*Федеральное государственное бюджетное образовательное учреждение* *высшего образования*

|  |  |
| --- | --- |
|  | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(национальный исследовательский университет)***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ \_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_Компьютерные Системы и сети (ИУ6)\_\_\_\_\_\_\_\_\_\_\_

**Отчет**

**по домашней работе № 2**

**Название домашней работы: Программирование на С++**

**Дисциплина: Объектно-ориентированное программирование**

Студент гр. ИУ6-22Б  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. П. Плютто**



(Подпись, дата) (И.О. Фамилия)



Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О. А. Веселовская**

(Подпись, дата) (И.О. Фамилия)

Москва, 2023

**Задание 1.**

Дана последовательность строк. Каждая строка состоит из слов, разделенных пробелами. Написать программу, обеспечивающую ввод строк и их корректировку. Корректировка заключается в удалении лишних пробелов и слов, состоящих из одного символа. Лишними считаются пробелы в начале и конце строки, а также более одного пробела между словами. Вывести на печать исходную и скорректированную последовательности строк. Реализовать как с использованием библиотечных функций работы со строками, так и с помощью собственной их реализации. Оценить время выполнения программы разными реализациями.

**Решение**

Так как необходимо выполнить задание двумя способами: с помощью стандартных библиотек и полностью своим решением, то для первого решения я воспользуюсь самым распространённым методом любой корректировки строк регулярными выражениями. Стоит заметить, что в C++ в стандартной библиотеке regax регулярные выражения компилируются, а не поддерживаются нативно, из-за этого скорость программы может возрастать. Для собственного решения я буду использовать стандартный цикл, а для того, чтобы не пропустить, например тройные пробелы в начале строки после каждого нахождения необходимой комбинации символов буду обнулять счетчик.

*#include* "iostream"

*#include* "regex"

*#include* <chrono>

using std::cin, std::cout,

std::regex, std::string,

std::regex\_replace, std::regex\_search;

string regexDesition(string str)

{

*auto* start\_time = std::chrono::steady\_clock::now();

regex re1("\\s[a-zA-Z1-9]\\s"), re2("\\s\\s"), re3("^\\s"), re4("\\s$");

*while* (regex\_search(str, re1) or

regex\_search(str, re2) or

regex\_search(str, re3) or

regex\_search(str, re4))

str =

regex\_replace(

regex\_replace(

regex\_replace(

regex\_replace(

str, re4, ""),

re3, ""),

re2, " "),

re1, " ");

*auto* end\_time = std::chrono::steady\_clock::now();

*auto* elapsed\_ns = std::chrono::duration\_cast<std::chrono::nanoseconds>(end\_time - start\_time);

std::cout << "Time:" << elapsed\_ns.count() << " ns\n";

*return* str;

}

*int* strSpace(*int* c)

{

*return* c == 32;

}

string strGetSub(string str, *int* start, *int* end)

{

string subStr;

*for* (*int* i = start; i < end; i++)

subStr = subStr + str[i];

*return* subStr;

}

string strDelete(string str, *int* start, *int* step)

{

*return* strGetSub(str, 0, start) +

strGetSub(str, start + step, str.length());

}

string myDesition(string str)

{

*auto* start\_time = std::chrono::steady\_clock::now();

*int* i = 0;

*while* (i < str.length())

{

*if* (strSpace(str[i]) and strSpace(str[i + 1]))

{

str = strDelete(str, i + 1, 1);

i = 0;

};

*if* (strSpace(str[i]) and strSpace(str[i + 2]))

{

str = strDelete(str, i + 1, 2);

i = 0;

};

*if* (i == 0 and strSpace(str[i]))

{

str = strDelete(str, i, 1);

i = 0;

};

*if* (i == str.length() - 1 and strSpace(str[i]))

{

str = strDelete(str, i, 1);

i = 0;

};

i++;

};

*auto* end\_time = std::chrono::steady\_clock::now();

*auto* elapsed\_ns = std::chrono::duration\_cast<std::chrono::nanoseconds>(end\_time - start\_time);

std::cout << "Time:" << elapsed\_ns.count() << " ns\n";

*return* str;

}

*int* main()

{

string str;

getline(cin, str);

cout << "Input:" << str << "\n\n";

cout << "My desition:\n"

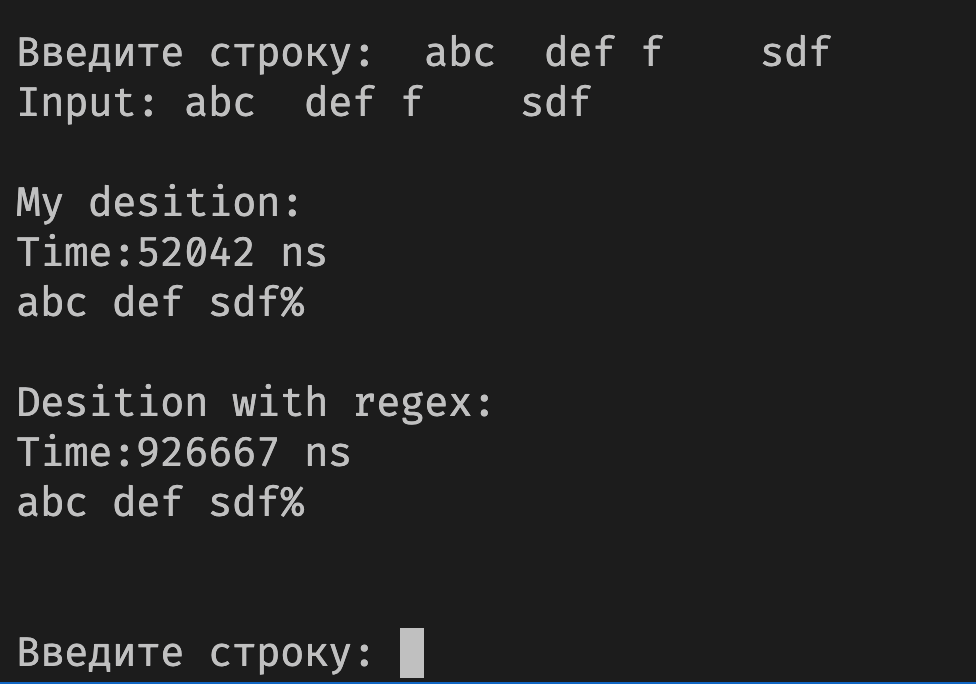
<< myDesition(str) << "%\n\n";

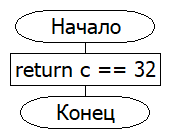
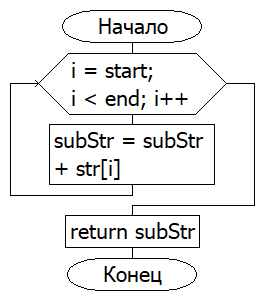
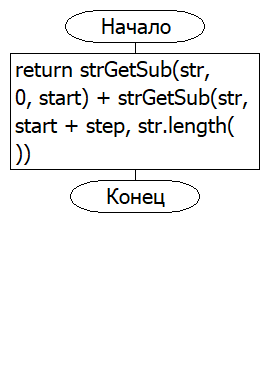
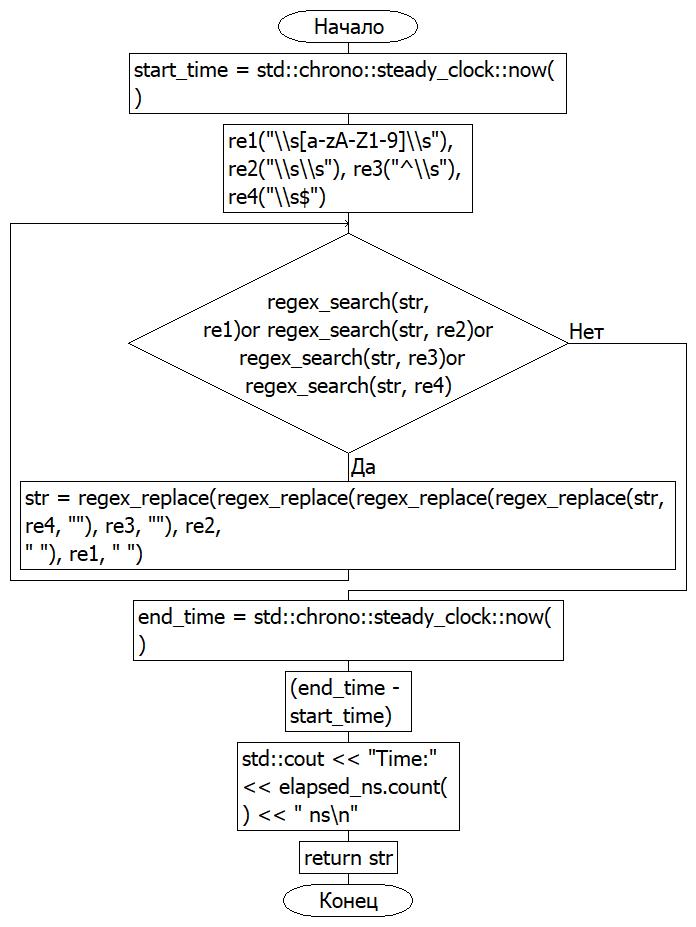
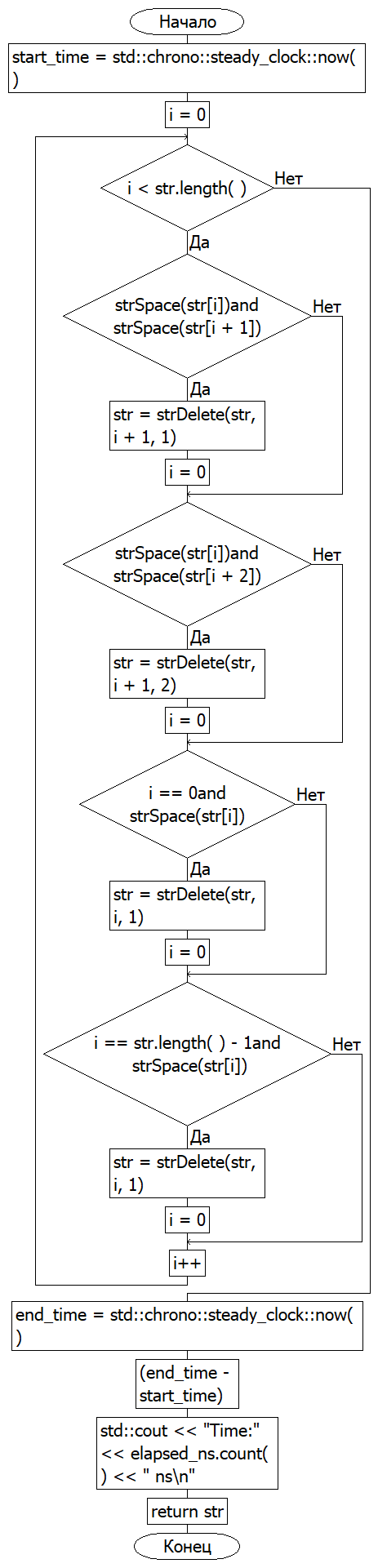
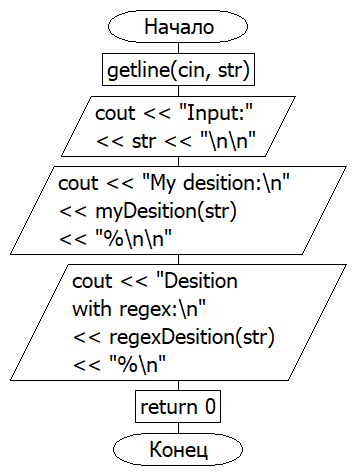
cout << "Desition with regex:\n"

<< regexDesition(str) << "%\n";

*return* 0;

}





Вывод: я научился изменять строки в C++

***Часть 2.2. Файлы***

Дан файл, содержащий записи о книгах и годах их публикации. Разработать программу, которая обеспечить индексацию книг по алфавиту и году выпуска. Записать индексы в отдельные файлы. Применить двоичный поиск нужных записей. Продемонстрировать использование индексов. Сравнить время поиска на объеме 1 млн. произвольных записей в случае поиска по индексным файлам и по исходному файлу без загрузки всех данных в оперативную память.

**Решение**

*#include* <iostream>

*#include* <fstream>

*#include* <string>

*#include* <chrono>

using std::ifstream, std::string, std::ofstream,

std::cout, std::cin,

std::getline, std::copy, std::getline, std::stoi;

string cutStr(string str)

{

*int* count = 0;

*for* (*int* i = 0; i < str.length(); i++)

{

*if* (str[i] == ' ')

{

count++;

*if* (count == 4)

{

str = str.substr(0, i - 3);

*break*;

}

}

*else*

{

count = 0;

}

}

*return* str;

}

string cutStrInv(string str)

{

*int* count = 0;

*for* (*int* i = 0; i < str.length(); i++)

{

*if* (str[i] == ' ')

{

count++;

*if* (count == 4)

{

str = str.substr(i, str.length());

*break*;

}

}

*else*

{

count = 0;

}

}

*return* str;

}

*int* getSize()

{

ifstream file("book\_titles.txt");

string line;

*int* i = 0;

*while* (getline(file, line))

{

i++;

}

cout << "Найден размер файла: " << i << " строк\n";

file.close();

*return* i;

}

string \*readFile(*int* i)

{

string \*mas = new string[i];

ifstream file("book\_titles.txt");

*int* j = 0;

*while* (getline(file, mas[j]))

{

j++;

};

file.close();

cout << "Файл прочитан\n";

*return* mas;

}

*int* partition(string arr[], *int* ims[], *int* low, *int* high)

{

string pivot = arr[high];

*int* i = low - 1;

*for* (*int* j = low; j < high; j++)

{

*if* (arr[j] < pivot)

{

i++;

string temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

*int* tm = ims[i];

ims[i] = ims[j];

ims[j] = tm;

}

}

string temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

*int* tm = ims[i + 1];

ims[i + 1] = ims[high];

ims[high] = tm;

*return* i + 1;

}

*void* quickSort(string arr[], *int* ims[], *int* low, *int* high)

{

*if* (low < high)

{

*int* pi = partition(arr, ims, low, high);

quickSort(arr, ims, low, pi - 1);

quickSort(arr, ims, pi + 1, high);

}

}

*void* indFile(*int* \*iMas, *int* size)

{

ofstream file("index.txt");

*for* (*int* i = 1; i < size; i++)

{

file << iMas[i] << "\n";

}

file << iMas[size];

file.close();

cout << "Индексы сохранены"

<< "\n";

}

*int* binarySearch(string arr[], *int* left, *int* right, string x)

{

*if* (right >= left)

{

*int* mid = left + (right - left) / 2;

string sh = cutStr(arr[mid]);

*if* (sh == x)

{

*return* mid;

}

*if* (sh > x)

{

*return* binarySearch(arr, left, mid - 1, x);

}

*return* binarySearch(arr, mid + 1, right, x);

}

*return* -1;

}

*int* searchInt(*int* arr[], *int* size, *int* x)

{

*for* (*int* i = 0; i < size; i++)

*if* (arr[i] == x)

*return* i;

*return* -1;

}

string fileSearch(*int* x)

{

ifstream file("book\_titles.txt");

string line;

*int* i = 0;

*while* (getline(file, line))

{

*if* (i == x)

*break*;

i++;

}

file.close();

*return* line;

}

*int* fileSearchInd(*int* x)

{

ifstream file("index.txt");

string line;

*int* i = 0;

*while* (getline(file, line))

{

*if* (i == x)

*break*;

i++;

}

file.close();

*return* stoi(line);

}

*int* binarySearchFile(*int* left, *int* right, string x)

{

*if* (right >= left)

{

*int* mid = left + (right - left) / 2;

string sh = cutStr(fileSearch(fileSearchInd(mid)));

*if* (sh == x)

{

*return* mid + 1;

}

*if* (sh > x)

{

*return* binarySearchFile(left, mid - 1, x);

}

*return* binarySearchFile(mid + 1, right, x);

}

*return* -1;

}

*auto* now()

{

*return* std::chrono::steady\_clock::now();

}

*auto* out(std::\_\_1::chrono::steady\_clock::time\_point start\_time)

{

std::cout << "Время: "

<< (std::chrono::duration\_cast<std::chrono::nanoseconds>(now() - start\_time)).count()

<< " ns\n";

}

*int* main()

{

const *int* size = getSize();

string \*ms = readFile(size);

*int* ims[size];

*for* (*int* i = 0; i < size; i++)

ims[i] = i;

quickSort(ms, ims, 0, size);

cout << "Файл отсортирован"

<< "\n";

indFile(ims, size);

string inp = "";

*while* (true)

{

cout << "\n\nВведите название книги для поиска или 0 чтобы выйти: ";

getline(cin, inp);

*if* (inp != "0")

{

*auto* start\_time = now();

*int* outSort = binarySearch(ms, 0, size - 1, inp);

cout << "Индекс: " << outSort

<< " | Номер строки в файле: " << searchInt(ims, size, outSort)

<< " | Год выхода: " << cutStrInv(ms[outSort])

<< "гг | Массивы в лок. памяти | ";

out(start\_time);

start\_time = now();

outSort = binarySearchFile(0, size - 1, inp);

cout << "Индекс: " << outSort

<< " | Номер строки в файле: " << fileSearchInd(outSort)

<< " | Год выхода: " << cutStrInv(ms[outSort])

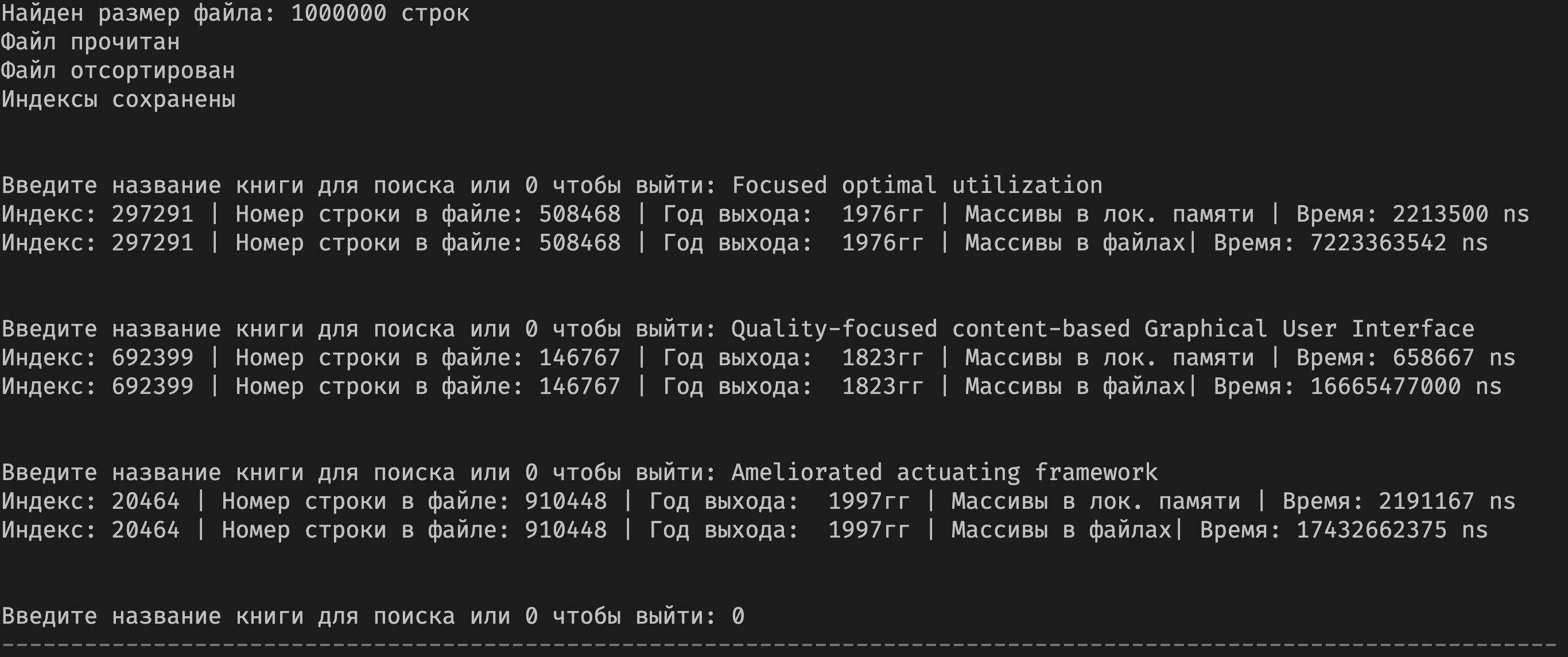
<< "гг | Массивы в файлах| ";

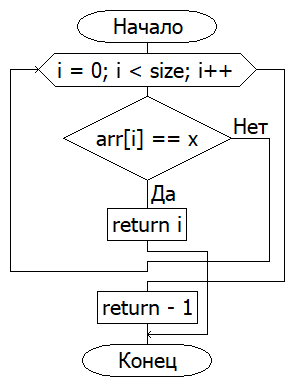
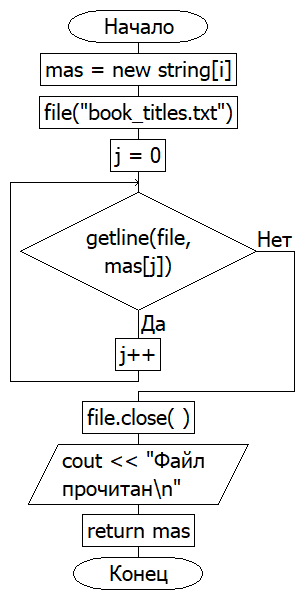
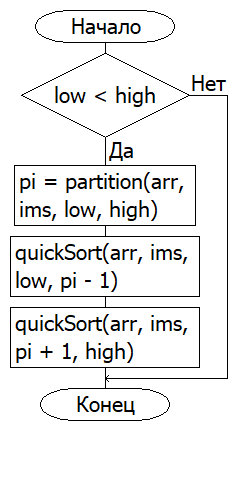
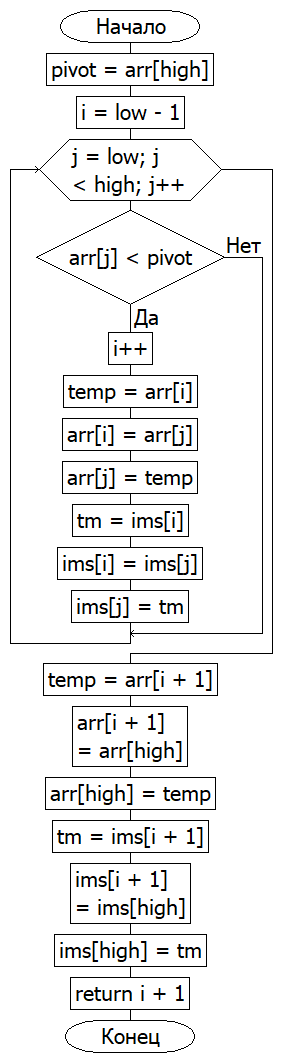
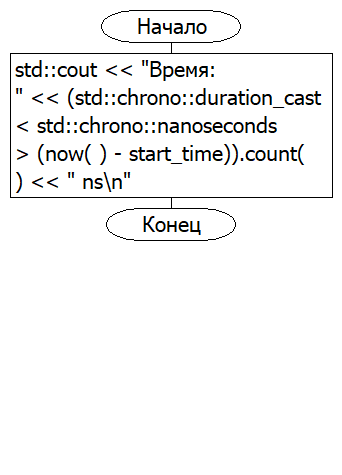
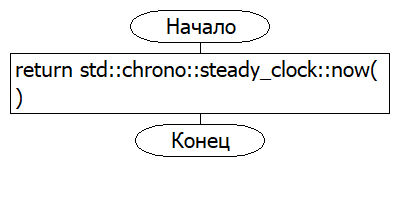
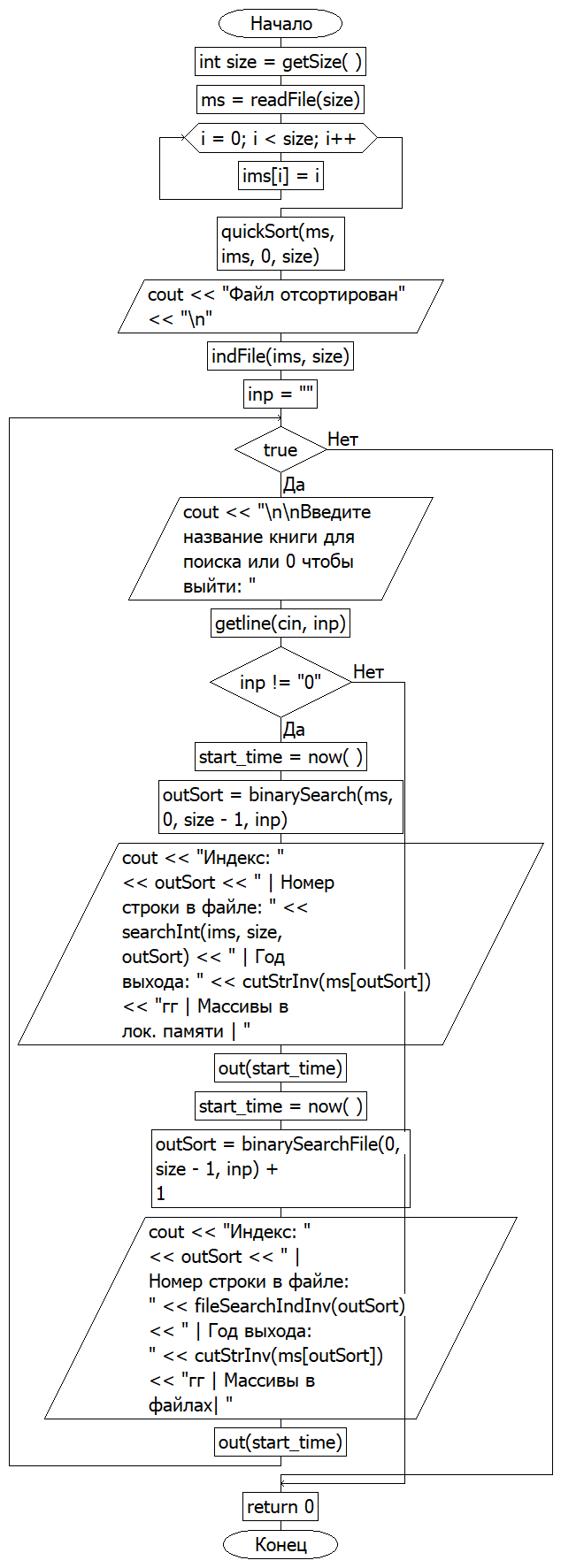
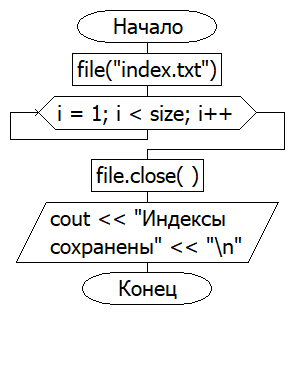
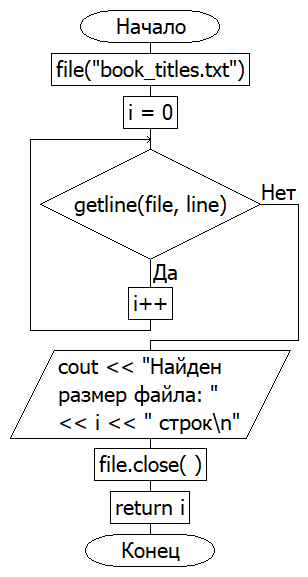
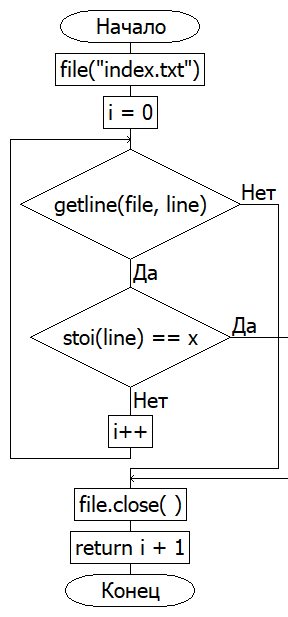
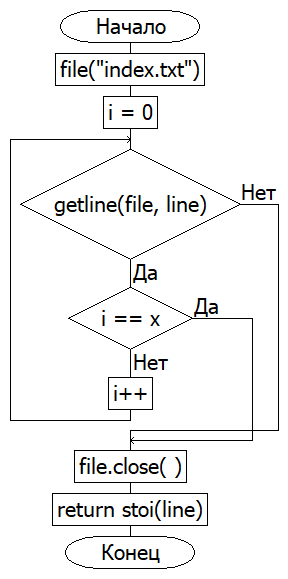
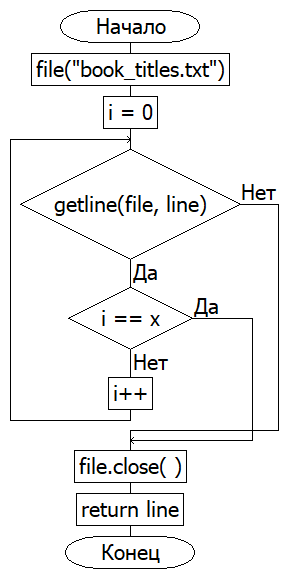
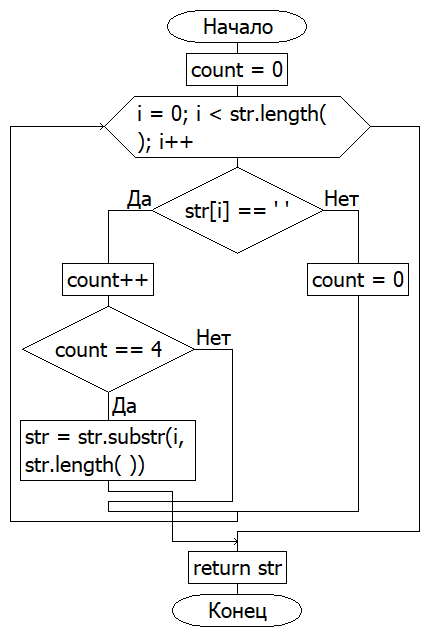
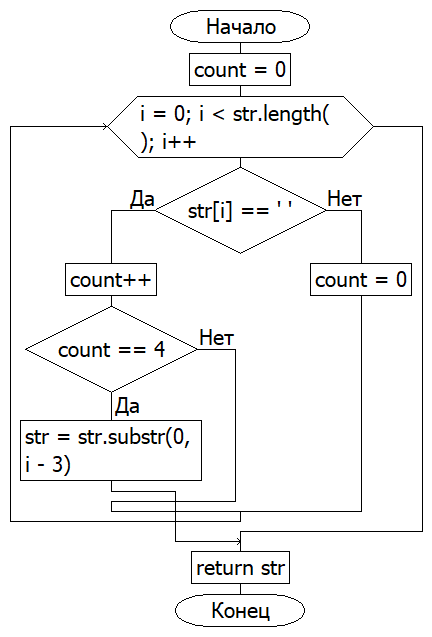
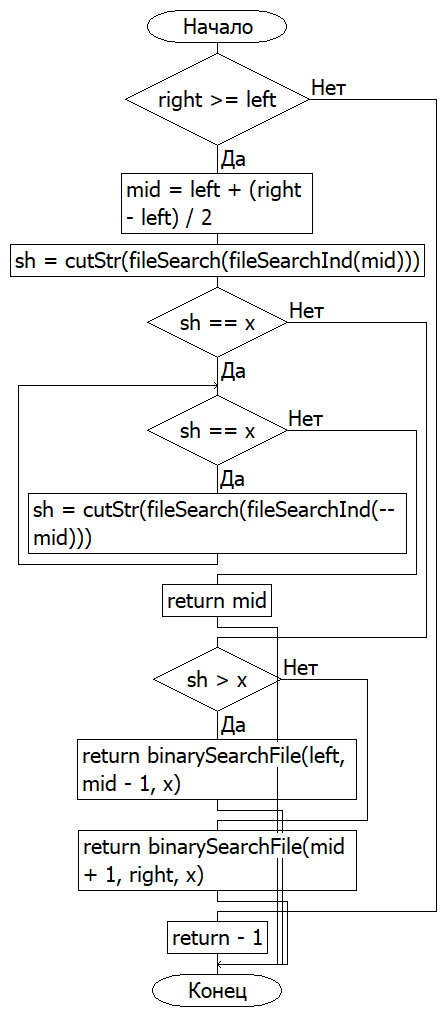
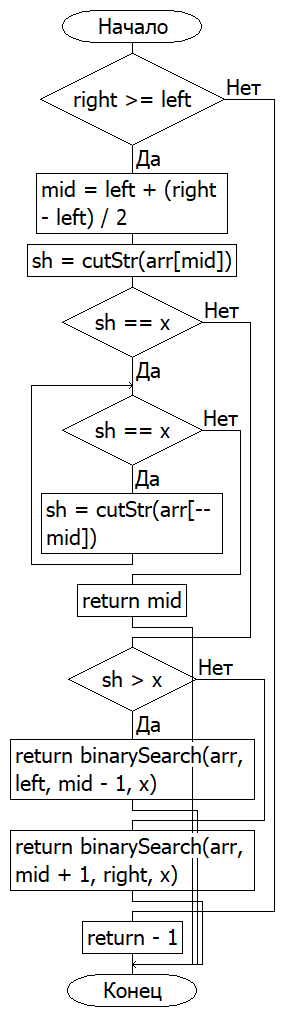
out(start\_time);

}

*else* *break*;};

*return* 0;}

****



Вывод: Я научился использовать файлы, быструю индексную сортировку, бинарный поиск, выяснил, что обращение к оперативной памяти занимает меньше времени, чем обращение к файлам на диске.

Задание 3

Описать объект, включающий заданные поля и методы. Написать программу, которая создает динамический массив объектов, содержит процедуры, работающие с указанными структурами, а также создает двоичное дерево, ключом в котором является слово, а значением – позиция объекта в массиве.

Объект – слово. Параметры: текст, длина, количество согласных букв. Методы: конструктор, определяющий поле текста и длину, и процедура, определяющая количество гласных букв. Определить количество слов в предложении, в которых согласных букв больше, чем гласных. Реализовать поиск объектов по слову с использованием дерева. Программа не должна содержать ограничение длины массива. Проверить, что все созданные объекты удаляются.

Решение

*#include* "iostream"

using std::string, std::getline, std::cin, std::cout;

*class* Word

{

*public:*

string text;

*int* length, consChars;

*int* getConsChars();

Word(string t, *int* len)

{

text = t;

length = len;

getConsChars();

}

};

*int* Word::getConsChars()

{

*char* cons[] = {'a', 'e', 'y', 'u', 'i', 'o'};

consChars = 0;

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

{

*for* (*int* j = 0; j < 6; j++)

{

*if* (text[i] == cons[j])

{

consChars++;

}

}

}

*return* consChars;

}

*struct* Node

{

Word data;

Node \*left;

Node \*right;

};

Node \*insert(Node \*root, Word val)

{

*if* (root == NULL)

{

*return* new Node{val, nullptr, nullptr};

}

*if* (val.text < root->data.text)

{

root->left = insert(root->left, val);

}

*else*

{

root->right = insert(root->right, val);

}

*return* root;

}

*bool* search(Node \*root, string val)

{

*if* (root == NULL)

{

*return* false;

}

*if* (root->data.text == val)

{

*return* true;

}

*if* (val < root->data.text)

{

*return* search(root->left, val);

}

*else*

{

*return* search(root->right, val);

}

}

*int* count;

*void* preorder(Node \*root)

{

*if* (root == NULL)

{

*return*;

}

*if* ((root->data.length - root->data.consChars) > root->data.consChars)

count++;

preorder(root->left);

preorder(root->right);

}

*int* z1(string sentence, *int* len, string \*toSearch, *int* lenToSearch)

{

Node \*splitSentence = NULL;

*int* j = 0;

*for* (*int* i = 0; i < len; i++)

{

*if* (sentence[i] == ' ')

{

Word word(sentence.substr(j, i - j), i - j);

splitSentence = insert(splitSentence, word);

j = i;

}

};

Word word(sentence.substr(j, len - j), len - j);

splitSentence = insert(splitSentence, word);

preorder(splitSentence);

*for* (*int* i = 0; i < lenToSearch; i++)

{

cout << toSearch[i] << ": " << search(splitSentence, toSearch[i]) << "\n";

}

*return* count;

}

*int* main()

{

string sentence;

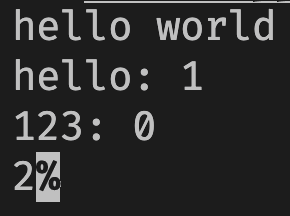
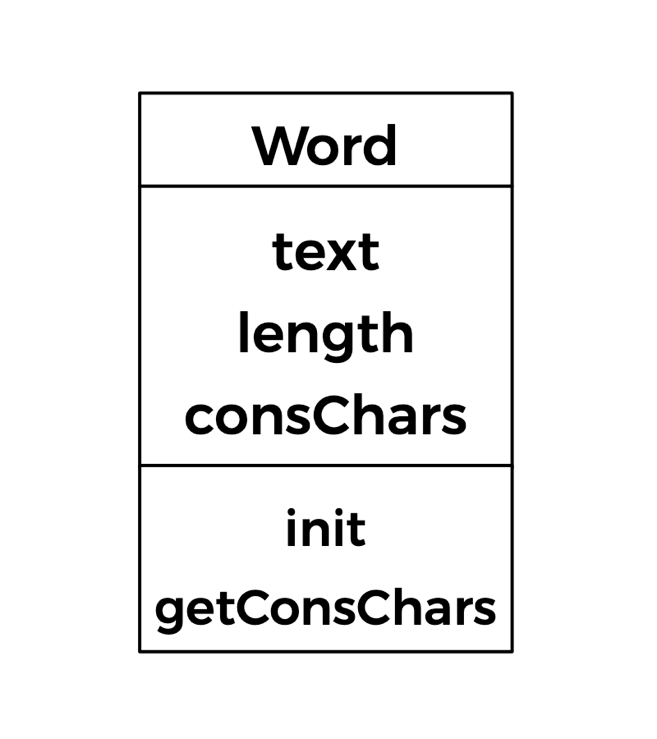
getline(cin, sentence);

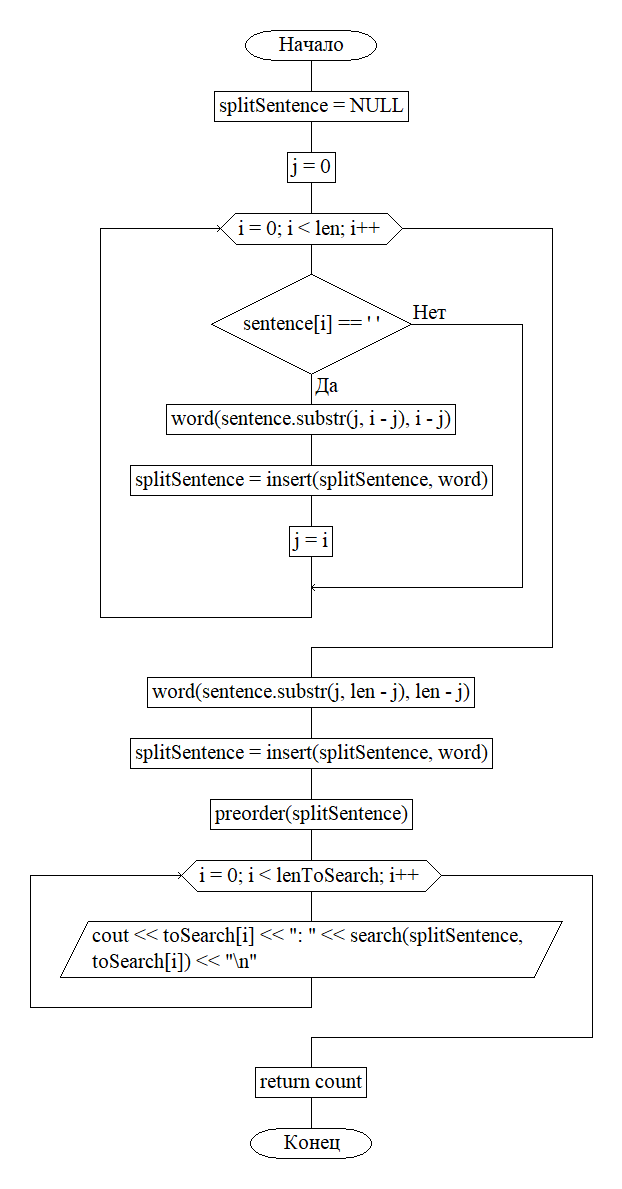
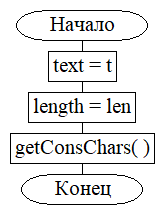
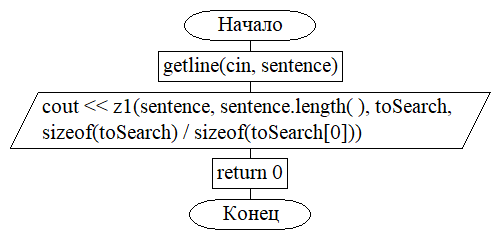
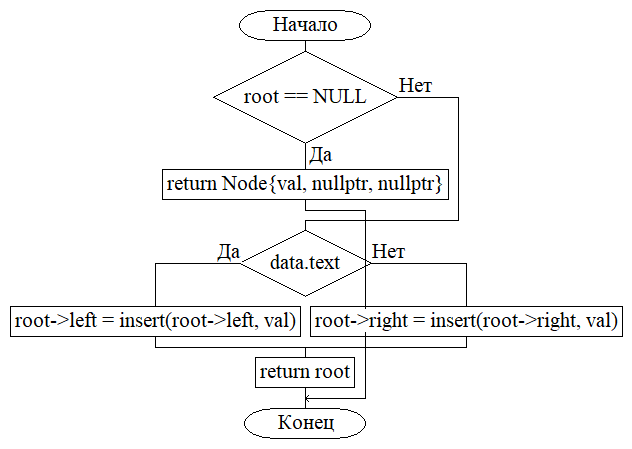
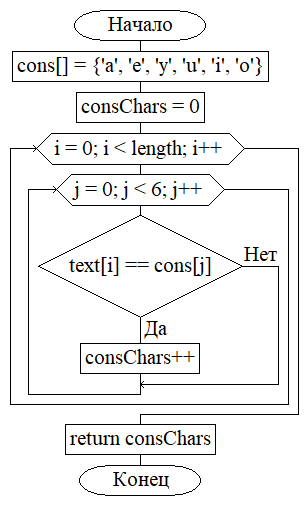
string toSearch[]{"hello", "123"};

cout << z1(sentence, sentence.length(), toSearch, sizeof(toSearch) / sizeof(toSearch[0]));

*return* 0;

}





Вывод: я научился реализовывать бинарные деревья в C++.