**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**КАФЕДРА САПР**

отчет

**по лабораторной работе №1**

**по дисциплине «Программирование»**

Тема: **«Представление строки символов с маркером, ее обработка»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1335 |  | Максимов Ю.Е. |
| Преподаватель |  | Калмычков В.А. |

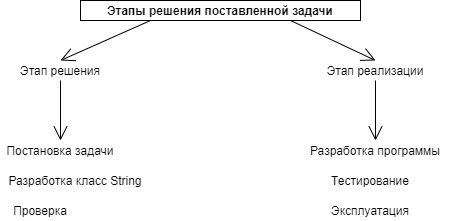
Санкт-Петербург

2022

Исходная формулировка задания

Заданы строка. Заменить последовательности пробелов в заданной строке одним пробелом.

1. Постановка задачи:
2. Разработать способ хранения строки символов с маркером
3. Организовать в памяти динамическое размещение данных в виде массива символов
4. Обеспечить эффективное выполнение операция по преобразованию строки
5. Реализовать чтение строки из файла любого размера



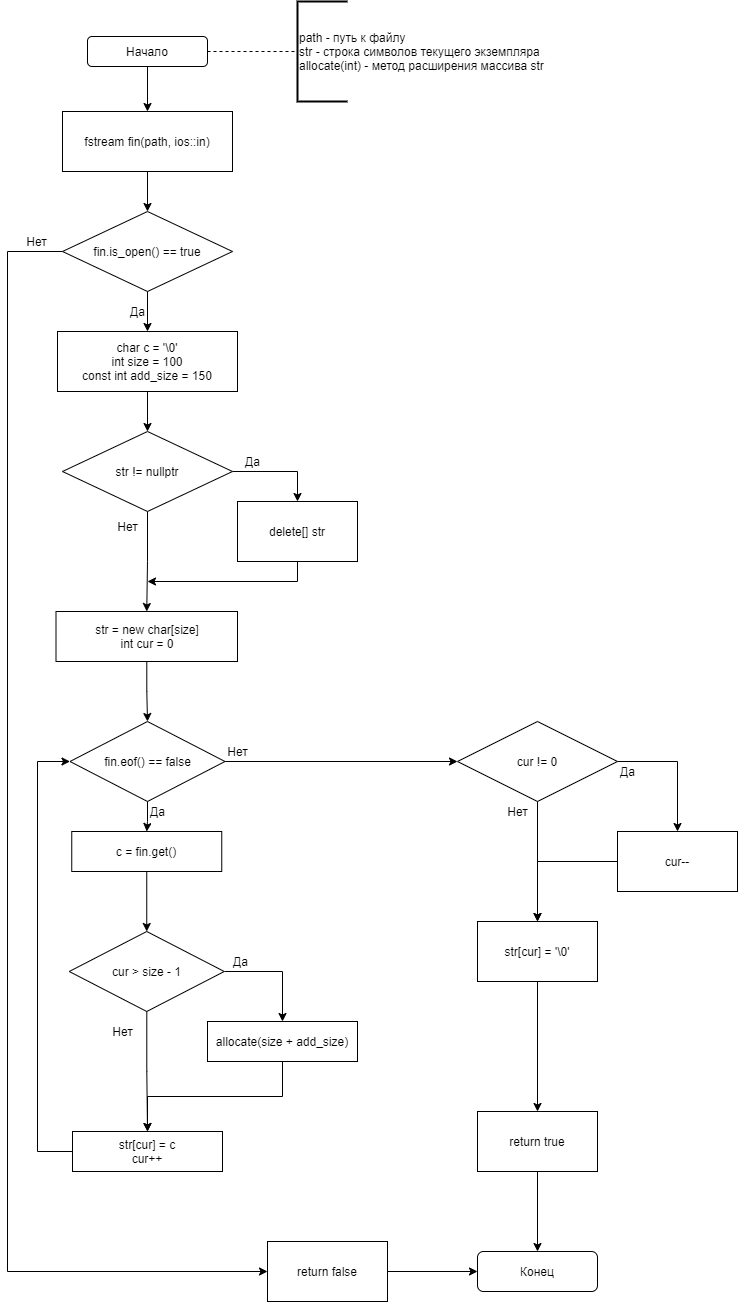
1. Анализ вариантов хранения и обработки информации.

Так как целью является реализация задачи представления строки символов с маркером, то вариант хранения элементов строки в статическом массиве, как мне кажется, уже будет являться неправильным решение. Наиболее удобным с точки зрения пользователя и программиста будет представить строку массивом символов char динамического размера.

Задача: реализовать класс String, представляющий строку символов с маркером в виду динамического массива символов char.

1. Описание методов.
2. Данные из файла произвольного размера, записанные в произвольном порядке можно преобразовать к строке с помощью встроенного в класс метода bool String::read\_data\_from\_file(const char\* path). Метод вернет результат чтения файла: *true* – если файл был прочитан, *false* – ошибка чтения файла.

Так как файлы зачастую имеют большой размер, начальный размер массива сделаем равным 100, а расширять его методом allocate() будем на 150 символов. Такое распределение позволит читать небольшие файлы без перераспределения памяти, а большие с минимальными перераспределениями.



1. Конструкторы класса String:

* Конструктор по умолчанию. Указателю присвоим значение nullptr. В дальнейшем будем проверять строку на пустоту. Конструктор позволит инициализировать экземпляр класса, но присвоить значение позднее.
* Конструктор с параметром - String::String(const char\* str). В теле конструктора вызовем метод void String::set(const char\* str). Строке экземпляра скопируем значение массива, переданного в качестве параметра.
* Конструктор копирования - String::String(const String& other). Аналогичен конструктору с параметром, за исключением того, что в метод set передаем массив другого экземпляра.

1. Метод вывода. Для вывода строки в консоль перегрузим оператор вывода:

ostream& operator<<(ostream& out, const String& obj) {

if (obj.str != nullptr)

out << obj.str << endl;

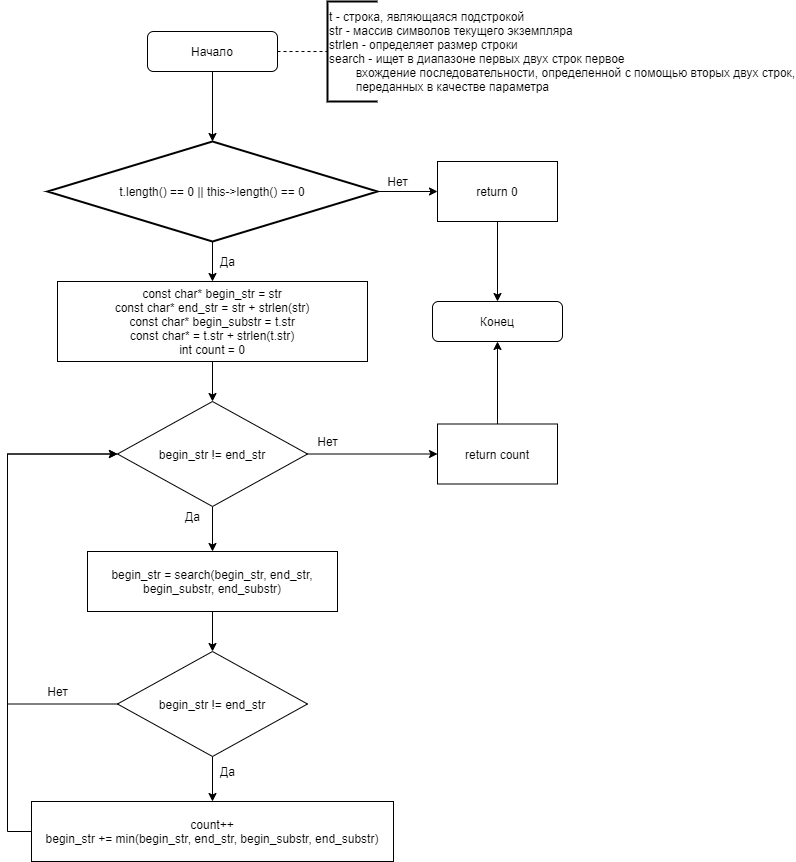
return out;

}

1. Метод пользовательского ввода. Для удобства пользовательского ввода создадим метод ввода строки. Фиксированная длина строки = 30, если в цикле у массива заканчивается кол-во свободных элементов, то «расширяем» массив, путем копирования его значений другому массиву с большим размером. В конце ввода в последний символ строки добавим нуль-символ.
2. Метод поиска минимального смещения. Для поиска очередного вхождения строки необходимо начинать поиск, смещаясь на конечную позицию найденного подстроки. Для определения того самого сдвига опишем метод:

int String::min(const char\* b1, const char\* e1, const char\* b2, const char\* e2)

1. Метод поиска количества вхождений одной строки (переданной в качестве параметра) в другой (текущем экземпляре класса).



1. Программа на C++:

Файл Source.cpp:

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

class MyString{

public:

MyString() {

str = nullptr;

length = 0;

}

MyString(const char\* str){

length = strlen(str);

this->str = new char[length + 1];

for (int i = 0; i < length; i++) {

this->str[i] = str[i];

}

this->str[length] = '\0';

}

~MyString(){

delete[] this->str;

}

MyString(const MyString& other){

length = strlen(other.str);

this->str = new char[length + 1];

for (int i = 0; i < length; i++){

this->str[i] = other.str[i];

}

this->str[length] = '\0';

}

MyString& operator =(const string& other){

if (this->str != nullptr){

delete[] str;

}

length = other.length();

this->str = new char[length + 1];

for (int i = 0; i < length; i++){

this->str[i] = other[i];

}

this->str[length] = '\0';

return \*this;

}

int Length(){

return length;

}

char& operator [](int index){

return this->str[index];

}

friend MyString erase( int elem, MyString& other) {

for (; elem < other.length; elem++) {

other.str[elem + 1] = other.str[elem + 2];

}

other.length--;

MyString newOther;

newOther.length = other.length;

newOther.str = new char[other.length];

for (int a = 0; a < other.length; a++) {

newOther[a] = other[a];

}

return newOther;

}

friend std::ostream& operator<< (std::ostream& out, const MyString& other) {

for (int a = 0; a < other.length; a++) {

out << other.str[a];

}

return out;

}

private:

char\* str;

int length;

char Mark = '@';

};

int main(){

MyString x ;

fstream fs;

fs.open("myFile.txt", fstream::out | fstream::in);

if (!fs.is\_open()) {

cout << "File 1 opening error!" << endl;

}

else {

cout << "File 1 open" << endl;

string z;

while(!fs.eof()) {

z = "";

getline(fs, z);

}

x = z;

for (int i = 0; i < x.Length(); i++)

{

if (x[i] == ' ')

while (x[i + 1] == ' ')

erase(i, x);

}

}

fs.close();

fs.open("myFile1.txt", fstream::in | fstream::out | fstream::app);

if (!fs.is\_open()) {

cout << "File 2 opening error!" << endl;

}

else {

cout << "File 2 open" << endl;

fs << x;

}

fs.close();

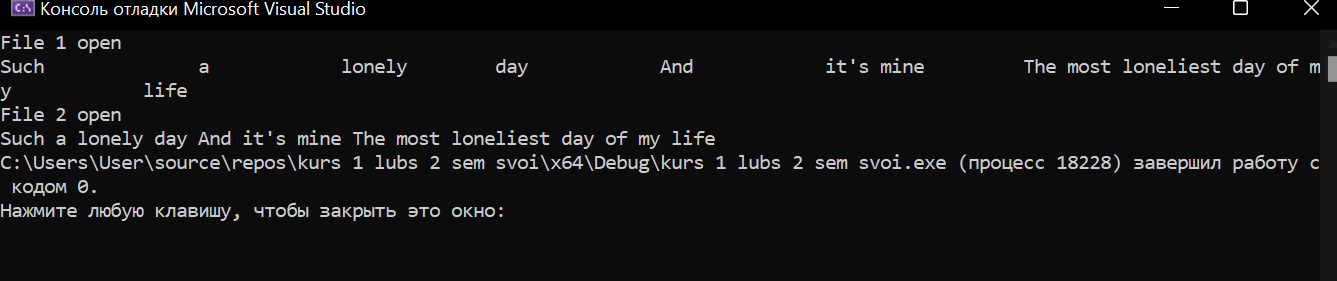
return 0;

}

1. Тестирование
2. Файл строк *myFile.txt*

*«Such a lonely day And it's mine The most loneliest day of my life»*

Результат:



1. Запись результата в файл *myFile.txt*
2. Вывод:
3. Разработан способ хранения строки символов с маркером.
4. Организовано динамическое размещение данных в виде массива символов (char \*).
5. Обеспечено эффективное выполнение операций по преобразованию строки (class String).
6. Реализовано чтение строки из файла любого размера (метод read\_data\_from\_file)