Министерство науки и высшего образования Российской Федерации

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа №8

по дисциплине

«Программирование»

Вариант 8

Выполнил: студент группы Асу15-бз

Чиков Дмитрий Александрович

Проверил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

г. Пермь – 2020

**Цель работы:**

1) Получить практические навыки работы с однонаправленными списками;

2) получить практические навыки работы с двунаправленными списками;

3) получить практические навыки работы с деревьями.

**Задачи работы:**

1. Сформировать однонаправленный список, тип информационного поля указан в варианте.
2. Распечатать полученный список.
3. Выполнить обработку списка в соответствии с заданием.
4. Распечатать полученный список.
5. Удалить список из памяти.
6. Сформировать двунаправленный список, тип информационного поля указан в варианте.
7. Распечатать полученный список.
8. Выполнить обработку списка в соответствии с заданием.
9. Распечатать полученный список.
10. Удалить список из памяти.
11. Сформировать идеально сбалансированное бинарное дерево, тип информационного поля указан в варианте.
12. Распечатать полученное дерево.
13. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.
14. Преобразовать идеально сбалансированное дерево в дерево поиска.
15. Распечатать полученное дерево.

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | Тип информационного поля char\*.  Добавить в список элементы с номерами 1, 3, 5 и т. д. | Тип информационного поля int.  Удалить из списка последний элемент с четным информационным полем. | Тип информационного поля char.  Найти количество элементов с заданным ключом. |

Для выполнения работы были реализованы следующие функции:

Файл bst.h: - содержит в себе элементы для работы с деревом

struct Node { - описание структуры дерева

…

};

struct Node\* process(struct Node\* root) – определение вертикальных координат элементов дерева

struct Node\* modify(struct Node\* root, int shft) – аналогично для горизонтальных координат

int search(struct Node\* root, char key, int c) – поиск элементов в дереве

void main\_search(struct Node\* root) – вспомогательная процедура

void inorder(struct Node\* root) – вывод дерева в консоль

struct Node\* getNode(char val, int height) – создание элементов дерева

struct Node\* Insert(struct Node\* root, int height, int n) – добавление значения в информационное поле

int findHeight(struct Node\* root) – нахождение «высоты» дерева

pair<struct Node\*, int> bst\_init() – первичное построение дерева

void Forming\_tree(pair<struct Node\*, int> p, struct Node\* root, int height) – основная процедура из которой вызываются все остальные.

В файле point.h:

struct point

struct point2

point\* make\_list(int n)

void print\_list(point\* beg)

void print\_list(point2\* beg)

void\* popBack(point2\* list)

point\* add\_point(point\* beg, int k)

point\* del\_point(point\* beg, int k)

point2\* make\_point2()

point2\* make\_list2(int n)

В файле draw.h: содержатся процедуры для рисования дерева с применением библиотеки glut.h

void draw\_main(struct Node\* root, int height, int argc, char\* argv[]) – основная процедура, откуда происходит вызов все остальных;

void KeyboardEvent(unsigned char key, int x, int y) – обработка нажатия;

void draw() – процедура вывода изображения на экран;

pair<pair<int, int>, pair<int, int> > removeRad(float x0, float y0, float x1, float y1) – вспомогательная процедура удаления лишних элементов;

void drawText(char data, int x0, int y0, bool searched) – выводит текст из информационного поля дерева;

void MidpointCircle(int x0, int y0) – находит центр круга;

void draw\_circle(int x0, int y0, int x, int y) – рисует круг – элемент дерева;

void draw\_line(int x0, int y0, int x1, int y1, int dir) – рисует линии – ветки между листьями и корнем;

void PlotPixel(GLint x, GLint y) – базовая процедура. Рисует пиксель;

А так же основная функция main(), в основном файле исходного кода, из которой происходит вызов всех остальных функций. Листинг приведен в приложении А.

Тестирование программы выполнялось с использованием двух наборов данных: каждое задание выполнялось дважды.

**Однонаправленный список:**

Результат работы программы для задач 1 и 2 представлен на рисунках 1 и 2.

Результат добавления элементов и печать списков после этого представлен на рисунках 3 и 4.

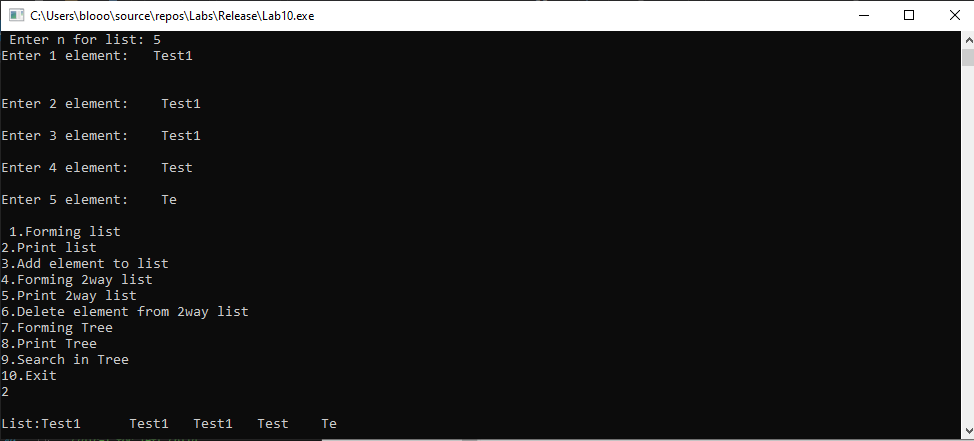


Рисунок 1 Формирование и печать однонаправленного списка

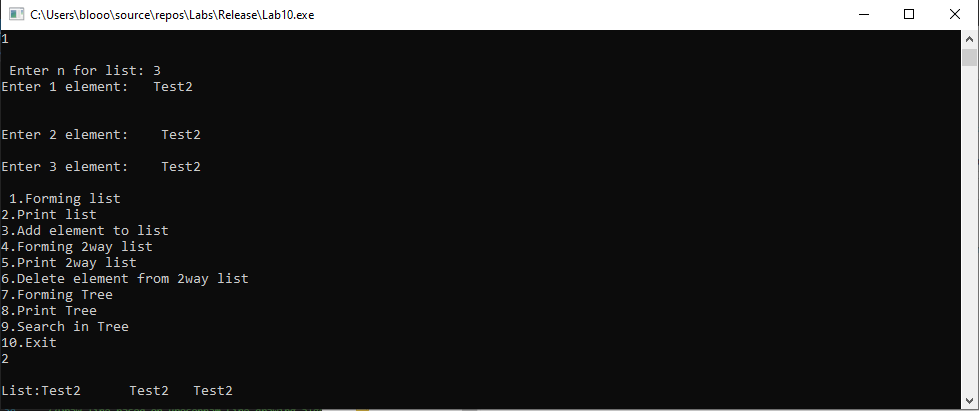


Рисунок 2 Формирование и печать однонаправленного списка

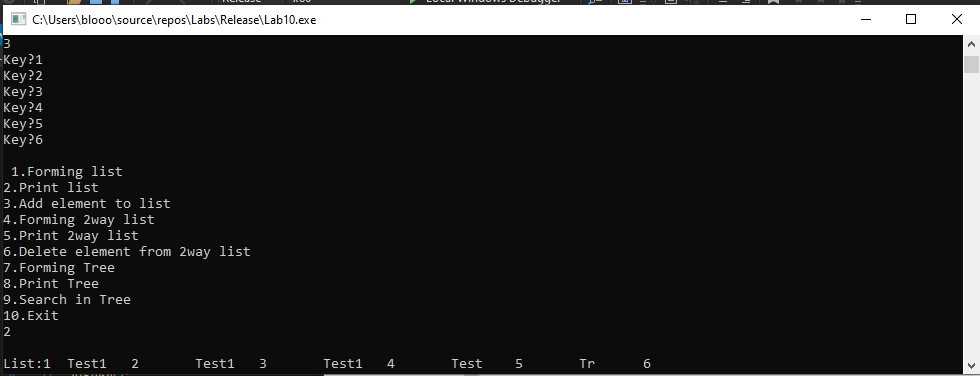


Рисунок 3 Добавление элементов на нечетные места в сформированный список

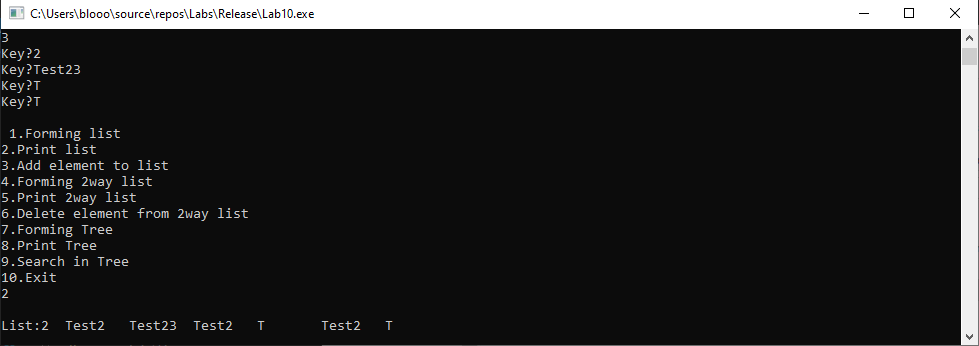


Рисунок 4 Добавление элементов на нечетные места в сформированный список

**Двунаправленный список:**

Результат работы программы для задач 6 и 7 представлен на рисунках 5 и 6.

Результат удаления элементов и печать списка после этого представлен на рисунках 7 и 8.

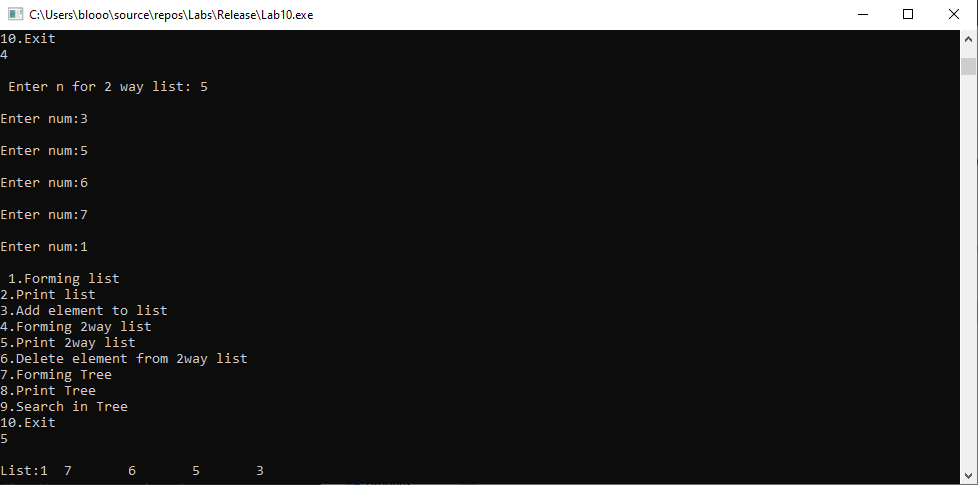


Рисунок 5 Формирование и печать двунаправленного списка

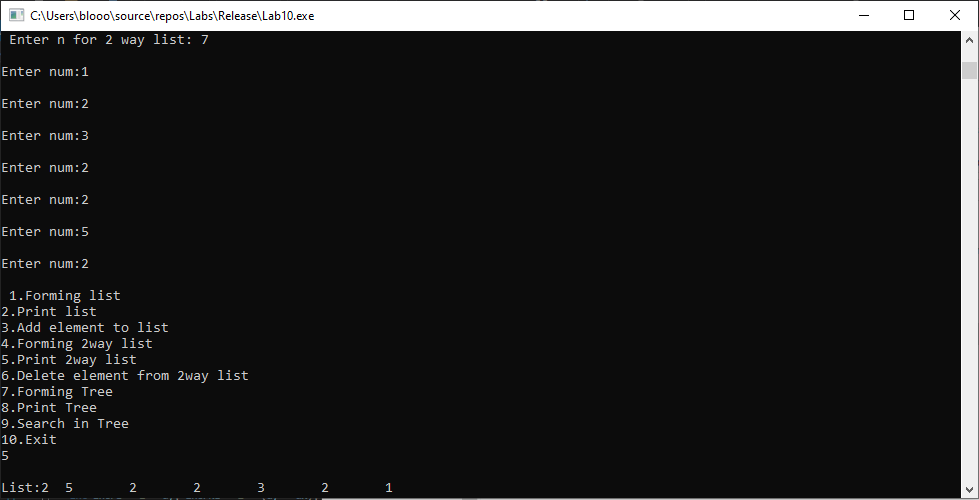


Рисунок 6 Формирование и печать двунаправленного списка

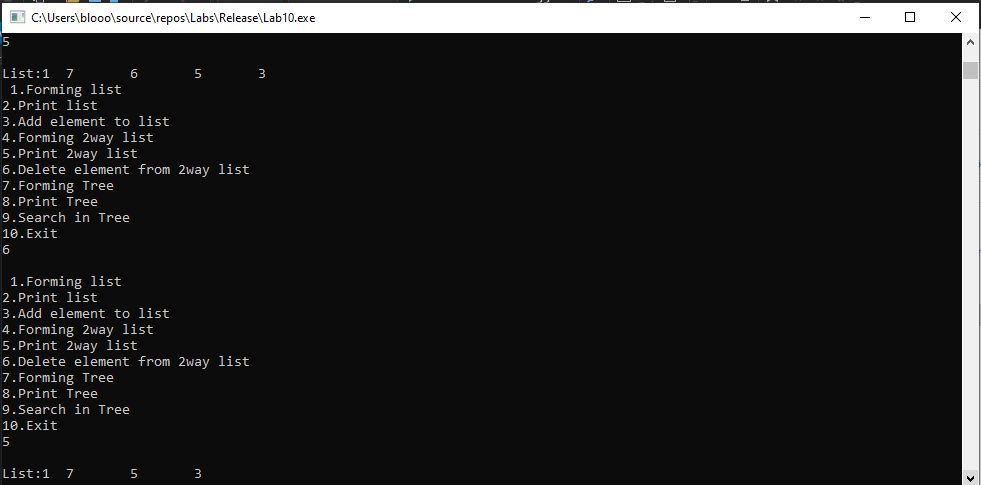


Рисунок 7 Удаление последнего четного элемента и печать

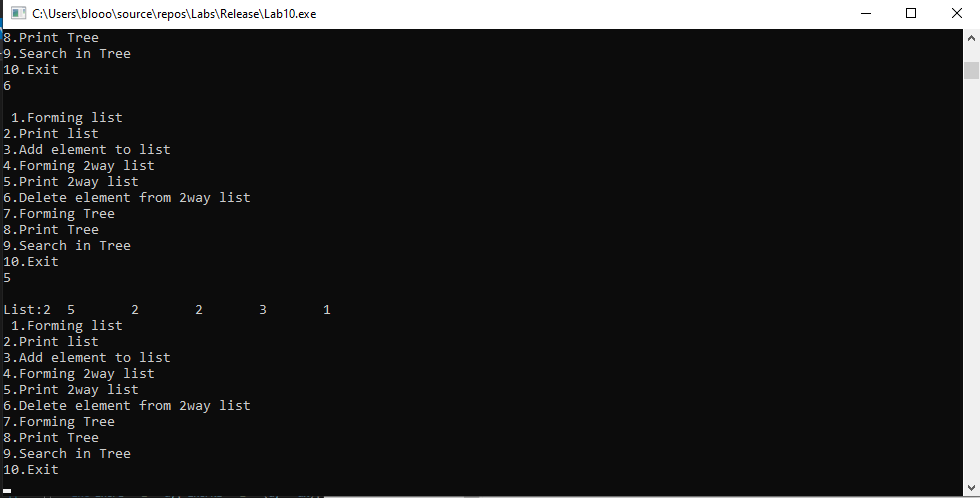


Рисунок 8 Удаление последнего четного элемента и печать

**Дерево:**

Формирование дерева представлено на рисунке 9. В результате формирования дерева в консоли отображается информация о элементах дерева: содержимое информационного поля, координаты, дальность от корня (не должна превышать единицу) и параметр «searched».

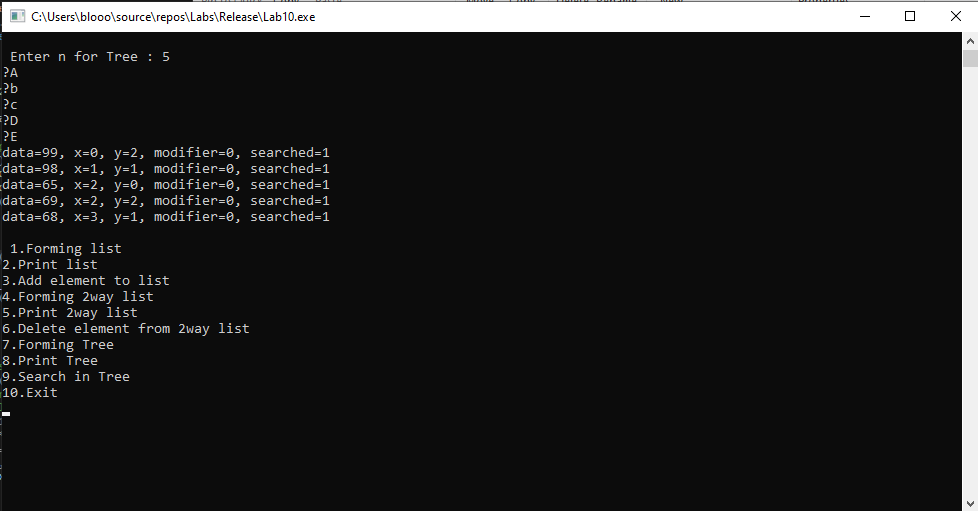


Рисунок 9 Формирование дерева

На рисунке 10 представлена вертикальная печать дерева.

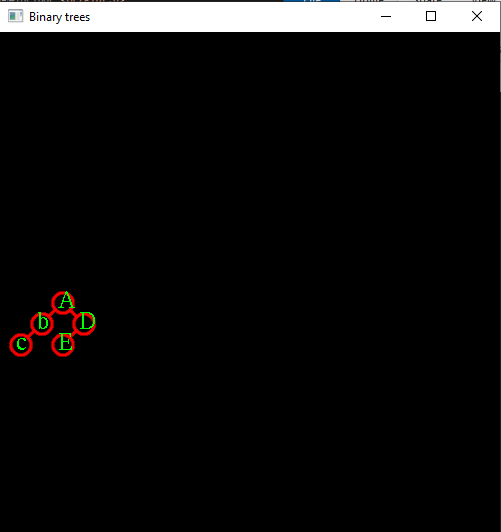


Рисунок 10 Печать дерева

Задача поиска элементов в дереве представлен на рисунках 11 и 12: После отображения информации в консоли есть возможность распечатать дерево – будут видны только удовлетворяющие критериям поиска элементы.

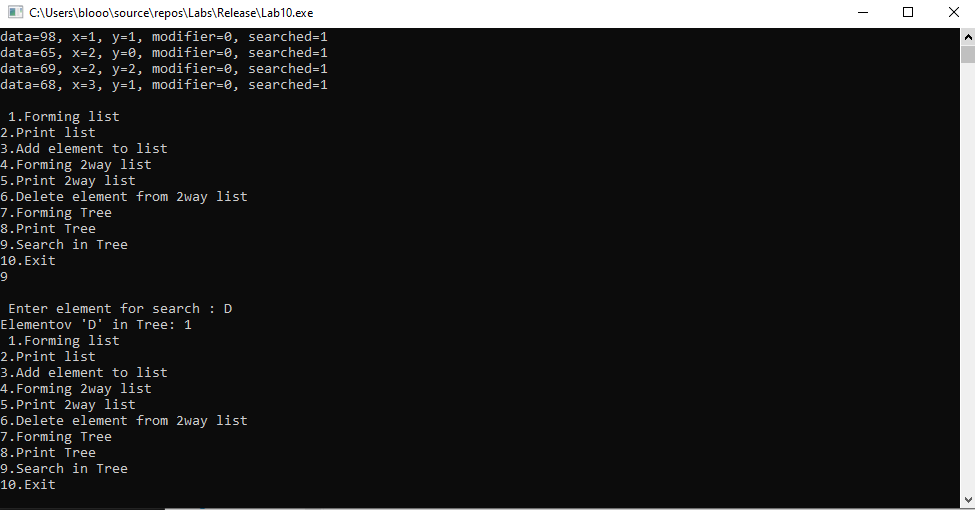


Рисунок 11 Поиск элемента в дереве

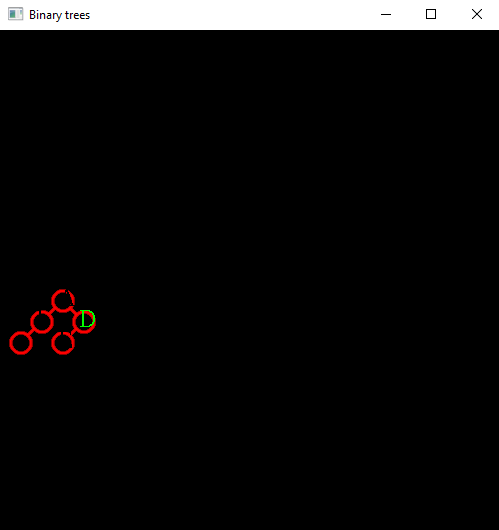


Рисунок 12 Печать дерева после поиска

Результаты второго тестирования находятся в папке с проектом.

**Приложение А**

**Bst.h**

/\*

\* .h file that just creates a basic BST to be used for our programs

\* Returns a pair containing the root and the height of the tree

\*/

#include <iostream>

#include <stdio.h>

#include <cstdlib>

#include <climits>

#include <stdlib.h>

#include <climits>

#include <algorithm>

#include <vector>

using namespace std;

#define inf INT\_MAX

#define \_inf INT\_MIN

int next\_pos[100] = { 0 }; //next available position for each level, initialized with 0

//Structure of node

struct Node {

char data;

int x, y; //x and y coordinates, y also stores the height of the node

int modifier; //amount of shift for the subtree

bool searched;

struct Node\* left, \* right; //left and right subtrees

};

struct Node\* process(struct Node\* root) {

struct Node\* left = root->left, \* right = root->right;

int height = root->y, prov\_pos = \_inf, actual\_pos, modifier = 0;

if (left == NULL && right == NULL) { //if leaf node actual will simply be the 1st available pos on its level

actual\_pos = next\_pos[height];

modifier = 0;

}

else if (left == NULL || right == NULL) { //only one child

if (left == NULL) { //only right child

right = process(right);

prov\_pos = right->x - 1;

}

else { //only left child

left = process(left);

prov\_pos = left->x + 1;

}

actual\_pos = max(prov\_pos, next\_pos[height]);

if (next\_pos[height] > prov\_pos) modifier = next\_pos[height] - prov\_pos;

}

else { //if both children present

left = process(left);

right = process(right);

prov\_pos = (left->x + right->x) / 2;

actual\_pos = max(prov\_pos, next\_pos[height]);

if (next\_pos[height] > prov\_pos) modifier = next\_pos[height] - prov\_pos;

}

next\_pos[height] = actual\_pos + 2; //incrementing next available position on this level

root->x = actual\_pos;

root->modifier = modifier;

return root;

}

//Does the approprite shifting of the subtrees based on modifier

struct Node\* modify(struct Node\* root, int shft) {

if (root == NULL) return NULL;

root->x = root->x + shft; //shift towards right

root->left = modify(root->left, shft + root->modifier);

root->right = modify(root->right, shft + root->modifier);

return root;

}

int search(struct Node\* root, char key, int c) {

if (root) {

if (root->data == key)

{

root->searched = 1;

c++;

}

else

root->searched = 0;

c=search(root->left, key, c);

c=search(root->right, key, c);

}

return c;

}

void main\_search(struct Node\* root) {

std::cout << "\n Enter element for search : ";

char h;

std::cin >> h;

int j = search(root, h, 0);

std::cout << "Elementov '" << h << "' in Tree: " << j;

}

//Inorder traversal of tree

void inorder(struct Node\* root) {

//Inorder traversal of tree

if (root == NULL) return;

inorder(root->left);

printf("data=%d, x=%d, y=%d, modifier=%d, searched=%d\n", root->data, root->x, root->y, root->modifier, root->searched);

inorder(root->right);

}

//Creates a new node and returns it

struct Node\* getNode(char val, int height) {

struct Node\* root = (struct Node\*)malloc(sizeof(struct Node));

root->data = val;

root->searched = 1;

root->x = root->modifier = 0;

root->y = height;

root->left = root->right = NULL;

return root;

}

//Insert node into BST

struct Node\* Insert(struct Node\* root, int height, int n) {

int nl, nr;

char data;

if (n == 0) { root = NULL; return root; }

nl = n / 2;

nr = n - nl - 1;

std::cout << "?";

std::cin >> data;

if (root == NULL)

root = getNode(data, height);

root->left = Insert(root->left, height + 1, nl);

root->right = Insert(root->right, height + 1, nr);

return root;

}

int findHeight(struct Node\* root) {

if (root == NULL) return 0;

return max(findHeight(root->left), findHeight(root->right)) + 1;

}

//Creates a binary tree and returns its root and height

pair<struct Node\*, int> bst\_init() {

struct Node\* root = NULL;

int n; //insert all elements in BST

std::cout << "\n Enter n for Tree : ";

std::cin >> n;

root = Insert(root, 0, n);

int height = findHeight(root);

pair<struct Node\*, int> p = make\_pair(root, height);

return p;

}

void Forming\_tree(pair<struct Node\*, int> p, struct Node\* root, int height)

{

p = bst\_init();

root = p.first;

height = p.second;

root = process(root); //assign coordinates

root = modify(root, 0); //shift based on modifier

inorder(root);

}

**Point.h**

#include <cstdlib>

#include <iostream>

#include <ctime>

#include <string>

#include <stdio.h>

struct point

{

char\* data; //информационное поле

point\* next; //адресное поле

};

//формирование двунаправленного списка

struct point2

{

int key;//информационное поле

point2\* next;//указатель на следующий элемент

point2\* prev;//указатель на предыдущий элемент

};

point\* make\_list(int n)

{

point\* beg;//указатель на первый элемент

point\* p, \* r;//вспомогательные указатели

beg = new(point);//выделяем память под первый элемент

std::cout << "Enter 1 element: ";

beg->data = new char[65536];

std::cin >> beg->data;//вводим значение информационного поля

std::cout << "\n";

beg->next = 0;//обнуляем адресное поле

//ставим на этот элемент указатель p (последний элемент)

p = beg;

int num;

for (int i = 0; i < n-1 ; i++)

{

num = i + 2;

r = new(point);//создаем новый элемент

std::cout << "\nEnter " << num << " element: ";

r->data = new char[65536];

std::cin >> r->data;

r->next = 0;

p->next = r;//связываем p и r

p = r;//ставим на r указатель p (последний элемент)

}

return beg;//возвращаем beg как результат функции

}

void print\_list(point\* beg)

//печать списка

{

std::cout << "\nList:";

point\* p = beg;//начало списка

while (p != 0)

{

std::cout << p->data << "\t";

p = p->next;//переход к следующему элементу

}

}

void print\_list(point2\* beg)

//печать списка

{

std::cout << "\nList:";

point2\* p = beg;//начало списка

while (p != 0)

{

std::cout << p->key << "\t";

p = p->next;//переход к следующему элементу

}

}

void\* popBack(point2\* list) {

point2 \*p = list, \*next, \*prev;//начало списка

while (p->next != NULL)

p = p->next; // переход к концу списка

do {

if (p->key % 2 != 0) {

p = p->prev; // переход к предыдущему узлу

}

else {

prev = p->prev; // узел, предшествующий lst

next = p->next; // узел, следующий за lst

if (prev != NULL)

prev->next = p->next; // переставляем указатель

if (next != NULL)

next->prev = p->prev; // переставляем указатель

free(p); // освобождаем память удаляемого элемента

break;

}

} while (p != NULL); // условие окончания обхода

return list;

}

point\* add\_point(point\* beg, int k)

//добавление элемента с номером k

{

point\* p = beg;//встали на первый элемент

point\* New = new(point);//создали новый элемент

std::cout << "Key?";

New->data = new char[65536];

std::cin >> New->data;

if (k == 0)//добавление в начало, если k=0

{

New->next = beg;

beg = New;

return beg;

}

for (int i = 0; i < k - 1 && p != 0; i++)

p = p->next;//проходим по списку до(k-1) элемента или до конца

if (p != 0)//если k-й элемент существует

{

New->next = p->next;//связываем New и k-й элемент

p->next = New;//связываем (k-1)элемент и New

}

return beg;

}

point\* del\_point(point\* beg, int k)

//удаление элемента с номером k из списка

{

point\* p = beg;

if (k == 0)//удаление первого элемента

{

beg = beg->next;

delete p;

return beg;

}

//проходим по списку до элемента с номером k-1

for (int i = 1; i < k && p->next != 0; i++)

p = p->next;

/\*если такого элемента в списке нет, то возвращаем указатель на начало списка в качестве результата функции\*/

if (p->next == 0) return beg;

point\* r = p->next;//ставим указатель r на k-й элемент

p->next = r->next;//связываем k-1 и k+1 элемент

delete r;//удаляем k-й элемент из памяти

return beg;

}

point2\* make\_point2()

//создание одного элемента

{

point2\* p = new(point2);

p->next = 0; p->prev = 0;//обнуляем указатели

int s;

std::cout << "\nEnter num:";

std::cin >> s;

p->key = s;

return p;

}

point2\* make\_list2(int n)

//создание списка

{

point2\* p, \* beg;

beg = make\_point2();//создаем первый элемент

for (int i = 1; i < n; i++)

{

p = make\_point2();//создаем один элемент

//добавление элемента в начало списка

p->next = beg;//связываем р с первым элементом

beg->prev = p;//связываем первый элемент с p

beg = p;// p становится первым элементом списка

}

return beg;

}

**draw.h**

/\*

\* program to draw the tree on openGL

\* Gets the tree root from main.cpp file

\* Lines between nodes are drawn using Bresenham Line drawing and circular nodes are drawn using midpoint algorithm

\*/

#include <GL/glut.h>

#include <stdlib.h>

#include <iostream>

#include <stdio.h>

#include <climits>

#include <queue>

#include <math.h>

using namespace std;

extern jmp\_buf env;

#define RADIUS 10 //radius of circle

#define SCALE 21 //coordinate scaling factor

#define WINDOW 500 //Size of Window

struct Node\* mainroot = NULL; //root of tree

//Initialize OpenGL

void init(void) {

glClearColor(0, 0, 0, 0.0); //set display-window background color to black

glMatrixMode(GL\_PROJECTION); //set projection paramaters

gluOrtho2D(0.0, 500.0, 0.0, 500.0);

}

//PlotPixel at given coordinate

void PlotPixel(GLint x, GLint y) {

y = WINDOW / 2 - y; //moving origin to top-left, default is bottom-left

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

glFlush(); //executes all OpenGL functions as quickly as possible

}

//Draw line based on Bresenham Line drawing algorithm

void draw\_line(int x0, int y0, int x1, int y1, int dir) {

//dir=1 for left child

//printf("line:(%d,%d) to (%d,%d)\n", x0, y0, x1, y1);

int dx = x1 - x0, dy = y1 - y0;

int d = 2 \* dy - dx;

int incrE = 2 \* dy, incrNE = 2 \* (dy - dx);

int x = x0, y = y0;

PlotPixel(x, y);

while (x < x1) {

if (d <= 0) { //choose E

d = d + incrE;

if (dir) y--; //if left child decrement y

}

else { //choose NE

d = d + incrNE;

if (!dir) y++;

}

x++;

PlotPixel(x, y);

}

}

//plot pixels based on a circle's symmetry

void draw\_circle(int x0, int y0, int x, int y) {

PlotPixel(x0 + x, y0 + y);

PlotPixel(x0 + y, y0 + x);

PlotPixel(x0 - y, y0 + x);

PlotPixel(x0 - x, y0 + y);

PlotPixel(x0 - x, y0 - y);

PlotPixel(x0 - y, y0 - x);

PlotPixel(x0 + y, y0 - x);

PlotPixel(x0 + x, y0 - y);

}

//Draw circle using midpoint circle algorithm

void MidpointCircle(int x0, int y0) {

//printf("Circle: (%d, %d)\n", x0, y0);

int x = 0, y = RADIUS;

int d = 1 - RADIUS;

int deltaE = 3, deltaSE = -2 \* RADIUS + 5;

draw\_circle(x0, y0, x, y);

while (y > x) {

if (d < 0) { //select E

d = d + deltaE;

deltaE = deltaE + 2;

deltaSE += 2;

}

else { //select SE

d = d + deltaSE;

deltaE += 2;

deltaSE += 4;

y--;

}

x++;

draw\_circle(x0, y0, x, y);

}

}

//function to convert integer to string

string to\_string(char s) {

string str = "";

str = str + s;

reverse(str.begin(), str.end());

return str;

}

//Write the data inside the drawn node

void drawText(char data, int x0, int y0, bool searched) {

GLvoid\* font\_style = GLUT\_BITMAP\_TIMES\_ROMAN\_24;

y0 = WINDOW / 2 - y0 - RADIUS / 2;

x0 = x0 - RADIUS / 2; //bring str to center of circle

glRasterPos2f(x0, y0);

glutBitmapCharacter(font\_style, data);

}

//Function to remove part of line that lies inside node's circle

pair<pair<int, int>, pair<int, int> > removeRad(float x0, float y0, float x1, float y1) {

float delx = x1 - x0, dely = y1 - y0;

float mod = sqrt(delx \* delx + dely \* dely);

delx = delx / mod \* RADIUS;

dely = dely / mod \* RADIUS;

x0 = x0 + delx;

x1 = x1 - delx;

y0 = y0 + dely;

y1 = y1 - dely;

return make\_pair(make\_pair(x0, y0), make\_pair(x1, y1));

}

//Function to process the tree and call appropriate line and circle drawing functions at appropriate coordinates

void draw() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 0.0);

glPointSize(2.0);

struct Node\* root = mainroot, \* current;

queue<struct Node\*> Q;

pair<pair<int, int>, pair<int, int> > p;

Q.push(root);

int x0, y0, x1, y1;

while (!Q.empty()) { //BFS of tree

root = Q.front();

Q.pop();

x0 = SCALE \* (root->x + 1);

y0 = SCALE \* (root->y + 1);

MidpointCircle(x0, y0); //draw circle

if (root->searched == 0)

{

glColor3f(0.0, 0.0, 0.0);

}

else

glColor3f(0.0, 1.0, 0.0); //change data color to green

drawText(root->data, x0, y0, root->searched); //draw node value

glColor3f(1.0, 0.0, 0.0); //change drawing color back to red

if (root->left) { //if left exists

current = root->left;

Q.push(current);

x1 = SCALE \* (current->x + 1);

y1 = SCALE \* (current->y + 1);

p = removeRad(x0, y0, x1, y1);

int x02 = p.first.first;

int y02 = p.first.second;

x1 = p.second.first;

y1 = p.second.second;

draw\_line(x1, y1, x02, y02, 1);

}

if (root->right) { //if right exists

current = root->right;

Q.push(current);

x1 = SCALE \* (current->x + 1);

y1 = SCALE \* (current->y + 1);

p = removeRad(x0, y0, x1, y1);

int x02 = p.first.first;

int y02 = p.first.second;

x1 = p.second.first;

y1 = p.second.second;

draw\_line(x02, y02, x1, y1, 0);

}

}

PlotPixel(WINDOW, WINDOW);

}

void KeyboardEvent(unsigned char key, int x, int y)

{

switch (key)

{

case 'q':

longjmp(env, 1);

//exit(0);

break;

}

}

void draw\_main(struct Node\* root, int height, int argc, char\* argv[]) {

mainroot = root;

//glut functions

glutInit(&argc, argv); //initialize GLUT

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); //initialize display mode

glutInitWindowSize(WINDOW, WINDOW);

glutInitWindowPosition(50, 50); //Position of top-left corner

glutCreateWindow("Binary trees");

init();

glutDisplayFunc(draw);

glutKeyboardFunc(KeyboardEvent);

if (!setjmp(env))

glutMainLoop(); //display everything and wait

}

**Lab8.cpp**

#include <cstdlib>

#include <iostream>

#include <ctime>

#include <string>

#include <stdio.h>

#include "Point.h"

#include "node.h"

int main()

{

int n,k,j=0;

char s;

point\* a{};

point2\* b{};

node\* c{};

std::cout << "\n Enter n : ";

std::cin >> n;

do

{

std::cout << "\n 1.Forming list\n";

std::cout << "2.Print list\n";

std::cout << "3.Add element to list\n";

std::cout << "4.Forming 2way list\n";

std::cout << "5.Print 2way list\n";

std::cout << "6.Delete element from 2way list\n";

std::cout << "7.Forming Tree\n";

std::cout << "8.Print Tree\n";

std::cout << "9.Search in Tree\n";

std::cout << "10.Exit\n";

std::cin >> k;

switch (k)

{

case 1: a = make\_list(n); break;//выделение памяти и заполнение

case 2: print\_list(a); break;//печать

case 3:

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

if (i % 2 == 0)

{

a = add\_point(a, i);

n++;

}

}

break;

case 4: b = make\_list2(n); break;

case 5: print\_list(b); break;

case 6: popBack(b); break;

case 7: c=Tree(n, c); break;

case 8: print(c); break;

case 9:

std::cout << "\n Enter element for search : ";

char h;

std::cin >> h;

j=search(c,h);

std::cout << "Elementov " << h << " in Tree: " << j; break;

}

} while (k != 10);//выход

delete[] a;

delete[] b;

delete[] c;

}