
    }
```

```
};


// <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>

int main() {
    CustomStack a;
    char txt[3001];
    fgets(txt, 3001, stdin);
    if (txt[strlen(txt) - 1] == '\n')
        txt[strlen(txt) - 1] == '\0';
    a.text_processing(txt);



    return 0;
}
```