Министерство образования и науки Российской Федерации

Санкт-Петербургский Политехнический Университет Петра Великого

1. —

Институт компьютерных наук и кибербезопасности

**ЛАБОРАТОРНАЯ РАБОТА № 4**

1. «**Подсчет количества целых шестнадцатеричных**
2. **чисел в программе на языке С**»
3. по дисциплине «Структуры данных»
   1. Выполнил
   2. студент гр. 5151001/40001 Кириллов Д.А.

<*подпись*>

* 1. Преподаватель
  2. асс. преподавателя Семьянов П.В.

<*подпись*>

1. Санкт-Петербург
2. 2025

**Цель работы**

Научиться выбирать оптимальную структуру данных для реализации самого быстрого поиска.

**Задачи**

1. Выбрать оптимальную структуру данных для реализации поиска целых шестнадцатеричных чисел в файле программы на языке Си.
2. Разработать алгоритм нахождения в файле данных чисел, алгоритм вставки и балансировки, если она требуется, а также алгоритм удаления комментариев из программы.
3. Произвести подсчет чисел.

**Особенности работы**

1. Файл с программой может быть очень большой.
2. Указанные объекты могут быть как равновероятными, так и нет.

**Выбор подходящей структуры данных**

Для данной работы было выбрано самобалансирующееся дерево, то есть дерево, которые адаптируется к тем входным данным, которые у нас есть. Данное дерево – это Splay-дерево. Основная идея заключается в том, что после любой операции над деревом, ту вершину, над которой мы делали операцию, перемещаем в самый корень дерева. В случае удаления вершины её родитель перемещается в самый корень. Данная идея приводит к тому, что если вершина встречается часто, то она будет находиться ближе к корню, следовательно, её поиск будет занимать меньше времени. Если вершины будут все равновероятны, то они будут достаточно равномерно распределяться по правому и левому поддеревьями. Основная операция данного дерева – операция «Splay» (вынос вершины в самый корень).

**Ход работы**

Была создана структура дерева SplayTree, в которой мы будем хранить 16-ричную запись числа, 10-ричную, количество повторений, а также ссылки на родителя и поддеревья.

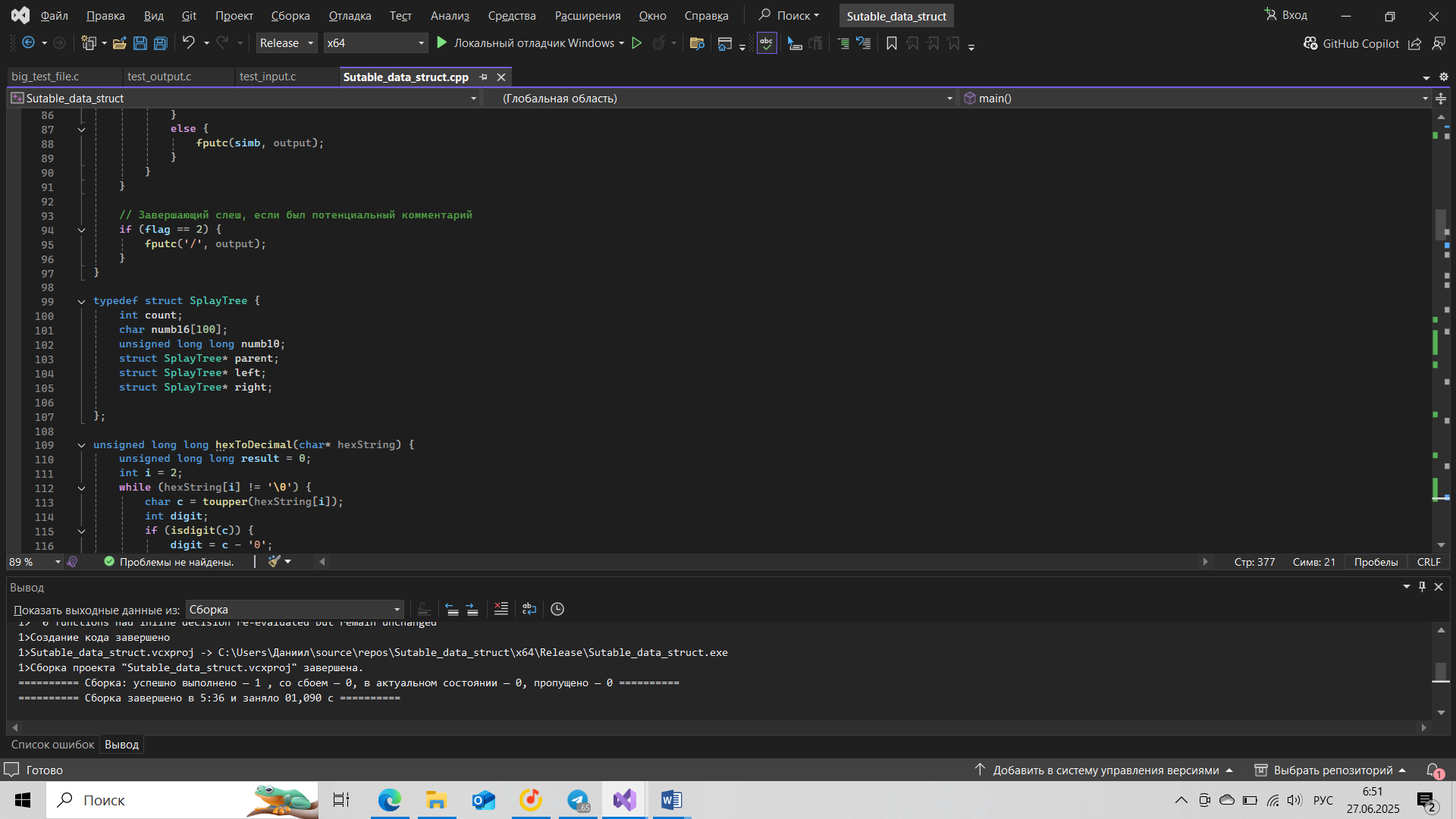


Рисунок 1 – структура дерева.

Так как файл нам неизвестен, мы будем считывать его посимвольно и искать префикс 0х, которым обозначаются числа в 16-ричной С.С., предварительно удалив из него комментарии.

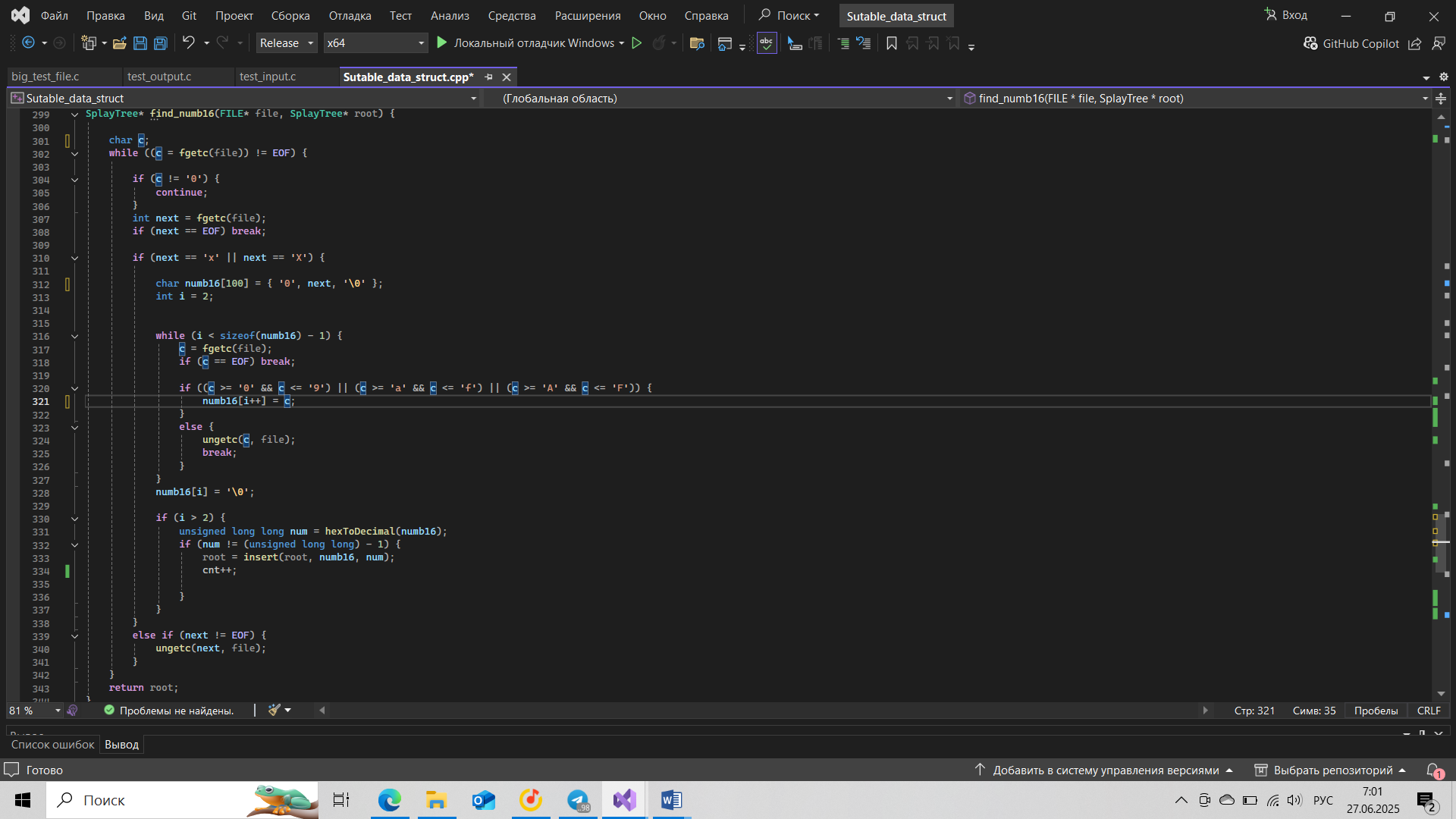


Рисунок 2 – операция поиска 16-ричного числа.

После нахождения числа мы переведем его в 10-ричную с.с и вызовем операцию вставки, в которой мы будем искать это числа и увеличивать счетчик, если оно есть в дереве, либо создадим новый узел.

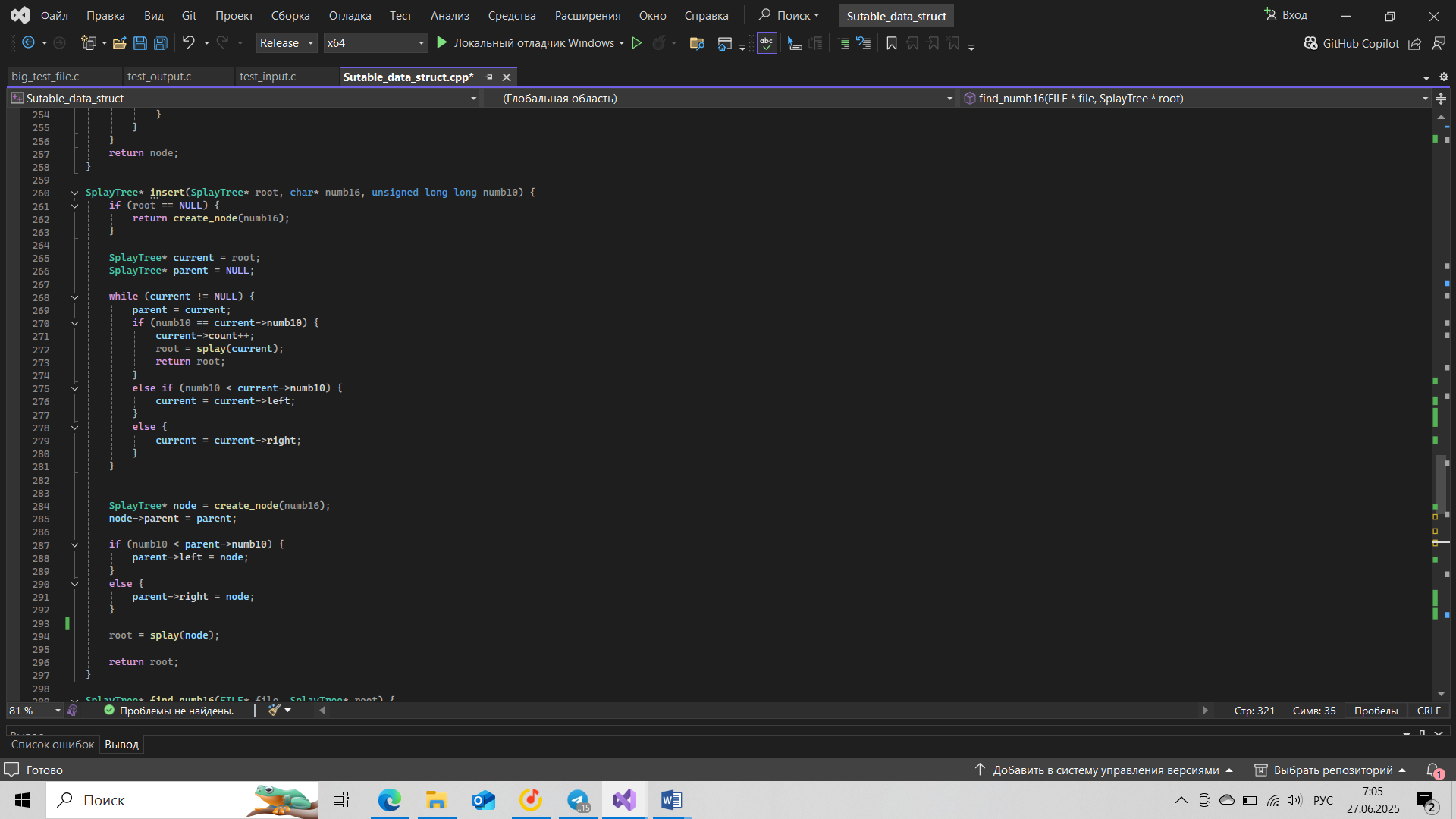


Рисунок 3 – функция вставки

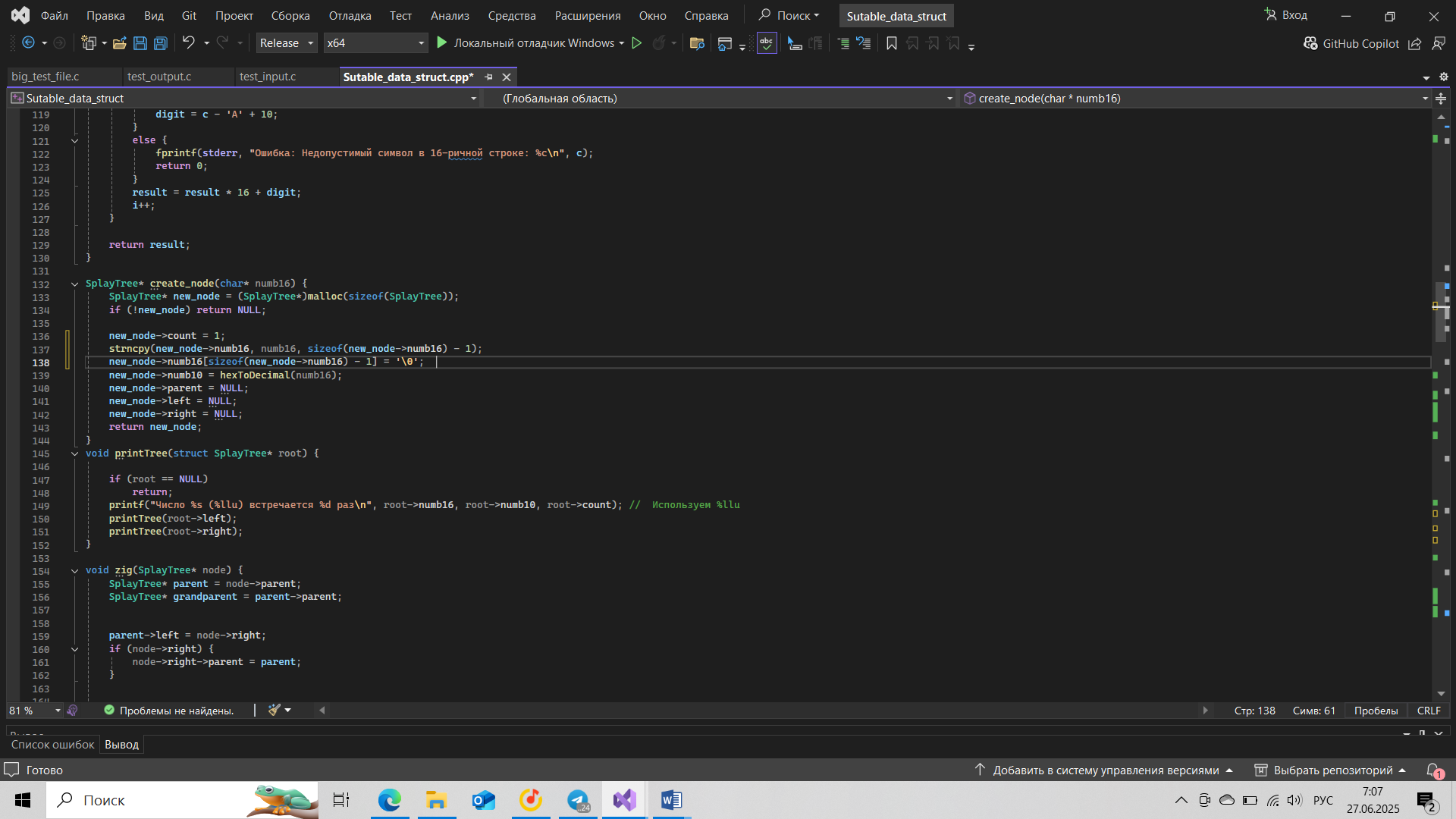


Рисунок 4 – функция создания нового узла

Однако в любом случае после будет идти операция Splay – подъем добавленного узла, либо узла, в котором увеличили счетчик, к корню. Осуществляться это будет с помощью функций zig и zag, с помощью которых можно реализовать все случаи балансировки, а именно zig, zag, zig-zig, zag-zag,zig-zag,zag-zig. Операция Splay будет осуществляться итеративно до момента, пока отцом узла не станет NULL, что означает, что узел стал корнем.

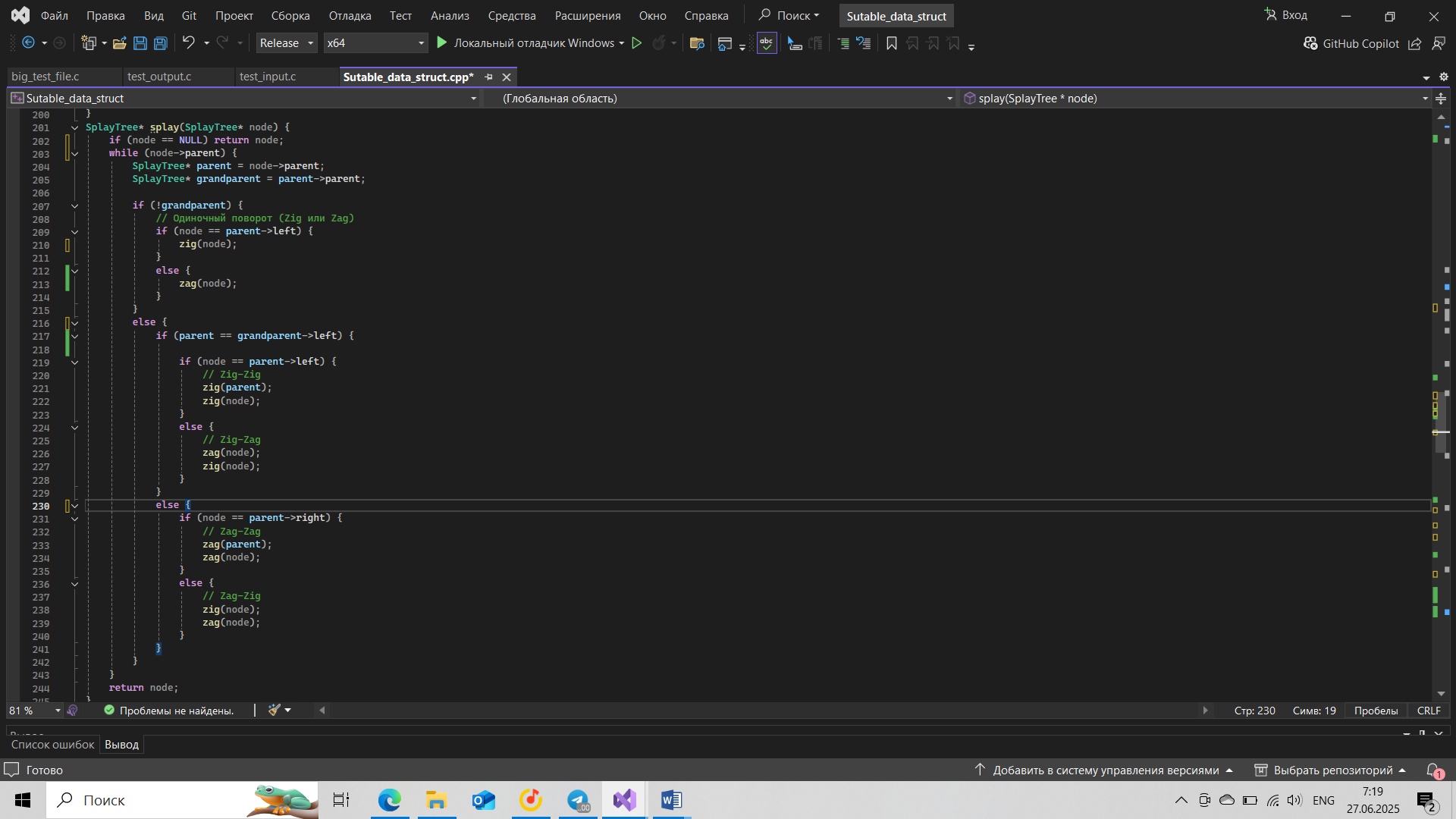


Рисунок 5 – операция Splay

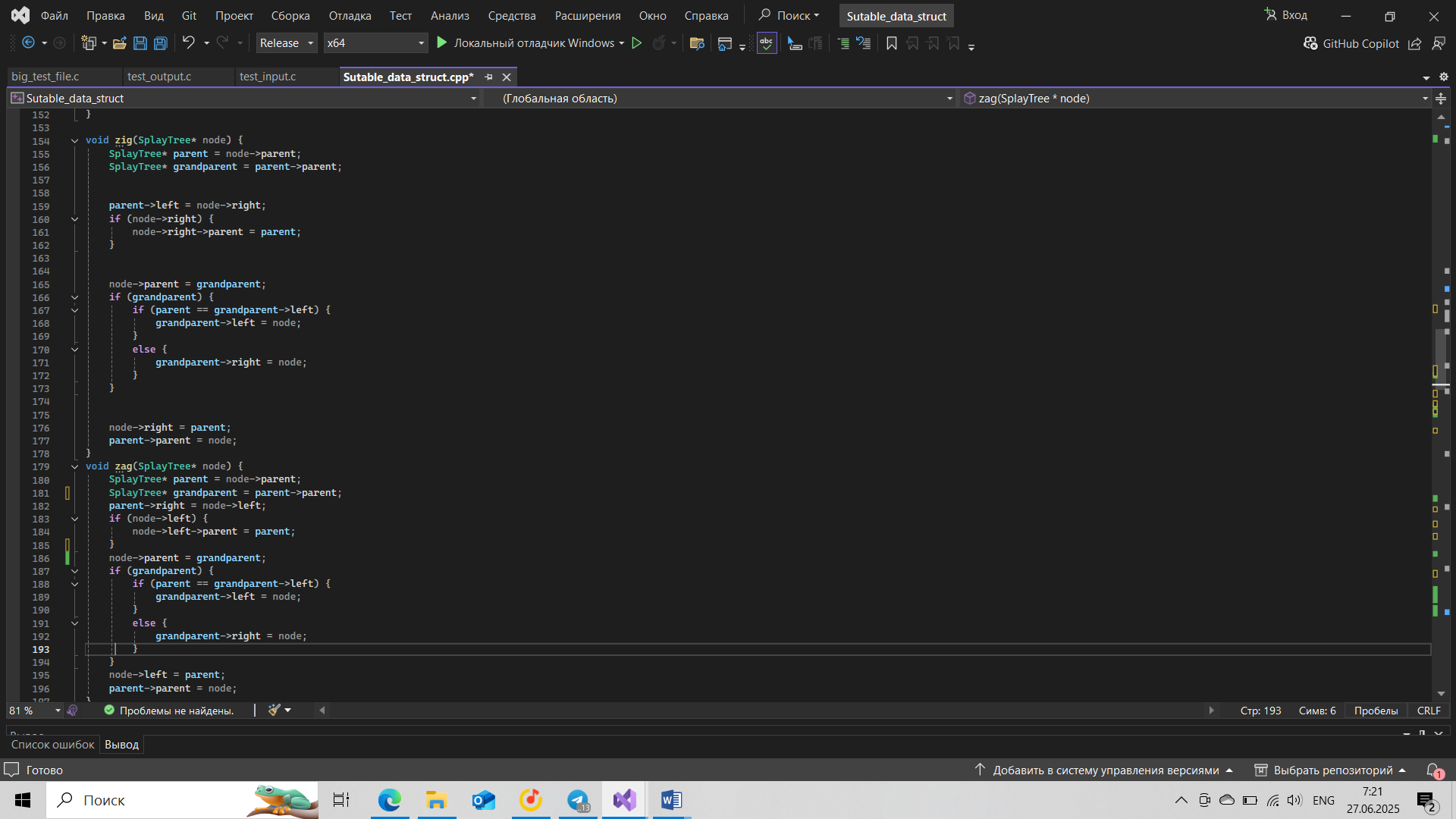


Рисунок 6 – повороты zig и zag

После добавления всех чисел в дерево будет вызвана функция префиксного вывода дерева (сначала выводится вершина, а потом идет обход потомков)

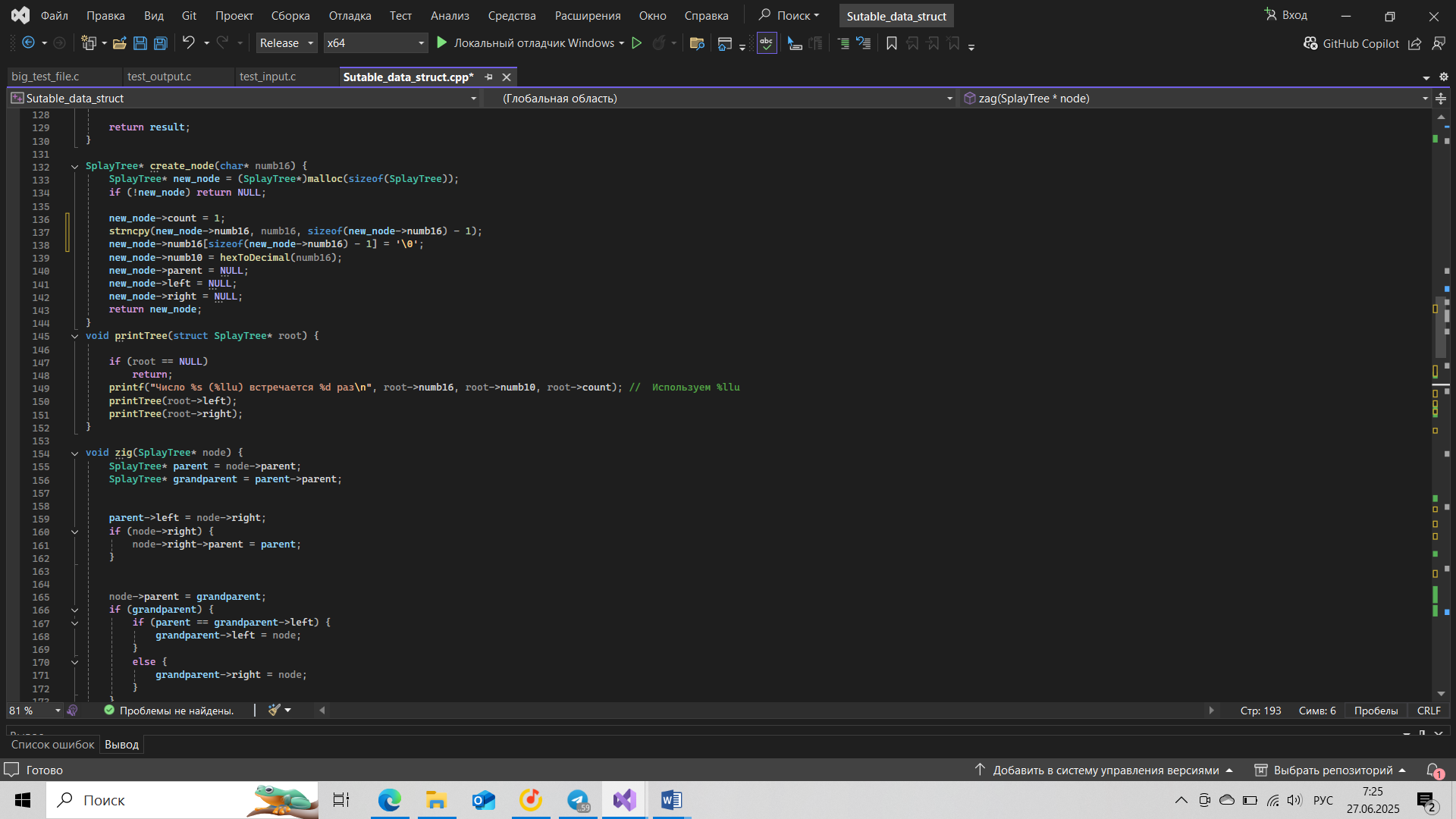


Рисунок 7 – префиксный обход дерева

**Результаты**

Был создан файл на си размером несколько мегабайт с неоднородным наполнением 16-ричными числами. Как и ожидалось, числа, которые встречались чаще всего, оказались ближе к вершине.

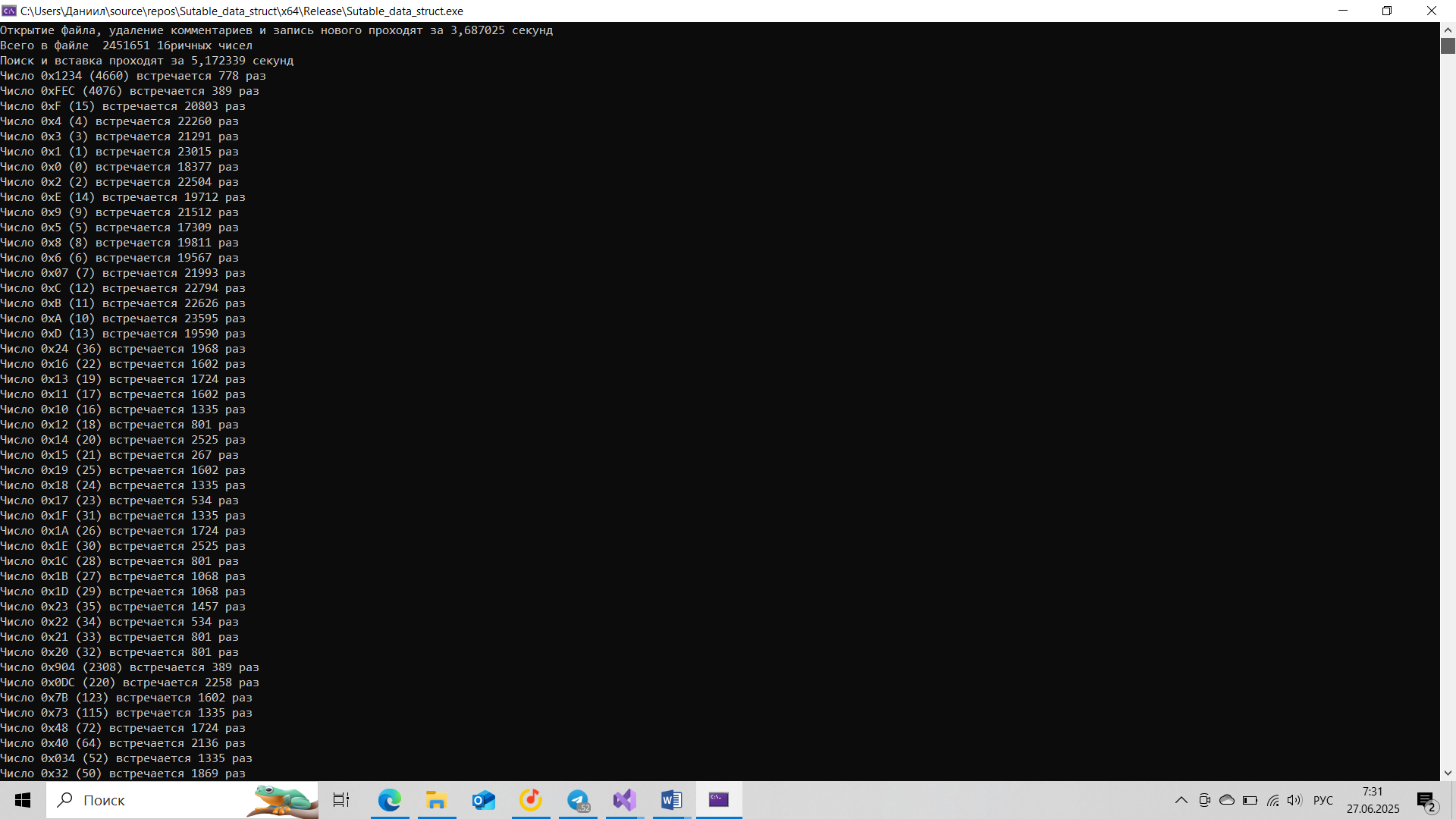


Рисунок 8 – результаты обработки файла

**Вывод**

После выполнения данной работы мы научились выбирать оптимальную структуру данных для реализации самого быстрого поиска выданных нам объектов. Научились реализовывать Splay – дерево, работать с ним и правильно работать с операцией Splay. Также вспомнили работу с рекурсиями.

Приложение 1

// Sutable\_data\_struct.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

#include <ctype.h>

#include <limits.h>

#include <time.h>

#include <intrin.h>

long cnt = 0;

void delcom(FILE\* input, FILE\* output) {

int flag = 0; // Используем int вместо short

int simb; // int для корректной работы с EOF

int presimb = 0;

while ((simb = fgetc(input)) != EOF) {

switch (flag) {

case 1: // Строковый литерал

fputc(simb, output);

if (simb == '\\') {

int next = fgetc(input);

if (next == EOF) break;

fputc(next, output);

}

else if (simb == '"') {

flag = 0;

}

break;

case 5: // Символьный литерал

fputc(simb, output);

if (simb == '\\') {

int next = fgetc(input);

if (next == EOF) break;

fputc(next, output);

}

else if (simb == '\'') {

flag = 0;

}

break;

case 2: // Потенциальный комментарий

if (simb == '/') {

flag = 3; // Однострочный комментарий

}

else if (simb == '\*') {

flag = 4; // Многострочный комментарий

}

else {

fputc('/', output);

fputc(simb, output);

flag = (simb == '"') ? 1 : (simb == '\'') ? 5 : 0;

}

break;

case 3: // Однострочный комментарий

if (simb == '\\' && fgetc(input) == '\n') {

continue;

}

if (simb == '\n') {

fputc(simb, output);

flag = 0;

}

break;

case 4: // Многострочный комментарий

if (presimb == '\*' && simb == '/') {

flag = 0;

}

presimb = simb;

break;

default: // Обычный текст

if (simb == '"') {

flag = 1;

fputc(simb, output);

}

else if (simb == '\'') {

flag = 5;

fputc(simb, output);

}

else if (simb == '/') {

flag = 2;

}

else {

fputc(simb, output);

}

}

}

// Завершающий слеш, если был потенциальный комментарий

if (flag == 2) {

fputc('/', output);

}

}

typedef struct SplayTree {

int count;

char numb16[100];

unsigned long long numb10;

struct SplayTree\* parent;

struct SplayTree\* left;

struct SplayTree\* right;

};

unsigned long long hexToDecimal(char\* hexString) {

unsigned long long result = 0;

int i = 2;

while (hexString[i] != '\0') {

char c = toupper(hexString[i]);

int digit;

if (isdigit(c)) {

digit = c - '0';

}

else if ((c >= 'A' && c <= 'F')|| (c >= 'a' && c <= 'f')) {

digit = c - 'A' + 10;

}

else {

fprintf(stderr, "Ошибка: Недопустимый символ в 16-ричной строке: %c\n", c);

return 0;

}

result = result \* 16 + digit;

i++;

}

return result;

}

SplayTree\* create\_node(char\* numb16) {

SplayTree\* new\_node = (SplayTree\*)malloc(sizeof(SplayTree));

if (!new\_node) return NULL;

new\_node->count = 1;

strncpy(new\_node->numb16, numb16, sizeof(new\_node->numb16) - 1);

new\_node->numb16[sizeof(new\_node->numb16) - 1] = '\0';

new\_node->numb10 = hexToDecimal(numb16);

new\_node->parent = NULL;

new\_node->left = NULL;

new\_node->right = NULL;

return new\_node;

}

void printTree(struct SplayTree\* root) {

if (root == NULL)

return;

printf("Число %s (%llu) встречается %d раз\n", root->numb16, root->numb10, root->count); // Используем %llu

printTree(root->left);

printTree(root->right);

}

void zig(SplayTree\* node) {

SplayTree\* parent = node->parent;

SplayTree\* grandparent = parent->parent;

parent->left = node->right;

if (node->right) {

node->right->parent = parent;

}

node->parent = grandparent;

if (grandparent) {

if (parent == grandparent->left) {

grandparent->left = node;

}

else {

grandparent->right = node;

}

}

node->right = parent;

parent->parent = node;

}

void zag(SplayTree\* node) {

SplayTree\* parent = node->parent;

SplayTree\* grandparent = parent->parent;

parent->right = node->left;

if (node->left) {

node->left->parent = parent;

}

node->parent = grandparent;

if (grandparent) {

if (parent == grandparent->left) {

grandparent->left = node;

}

else {

grandparent->right = node;

}

}

node->left = parent;

parent->parent = node;

}

SplayTree\* splay(SplayTree\* node) {

if (node == NULL) return node;

while (node->parent) {

SplayTree\* parent = node->parent;

SplayTree\* grandparent = parent->parent;

if (!grandparent) {

// Одиночный поворот (Zig или Zag)

if (node == parent->left) {

zig(node);

}

else {

zag(node);

}

}

else {

if (parent == grandparent->left) {

if (node == parent->left) {

// Zig-Zig

zig(parent);

zig(node);

}

else {

// Zig-Zag

zag(node);

zig(node);

}

}

else {

if (node == parent->right) {

// Zag-Zag

zag(parent);

zag(node);

}

else {

// Zag-Zig

zig(node);

zag(node);

}

}

}

}

return node;

}

SplayTree\* insert(SplayTree\* root, char\* numb16, unsigned long long numb10) {

if (root == NULL) {

return create\_node(numb16);

}

SplayTree\* current = root;

SplayTree\* parent = NULL;

while (current != NULL) {

parent = current;

if (numb10 == current->numb10) {

current->count++;

root = splay(current);

return root;

}

else if (numb10 < current->numb10) {

current = current->left;

}

else {

current = current->right;

}

}

SplayTree\* node = create\_node(numb16);

node->parent = parent;

if (numb10 < parent->numb10) {

parent->left = node;

}

else {

parent->right = node;

}

root = splay(node);

return root;

}

SplayTree\* find\_numb16(FILE\* file, SplayTree\* root) {

char c;

while ((c = fgetc(file)) != EOF) {

if (c != '0') {

continue;

}

int next = fgetc(file);

if (next == EOF) break;

if (next == 'x' || next == 'X') {

char numb16[100] = { '0', next, '\0' };

int i = 2;

while (i < sizeof(numb16) - 1) {

c = fgetc(file);

if (c == EOF) break;

if ((c >= '0' && c <= '9') || (c >= 'a' && c <= 'f') || (c >= 'A' && c <= 'F')) {

numb16[i++] = c;

}

else {

ungetc(c, file);

break;

}

}

numb16[i] = '\0';

if (i > 2) {

unsigned long long num = hexToDecimal(numb16);

if (num != (unsigned long long) - 1) {

root = insert(root, numb16, num);

cnt++;

}

}

}

else if (next != EOF) {

ungetc(next, file);

}

}

return root;

}

void freeSplayTree(struct SplayTree\* root) {

if (root == NULL) {

return;

}

freeSplayTree(root->left);

freeSplayTree(root->right);

free(root);

}

int main()

{

setlocale(LC\_ALL, "rus");

unsigned \_\_int64 start = \_\_rdtsc();

FILE\* input = fopen("big\_test\_file.c", "r");

FILE\* output = fopen("test\_output.c", "w+");

delcom(input, output);

fclose(input);

fclose(output);

unsigned \_\_int64 end = \_\_rdtsc();

printf("Открытие файла, удаление комментариев и запись нового проходят за %f секунд\n", (double)(end - start) / 1900000000.0);

FILE\* test\_prog = fopen("test\_output.c", "r");

SplayTree\* root = NULL;

start = \_\_rdtsc();

root = find\_numb16(test\_prog, root);

end = \_\_rdtsc();

printf("Всего в файле %ld 16ричных чисел \nПоиск и вставка проходят за %f секунд\n", cnt,(double)(end - start) / 1900000000.0);

printTree(root);

freeSplayTree(root);

fclose(test\_prog);

return 0;

}