Лабораторная работа №5
Студент:
Круглов Артем Евгеньевич

Отчет

Список файлов

```
lab5.cpp
engine.cpp
f_add.cpp
tests.cpp
dop.cpp
engine.h
f_add.h
tests.h
dop.h
Makefile
input1
output1
input2
output2
input3
output3
```

lab5.cpp

Основной файл проекта с функцией int main().

```
#include <iostream>
#include <cstring>
#include <fstream>
#include "f_add.h"
#include "engine.h"
#include "tests.h"

using namespace std;

int main(int argc, char** argv) {

    runAllTests();
    bool fromFile = false;
    bool toFile = false;
    char* fromFileName = new char[1024];
    char* toFileName = new char[1024];
    if(argc > 1) {
        if (strcmp(argv[1], "--fromFile") == 0) {
            if (argc == 2) {
```

```
                cout << "Не хватает параметров" << endl;
                return 1;
            }
            fromFile = true;
            fromFileName = argv[2];
        }
        if (strcmp(argv[1], "--toFile") == 0) {
            if (argc == 2) {
                cout << "Не хватает параметров" << endl;
                return 1;
            }
            toFile = true;
            toFileName = argv[2];
        }
        if(argc > 3) {
            if (strcmp(argv[3], "--fromFile") == 0) {
                if (argc == 4) {
                    cout << "Не хватает параметров" << endl;
                    return 1;
                }
                if (fromFile) {
                    cout << "Два раза --fromFile" << endl;
                    return 1;
                }
                fromFile = true;
                fromFileName = argv[4];
            }
            if (strcmp(argv[3], "--toFile") == 0) {
                if (argc == 4) {
                    cout << "Не хватает параметров" << endl;
                    return 1;
                }
                if (toFile) {
                    cout << "Два раза --toFile" << endl;
                    return 1;
                }
                toFile = true;
                toFileName = argv[4];
            }

        }
    }
    if (!fromFile) {
        ofstream f1;
        f1.open("tmpFile");
        char* s = new char[1024];
        if (!f1.is_open()) {
            cout << "Проблемы с открытием файла" << endl;
            return 1;
        }
        cout << "Вводите предложения. После окончания текста,
введите quit" << endl;
```

```
            while (true) {
                cin >> s;
                if (strcmp(s,"quit") == 0) {
                    break;
                }
            f1 << s << endl;
            }
            f1.close();
            strcpy(fromFileName, "tmpFile");

        }
        doIt(fromFileName, toFile, toFileName);

        return 0;
    }
```

Описание:

Перед выполнением основного кода запускается тест, проверяющий корректность работы программы на 3-х предзаданных файлах с заранее подготовленными правильными результатами.

Далее производится разбор параметров, переданных в консоле. Учитывается возможность того, что опции `--fromFile` и `--toFile` могут идти в произвольном порядке. Обрабатываются в том числе исключения, когда опция присутствует, но само имя файла не указано.

После разбора параметров выполняется ввод текста с консоли (в том случае, если параметр --fromFile не выбран). Поскольку текст может содержать более одной строки, предусмотрен ввод слова «quit», означающий конец вводимого текста. Вводимый текст пишется во временный файл с именем «tmpFile».

Дальше запускается основная функций проекта `doIt(fromFileName, toFile, toFileName)`, которой передаются в качестве параметров имя входного файла, флаг надо ли результаты писать в файл и имя выходного файла.

engine.cpp

Файл с основной функцией проекта, выполняющей разбор текста и вывод результата

Список функций:

```
void doIt(char* fromFileName, bool toFile, char* toFileName);
```

Функция, получающая текст из входного файла с именем `fromFileName` и в зависимости от флага `toFile` выводящая результат или в консоль или в файл с именем `toFileName`. Функция выполняет в том числе дополнительную задачу из лабораторной работы — вывод самого длинного предложения с обратным порядком слов в файл `Nine.txt`.

```
#include <iostream>
```

```
#include <cstring>
#include <fstream>
#include "f_add.h"
#include "dop.h"

using namespace std;

void doIt(char* fromFileName, bool toFile, char* toFileName)
{
    ifstream f;
    f.open(fromFileName);
    if (!f) {
        cout << "Could not open file - " << fromFileName <<
endl;
        return void();
    }
    int n = 100;
    int k = 100;
    char ch;
    char* s = new char[n];
    s[0] = '\0';
    char** arr = new char*[k];
    arr[0] = NULL;
    while(f.get(ch)) {
        if (ch == '\n') {
            ch = ' ';
        }
        s = addchar(s, ch, &n);
        if (ch == '.') {
            arr = addstring(arr, s, &k);
            n = 100;
            s = new char[n];
            s[0] = '\0';
        }
    }
    for (int i = 0; i < mystrlen(arr); i++) {
        if (arr[i][0] == ' ') {
            arr[i] ++;
        }
    }
    sort(arr);
    char* longest = reverse_words(arr[mystrlen(arr)-1]);
    ofstream fn("Nine.txt");
    fn << longest;
    fn.close();
    if (toFile) {
        to_file(arr, toFileName);
        } else {
            print(arr);
            }

}
```

Описание:

Открываем файл на чтение. Обрабатываем исключение, когда файл по каким-то причинам не открылся.

В переменной `char* s` будем накапливать текущее предложение из файла. В переменной `char** arr` будем хранить массив указателей на предложения. Поскольку и длина предложения и количество предложений могут быть большими, предусмотрен следующий вариант выделения памяти и добавления очередного символа к существующим: Первоначально выделяется память под 100 символов (включая последний символ с кодом `\0`). Дальше читаем символы из файла и добавляем их к строке s с помощью функции `addchar` из модуля `f_add.cpp`. Эта функция сравнивает длину строки с размером выделенной под нее памяти. Если очередной символ уже не влезает (размер выделенной памяти равен длине строки), функция выделяет большую память, копирует туда строку и уже после этого добавляет очередной символ. Поскольку для этого нужно знать размер выделенной памяти, мы передаем его в функцию по ссылке (передаем `&n`) для того, чтобы если он изменится это отразилось бы на переменной `n`. Точно такой же алгоритм предусмотрен для самого массива указателей.

При чтении символов из файла заменяем символ `\n` на пробел. В случае получения точки добавляем строку к массиву, выделяем память под новую строку, ставим первым символом `\0`, чтобы строка была пустой.

После того, как файл полностью прочитан, удаляем первые пробелы в каждом предложении, если они там есть.

Сортируем массив с помощью функции `sort(char** arr)` из модуля `f_add.cpp`

Переворачиваем самую длинную строку с помощью функции `reverse` из модуля `dop.cpp`. Записываем ее в файл `Nine.txt`. В задании не указано, что делать с точкой в конце предложения. Я оставил ее как часть последнего слова. При необходимости ее легко убрать.

В зависимости от флага `toFile` выводим результат или в консоль функцией `print(arr)` или в файл функцией `to_file(arr, toFileName)`.

f_add.cpp

Модуль, содержащий вспомогательные функции для работы основной функции

Список функций:

```
char* addchar(char* s, char ch, int* n);
char** addstring(char** arr, char* s, int* k);
int mystrlen(char** arr);
char** mystrcpy(char** arrnew, char** arr);
char** sort(char** arr);
void print(char** arr);
void to_file(char** arr, char* toFileName);
```

mystrlen(char** arr):

```
int mystrlen(char** arr) {
    int i = 0;
    while (arr[i] != NULL) {
        i++;
    }
    return i;
}
```

Функция возвращает длину массива `char**`. Длиной считаем число ссылок до первого `NULL`.

mystrcpy(char** arrnew, char** arr)

```
char** mystrcpy(char** arrnew, char** arr) {
    int i =0;
    while (arr[i] != NULL) {
        arrnew[i] = arr[i];
        i++;
    }
    arrnew[i] = NULL;
    return arrnew;

}
```

Функция, аналогичная `strcpy()`, копирующая один массив `char**` в другой.

addchar(char* s, char ch, int* n)

```
char* addchar(char* s, char ch, int* n) {
    if (strlen(s) == *n - 1) {
        *n += 100;
        char* snew = new char[*n];
        strcpy(snew, s);
        int x = strlen(s);
        delete s;
        snew[x] = ch;
        snew[x+1] = '\0';
        return snew;
    }
    int x = strlen(s);
    s[x] = ch;
    s[x+1] = '\0';
    return s;
}
```

Функция добавляет символ к строке, учитывая размер выделенной под строку памяти. Выделенная память передается в переменной `n`. Если длина строки равна `n-1` (еще один символ нужен для хранения последнего нулевого символа), я выделяю большую память под строку, копирую туда то, что есть в исходной строке и добавляю новый символ. Функция возвращает ссылку на новую строку.

addstring(char** arr, char* s, int* k)

```
char** addstring(char** arr, char* s, int* k) {
    if (mystrlen(arr) == *k - 1) {
        *k += 100;
        char** arrnew = new char*[*k];
        arrnew = mystrcpy(arrnew, arr);
        delete arr;
        arr = NULL;
        int x = mystrlen(arr);
        arrnew[x] = s;
        arrnew[x+1] = NULL;
        return arrnew;
    }
    int x = mystrlen(arr);
    arr[x] = s;
    arr[x+1] = NULL;
    return arr;
}
```

Функция добавляет строку к массиву. Аналогично предыдущей функции отрабатывается случай, когда выделенная память закончилась. В этом случае выделяем больший объем, копируем туда текущий массив и уже туда добавляем ссылку на новое предложение.

sort(char** arr)

```
void sort(char** arr) {
    for (int i = 0; i < mystrlen(arr); i++) {
        for(int q = 0; q < mystrlen(arr)-1; q++) {
            if (strlen(arr[q]) > strlen(arr[q+1])) {
                char* t = arr[q];
                arr[q] = arr[q+1];
                arr[q+1] = t;
            }
        }
    }

}
```

Сортировка пузырьковым методом.

print(char** arr)

```
void print(char** arr) {
    for(int i = 0; i < mystrlen(arr); i++) {
        cout << arr[i] << endl;
    }
}
```

Функция выводит содержимое массива в консоль

to_file(char** arr, char* toFileName)

```
void to_file(char** arr, char* toFileName) {
    ofstream f3(toFileName);
    for(int i = 0; i < mystrlen(arr); i++) {
        f3 << arr[i];
        if (i != mystrlen(arr)-1) {
            f3 << endl;
        }
    }
    f3.close();
}
```

Функция выводит содержимое массива в файл с заданным именем.

dop.cpp

Модуль с выполнением дополнительного задания.

Список функций:

```
char* AddStr(char* s1, char* s2, int n)
char* reverse_words(char* str);
```

AddStr(char* s1, char* s2, int n)

```
char* AddStr(char* s1, char* s2, int n) {
    int k1, k2;
    k1 = strlen(s1);
    k2 = strlen(s2);
    char* sre = new char[k1 + k2 + 2];
    strncpy(sre, s2, n);
    sre[n] = '\0';
    strcat(sre, " ");
    sre[n+1] = '\0';
    strcat(sre, s1);
```

```
    sre[k1 + k2 + 1] = '\0';
    return sre;
}
```

Функция добавляет к началу строки `s1` `n` символов строки `s2`, добавляя между ними пробел. Выделяю память под новую строку в нужном размере (+1 на пробел и +1 на последний `\0`).

reverse_words(char* strch)

```
char* reverse_words(char* strch) {
    char* output = new char[strlen(strch) + 1];
    output[0] = '\0';
    int i = 0;
    while (i < strlen(strch)) {
        if (strch[i] != ' ') {
            i++;
            continue;
        }
        output = AddStr(output, strch, i);
        strch += i + 1;
        i = 0;
    }
    output = AddStr(output, strch, strlen(strch));
    return output;
}
```

Функция разворачивает предложение так, как нужно по дополнительному заданию. Читаю символы из предложения по очереди, в случае обнаружения пробела добавляю в `output` нужное число символов строки до пробела. После этого удаляю из строки символы с пробелом, просто увеличивая указатель на `i+1`

tests.cpp

Модуль с тестами

Перечень функций:

```
bool cmpFiles(char* fileName1, char* fileName2)
bool runTest(char* sourceFile, char* rightResultFile)
void runAllTests()
```

cmpFiles(char* fileName1, char* fileName2)

```
bool cmpFiles(char* fileName1, char* fileName2) {
    bool cmp = true;
    ifstream f1(fileName1);
```

```
    ifstream f2(fileName2);
    char ch1;
    char ch2;
    while (f1.get(ch1)) {

        if (!f2.get(ch2)) {
            cmp = false;
            break;
        }
        if (ch1 != ch2) {
            cmp = false;
            break;
        }
    }
    if (f2.get(ch2)) {
        cmp = false;
    }
    f1.close();
    f2.close();

    return cmp;
}
```

Функция сравнивает два файла, читая из каждого по байту. Возвращает true если файлы совпадают, false если нет.

runTest(char* sourceFile, char* rightResultFile)

```
bool runTest(char* sourceFile, char* rightResultFile) {
    char* tmpFileName = new char[100];
    strcpy(tmpFileName, "tmp_file");
    doIt(sourceFile, true, tmpFileName);
    if (cmpFiles(tmpFileName, rightResultFile)) {
        return true;
    }
    return false;
}
```

Функция проверяет работу программы для входном файле `sourceFile`, сравнивая результат с эталонным в файле `rightResultFile`.

runAllTests()

```
void runAllTests() {
    char* source = new char[1024];
    char* dest = new char[1024];
    strcpy(source, "input1");
    strcpy(dest, "output1");
```

```
    if (runTest(source, dest)) {
        cout << "Test 1 passed" << endl;
    } else {
        cout << "Test 1 failed" << endl;
    }
    strcpy(source, "input2");
    strcpy(dest, "output2");
    if (runTest(source, dest)) {
        cout << "Test 2 passed" << endl;
    } else {
        cout << "Test 2 failed" << endl;
    }
    strcpy(source, "input3");
    strcpy(dest, "output3");
    if (runTest(source, dest)) {
        cout << "Test 3 passed" << endl;
    } else {
        cout << "Test 3 failed" << endl;
    }
}
```

Функция запускает три заранее подготовленных теста, включая пример, приведенный в задании к Лабе.