МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: БДП

Студент гр. 8381 _____ Гречко В.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

Цель работы.

Ознакомиться с основными характеристиками и особенностями такой структуры данных, как бинарное дерево поиска, изучить особенности ее реализации на языке программирования C++. Разработать программу, использующую бинарное дерево поиска для удаления элемента и её визуализацию.

Задание.

По заданному файлу F (типа file of Elem), все элементы которого различны, построить структуру данных определённого типа – БДП: рандомизированная пирамида поиска.

Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem, и если входит, то удалить элемент e из структуры данных. Предусмотреть возможность повторного выполнения с другим элементом.

Основные теоретические положения.

При построении рандомизированного дерева основное исходное положение заключается в том, что любой элемент может с одинаковой вероятностью быть корнем дерева. Причем это справедливо и для поддеревьев рандомизированного дерева. Поэтому при включении нового элемента в дерево, случайным образом выбирается включение элемента в качестве листа, или в качестве корня.

Вероятность появления нового узла в корне дерева или поддерева определяется, как 1/(n+1), где n – число узлов в дереве или поддереве. То есть дерево выглядит так, будто элементы поступали в случайном порядке.

Поскольку принимаемые алгоритмом решения являются случайными, то при каждом выполнении алгоритма при одной и то же последовательности включений будут получаться деревья различной конфигурации. Таким образом,

несмотря на увеличение трудоемкости операции включений, за счет рандомизации получается структура дерева, близкая к сбалансированной.

Выполнение работы.

Написание работы производилось на базе операционной системы Сборка, отладка производились в QtCreator. Исходные коды файлов программы представлены в приложениях А-Ж.

Для реализации программы был разработан графический интерфейс с помощью встроенного в QtCreator UI-редактора. Он представляет из себя поле ввода, кнопку считывания, поле ввода для поиска и удаления элемента, а также поле вывода с возможностью графического отображения результата. Основные слоты для работы графического интерфейса приведены в табл. 1.

Таблица 1 – Слоты класса MainWindow и их назначение

Метод	Назначение
`on_printTree_clicked()`	Слот, отвечающий за считывание данных и графического вывода
`on_delete_elem_clicked()`	Слот, отвечающий за поиск и удаление элемента
`on_choose_file_clicked()`	Слот, отвечающий за считывание данных из файла

Для реализации бинарного дерева были созданы структуры узла Node и самого дерева BinTree, представленные на рис. 2.

Рисунок 2 – Структуры бинарного дерева и узла

Также были реализованы функции, создающие и изменяющие бинарное дерево, приведенные в табл. 2.

Таблица 2 – Основные функции работы с бинарным деревом

Функция	Назначение
`void BinTree()`	Создает пустое бинарное дерево
`Node* insert(Node* p, int k)`	Рандомизированная вставка нового узла с ключом k в дерево p
`int max_depth(Node *hd)`	Возвращает максимальную глубину дерева
`Node* insertroot(Node* p, int k)`	Вставка нового узла с ключом k в корень дерева p
`Node* rotateright(Node* p)`	Правый поворот вокруг узла p
`Node* rotateleft(Node* q)`	Левый поворот вокруг узла q
`int getsize(Node* p)`	Получение размера дерева
`void fixsize(Node* p)`	Установление корректного размера дерева
`Node* join(Node* p, Node* q)`	Объединение двух деревьев
`Node* remove(Node* p, int k)`	Удаление из дерева p первого найденного узла с ключом k

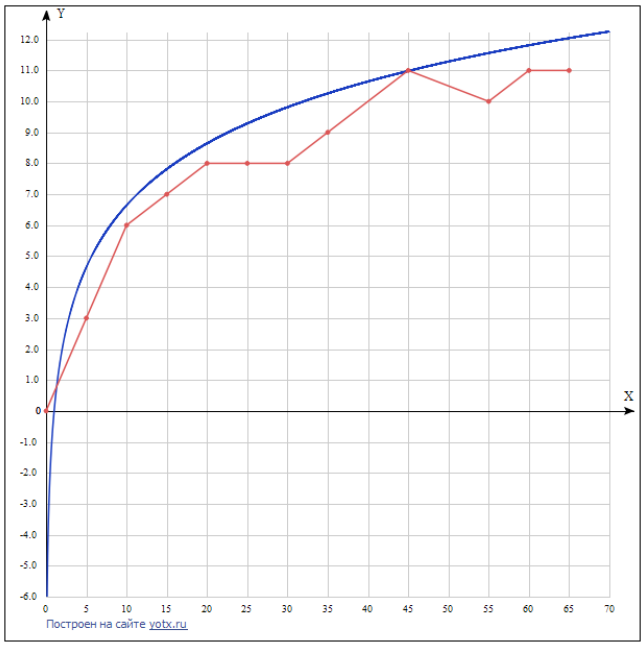
Программа имеет возможность графического отображения полученного бинарного дерева с помощью виджета QGraphicsView. Функции, необходимые для графического представления дерева, представлены в табл. 3.

Таблица 3 – Функции, связующие графический интерфейс и алгоритмы

Функция	Назначение
`QGraphicsScene *graphic(BinTree *tree, QGraphicsScene *&scene, int depth)`	По заданному бинарному дереву выполняет рисование в объекте
`int treePainter(QGraphicsScene *&scene, Node *node, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth)`	Рекурсивный алгоритм обхода дерева и рисования узлов в заданном объекте

Оценка сложности алгоритма.

Построенное дерево окажется неплохо сбалансированным: его высота будет порядка n.



Тестирование программы.

Вид программы после выполнения представлен на рис. 3.

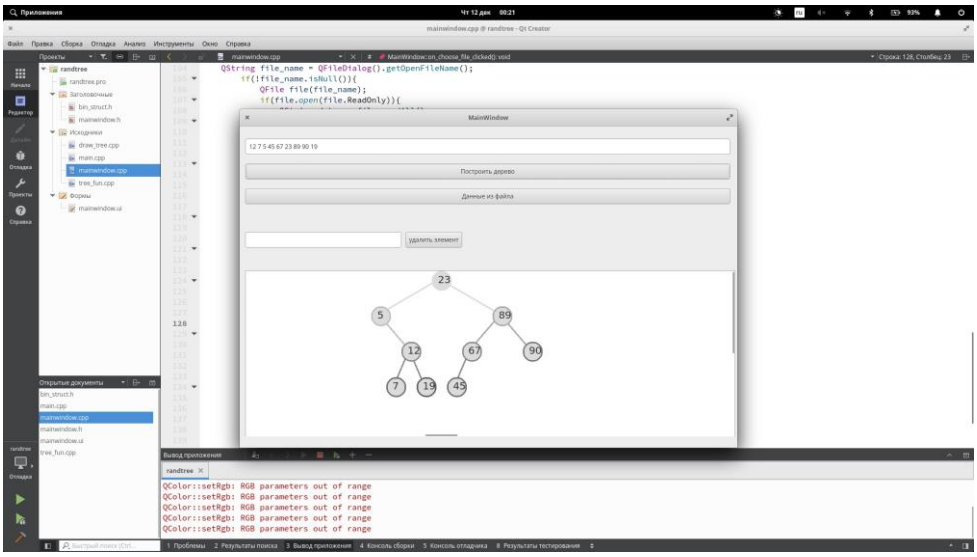


Рисунок 3.1 – Графический интерфейс программы

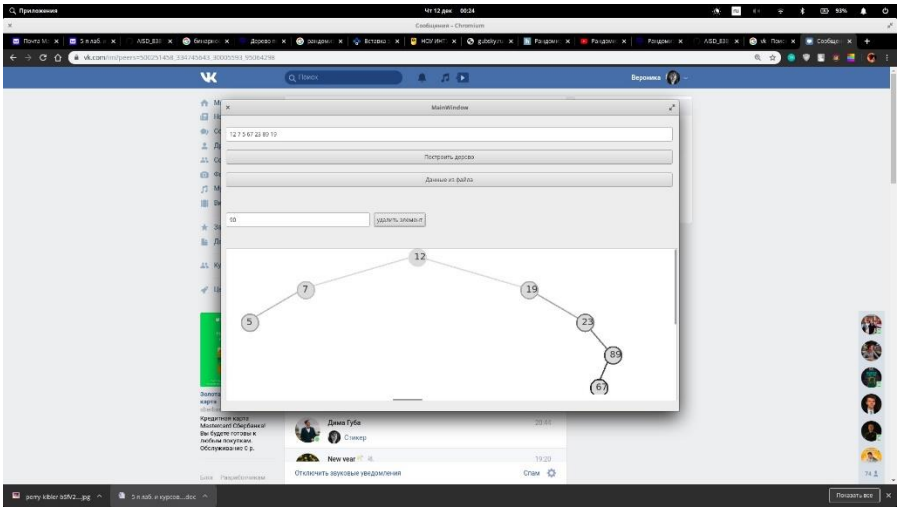


Рисунок 3.2 – Графический интерфейс программы

Также был рассмотрен случай некорректно введённых данных на рис. 4.

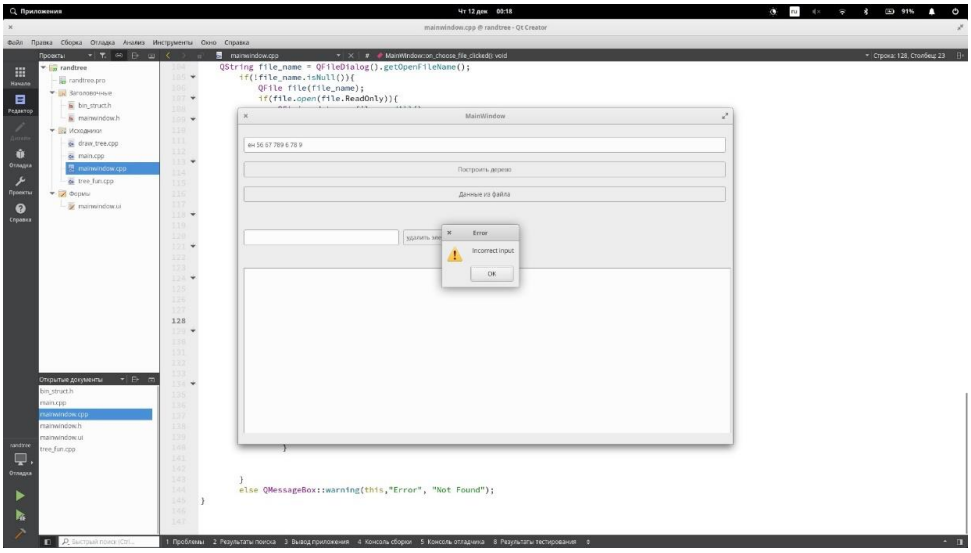


Рисунок 4 – Ошибка ввода

Выводы.

В ходе выполнения лабораторной работы была написана программа, создающая бинарное дерево поиска и удаляющая заданный элемент. Печать бинарного дерева выполняется графически.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN.CPP

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.H

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QGraphicsItem>
#include <QGraphicsView>
#include <QGraphicsEffect>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_printTree_clicked();

    void on_delete_elem_clicked();

    void on_choose_file_clicked();


private:
    Ui::MainWindow *ui;
    QGraphicsScene *scene;
};

#endif // MAINWINDOW_H
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.CPP

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include<bin_struct.h>
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    scene = new QGraphicsScene;
    ui->graphicsView->setScene(scene);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_printTree_clicked()
{

    QString data = ui->input_tree->text();
    QStringList abc = data.split(' ');
    int* mas = new int[100];
    int i =0;
    bool go = true;
    for (auto x:abc){
         bool convertOK;
         x.toInt(&convertOK);
         if(convertOK == false){
             go = false;

         }
         else{
            mas[i] = x.toInt();
            i++;
         }
    }
    if(!go){
        QMessageBox::warning(this,"Error", "Incorrect input");
        return;
    }
    BinTree* BT = new (BinTree);
    for(int j = 0; j < i; j++){
        BT->Head = BT->insert(BT->Head, mas[j]);
    }

    int depth = BT->max_depth(BT->Head);
    graphic(BT, scene, depth);

}
```

```
void MainWindow::on_delete_elem_clicked()
{
    QString data = ui->input_tree->text();
    QStringList abc = data.split(' ');
    int* mas = new int[100];
    int i =0;
    bool go = true;
    for (auto x:abc){
         bool convertOK;
         x.toInt(&convertOK);
         if(convertOK == false){
             go = false;

         }
         else{
            mas[i] = x.toInt();
            i++;
         }
    }
    if(!go){
        QMessageBox::warning(this,"Error", "Incorrect input");
        return;
    }
    BinTree* BT = new (BinTree);
    for(int j = 0; j < i; j++){
        BT->Head = BT->insert(BT->Head, mas[j]);
    }
    QString elem_ = ui->input_del->text();
    bool convert_;
    int elem = elem_.toInt(&convert_);
    if(!convert_){
        QMessageBox::warning(this,"Error", "Incorrect input");
        return;
    }
    BT->Head = BT->remove(BT->Head, elem);
    int depth = BT->max_depth(BT->Head);
    QString out;
    for(int l = 0; l < i - 1; l++){
        if (mas[l] == elem){
            for(int k = l; k < i - 1; k++){
                mas[k] = mas[k + 1];
            }
        }
        out.append(QString::number(mas[l]));
        if(l != i -2) out.append(" ");
    }
    ui->input_tree->setText(out);
    graphic(BT, scene, depth);
}

void MainWindow::on_choose_file_clicked()
{
    QString file_name = QFileDialog().getOpenFileName();
        if(!file_name.isNull()){
            QFile file(file_name);
            if(file.open(file.ReadOnly)){
```

```
                QString data  = file.readAll();
                 if (data == "") {
                     QMessageBox::critical(this, "Error!",
"P’PIPµPrPëC,Pµ PrPµCЂPµPIPs");
                     return;
                 }
                 else{
                     QStringList abc = data.split(' ');
                     int* mas = new int[100];
                     int i =0;
                     bool go = true;
                     for (auto x:abc){
                          bool convertOK;
                          x.toInt(&convertOK);
                         if(convertOK == false){
                              go = false;
                          }
                          else{
                            mas[i] = x.toInt();
                             i++;
                           }
                     }
                     if(!go){
                             QMessageBox::warning(this,"Error", "Incor-
rect input");
                             return;
                     }
                     BinTree* BT = new (BinTree);
                     for(int j = 0; j < i; j++){
                         BT->Head = BT->insert(BT->Head, mas[j]);
                     }
                     int depth = BT->max_depth(BT->Head);
                     graphic(BT, scene, depth);
                     }
                 }


        }
        else QMessageBox::warning(this,"Error", "Not Found");
}
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ПРОГРАММЫ. BIN_STRUCT.H

```
#ifndef BIN_STRUCT_H
#define BIN_STRUCT_H
#include <QGraphicsItem>
#include <QGraphicsView>
#include <QGraphicsEffect>
#include <QString>
#include <QFileDialog>
#include <QMessageBox>
#include <QTextEdit>
#include <QMainWindow>
#include <QStandardPaths>
#include <QtGui>
#include <QColorDialog>
#include <QInputDialog>
#include <QPushButton>
#include <QStringList>

struct Node // CЃC,CЂCѓPєC,CѓCЂP° PґP»CЏ PїCЂPµPґCЃC,P°PIP»PµPSPёCЏ
CѓP·P»PsPI PґPµCЂPµPIP°
{
    int key;
    int size;
    Node* left;
    Node* right;
    Node(int k) { key = k; left = right = 0; size = 1; }

};


class BinTree
{
private:
    Node* Current = nullptr;
public:
    Node* Head = nullptr;
    BinTree();
    Node* insert(Node* p, int k);
    Node* insertroot(Node* p, int k);
    Node* rotateright(Node* p);
    Node* rotateleft(Node* q);
    int max_depth(Node *hd);
    int getsize(Node* p);
    void fixsize(Node* p);
    Node* join(Node* p, Node* q);
    Node* remove(Node* p, int k);

};
int treePainter(QGraphicsScene *scene, Node *node, int w, int h, int
wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth,
int c);
```

```
QGraphicsScene *graphic(BinTree *tree, QGraphicsScene *scene, int
depht);
#endif // BIN_STRUCT_H
```

ПРИЛОЖЕНИЕ Д

ИСХОДНЫЙ КОД ПРОГРАММЫ. TREE_FUN.CPP

```
#include <bin_struct.h>

BinTree::BinTree(){
    Head = nullptr;
    Current = Head;
}
int BinTree::max_depth(Node *hd){
    if (hd == NULL) return 0;
    else{
        int lDepth = max_depth(hd->left);
        int rDepth = max_depth(hd->right);

        if (lDepth > rDepth)  return(lDepth + 1);
        else return(rDepth + 1);
    }
}
int BinTree::getsize(Node* p) // PsP±PµCЪC,PєP° PґP»CЦ PїPsP»CЦ size,
CЪP°P±PsC,P°PµC, CЃ PїCґCЃC,C‹PjPë PґPµCЪPµPICЊCЦPjPë (t=NULL)
{
    if( !p ) return 0;
    return p->size;
}

void BinTree::fixsize(Node* p) // CґCЃC,P°PSPsPIP»PµPSPëPµ
PєPsCЪCЪPµPєC,PSPsPiPs CЪP°P·PjPµCЪP° PґPµCЪPµPIP°
{
    p->size = getsize(p->left)+getsize(p->right)+1;
}

 Node* BinTree::rotateright(Node* p) // PїCЪP°PIC‹P№ PїPsPIPsCЪPsC,
PIPsPєCЪCґPi CґP·P»P° p
{
    Node* q = p->left;
    if( !q ) return p;
    p->left = q->right;
    q->right = p;
    q->size = p->size;
    fixsize(p);
    return q;
}
Node* BinTree::rotateleft(Node* q) // P»PµPIC‹P№ PїPsPIPsCЪPsC, PIPsPєCЪCґPi
CґP·P»P° q
{
    Node* p = q->right;
    if( !p ) return q;
    q->right = p->left;
    p->left = q;
    p->size = q->size;
    fixsize(q);
    return p;
}
Node* BinTree::insertroot(Node* p, int k) // PICЃC,P°PIPєP° PSPsPIPsPiPs
CґP·P»P° CЃ PєP»CЂC‡PsPj k PI PєPsCЪPµPSCЊ PґPµCЪPµPIP° p
{
    if( !p ) return new Node(k);
    if( k < p->key )
    {
```

```
        p->left = insertroot(p->left,k);
        return rotateright(p);
    }
    else
    {
        p->right = insertroot(p->right,k);
        return rotateleft(p);
    }
}
Node* BinTree::insert(Node* p, int k) // CЂP°PSPґPsPjPёP·PёCЂPsPIP°PSPSP°CЏ
PICЃC,P°PIPєP° PSPsPIPsPiPs CґP·P»P° CЃ PєP»CЂC‡PsPj k PI PґPµCЂPµPIPs p
{
    if( !p ) return new Node(k);
    if( rand()%(p->size+1)==0 )
        return insertroot(p,k);
    if( p->key>k )
        p->left = insert(p->left,k);
    else
        p->right = insert(p->right,k);
    fixsize(p);
    return p;
}
Node*BinTree:: join(Node* p, Node* q) // PsP±CЉPµPґPёPSPµPSPёPµ PґPICґC…
PґPµCЂPµPICЊPµPI
{
    if( !p ) return q;
    if( !q ) return p;
    if( rand()%(p->size+q->size) < p->size )
    {
        p->right = join(p->right,q);
        fixsize(p);
        return p;
    }
    else
    {
        q->left = join(p,q->left);
        fixsize(q);
        return q;
    }
}
Node*BinTree::remove(Node* p, int k) // CґPґP°P»PµPSPёPµ PёP· PґPµCЂPµPIP° p
PїPµCЂPIPsPiPs PSP°P№PґPµPSPSPsPiPs CґP·P»P° CЃ PєP»CЂC‡PsPj k
{
    if( !p ) return p;
    if( p->key==k )
    {
        Node* q = join(p->left,p->right);
        delete p;
        return q;
    }
    else if( k<p->key )
        p->left = remove(p->left,k);
    else
        p->right = remove(p->right,k);
    return p;
}
```

ПРИЛОЖЕНИЕ Е

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN_FUN.H

```
#ifndef MAIN_FUN_H
#define MAIN_FUN_H
#include <bin_struct.h>


int treePainter(QGraphicsScene *scene, Node *node, int w, int h, int
wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth);

QGraphicsScene *graphic(BinTree *tree, QGraphicsScene *scene, int
depht);

#endif // MAIN_FUN_H
```

ПРИЛОЖЕНИЕ Ж

ИСХОДНЫЙ КОД ПРОГРАММЫ. DRAW_TREE.CPP

```
#include<bintree.h>
#include<functionstree.h>
#include<cmath>
QGraphicsScene *graphic(BinTree *tree, QGraphicsScene *&scene, int depth)
{
    if (tree == nullptr)
        return scene;
    scene->clear();
    QPen pen;
    QColor color;
    color.setRgb(220, 220, 220);
    pen.setColor(color);
    QBrush brush (color);
    QFont font;
    font.setFamily("Tahoma");
    pen.setWidth(3);
    int wDeep = static_cast<int>(pow(2, depth + 2));
    int hDelta = 70;
    int wDelta = 15;
    font.setPointSize(wDelta);
    int width = (wDelta*wDeep)/2;
    treePainter(scene, tree->Head, width/2, hDelta, wDelta, hDelta, pen, brush,
font, wDeep);
    return scene;
}

int treePainter(QGraphicsScene *&scene, Node *node, int w, int h, int wDelta,
int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth)
{
    if ((node == nullptr) || (node->data == '^'))
        return 0;
    QString out;
    out += node->data;
    QGraphicsTextItem *textItem = new QGraphicsTextItem;
    textItem->setPos(w, h);
    textItem->setPlainText(out);
    textItem->setFont(font);
    scene->addEllipse(w-wDelta/2, h, wDelta*5/2, wDelta*5/2, pen, brush);
    if ((node->left != nullptr) && (node->left->data != '^') )
        scene->addLine(w+wDelta/2, h+wDelta, w-(depth/2)*wDelta+wDelta/2,
h+hDelta+wDelta, pen);
    if (node->right != nullptr)
        scene->addLine(w+wDelta/2, h+wDelta, w+(depth/2)*wDelta+wDelta/2,
h+hDelta+wDelta, pen);
    scene->addItem(textItem);
    treePainter(scene, node->left, w-(depth/2)*wDelta, h+hDelta, wDelta, hDelta,
pen, brush, font, depth/2);
    treePainter(scene, node->right, w+(depth/2)*wDelta, h+hDelta, wDelta,
hDelta, pen, brush, font, depth/2);
    return 0;
}
```