Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Кафедра **«**Информационные технологии и автоматизированные системы**»**

**ОТЧЕТ**

**ПО ТВОРЧЕСКОЙ РАБОТЕ №1**

Дисциплина: «Основы алгоритмизации и программирования»

Семестр 2

Тема: АРМ разработчика компьютерной игры

Выполнил работу

Студент группы РИС-22-1Б

Поважный В. Е.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

г. Пермь-2023

**Введение**

Для выполнения лабораторной работы требуется сделать отчет программы. Создание программ – отличный способ практики программирования.

**Постановка задачи**

Реализация приложения на основе Windows Forms, облегчающего работу разработчика. Распаковка и запаковка архивов с данными/конвертирование 3d моделей/осмотр 3d моделей.

**Код программы**

**mk64n.h**

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <cstring>

#include <iomanip>

#include <windows.h>

#include <shlwapi.h>

#include <vector>

#include <filesystem>

namespace fs = std::filesystem;

public ref class inf2obj

{

public:

bool block = false;

float\* vertices = NULL;

int num\_triangles = 0;

int num\_vertices = 0;

int\* verTri = NULL;

std::string\* path;

inf2obj()

{

num\_triangles = 0;

num\_vertices = 0;

vertices = NULL;

verTri = NULL;

}

~inf2obj()

{

delete[] vertices;

delete[] verTri;

}

void Reset()

{

num\_triangles = 0;

num\_vertices = 0;

vertices = NULL;

verTri = NULL;

}

std::string openfilename();

bool openFile(char\*& f, size\_t& fsize, std::string filename);

unsigned long readlong(char\*& f, size\_t& pos);

std::string getFilenamePath(const std::string str);

std::string getFilenameFile(const std::string str);

int Convert();

};

struct fmtINF

{

public:

uint32\_t num\_vertices = 0;

uint32\_t num\_triangles = 0;

float\* vertices;

float\* tvertices;

float\* normals;

uint16\_t\* nor\_triangles;

int\* ver\_triangles;

uint32\_t\* tex\_triangles;

fmtINF()

{

vertices = NULL;

tvertices = NULL;

normals = NULL;

nor\_triangles = NULL;

ver\_triangles = NULL;

tex\_triangles = NULL;

}

~fmtINF()

{

if (vertices = NULL) delete[] vertices;

if (tvertices = NULL) delete[] tvertices;

if (normals = NULL) delete[] normals;

if (nor\_triangles = NULL) delete[] nor\_triangles;

if (ver\_triangles = NULL) delete[] ver\_triangles;

if (tex\_triangles = NULL) delete[] tex\_triangles;

}

unsigned short readshort(char\*& f, size\_t& pos);

unsigned long readlong(char\*& f, size\_t& pos);

float readfloat(char\*& f, size\_t& pos);

bool openFile(char\*& f, size\_t& fsize, std::string filename);

bool readVer(std::string fpath, std::string fname);

void write\_obj(std::string file);

int\* GetVerteciesTriangles();

};

struct fmtTRI

{

public:

uint32\_t num\_triangles = 0;

uint16\_t\* nor\_triangles;

int\* ver\_triangles;

uint32\_t\* tex\_triangles;

fmtTRI()

{

nor\_triangles = NULL;

ver\_triangles = NULL;

tex\_triangles = NULL;

}

~fmtTRI()

{

if (nor\_triangles = NULL) delete[] nor\_triangles;

if (ver\_triangles = NULL) delete[] ver\_triangles;

if (tex\_triangles = NULL) delete[] tex\_triangles;

}

unsigned short readshort(char\*& f, size\_t& pos);

unsigned long readlong(char\*& f, size\_t& pos);

bool openFile(char\*& f, size\_t& fsize, std::string filename);

bool readTri(std::string fpath, std::string fname);

};

**Undat.h**

**#include** <stdio.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <cstring>

#include <iomanip>

#include <windows.h>

#include <shlwapi.h>

#include <vector>

#include <filesystem>

namespace fs = std::filesystem;

extern int \_progress;

extern int endPoint;

public ref class Unpacker

{

public:

bool createBackup;

bool changeStartScreen;

std::string\* \_currentPath;

Unpacker()

{

\_progress = 0;

endPoint = 0;

createBackup = true;

changeStartScreen = true;

}

~Unpacker()

{

delete \_currentPath;

}

public:

bool OpenFile(char\*& f, size\_t& fsize, std::string filename);

void SetPath(std::string path);

void Extract(std::string name);

void PackBack(std::string name);

};

struct DAT

{

public:

uint32\_t num\_folders = 0;

uint32\_t num\_files = 0;

uint32\_t\* folderID;

uint32\_t\* fileFolderID;

uint32\_t\* fileFolderIDTemp;

size\_t\* fileOffset;

uint32\_t\* fileSize;

uint32\_t\* fileTSTAMPfst;

uint32\_t\* fileTSTAMPsnd;

std::string\* folderName;

std::string\* fileName;

std::string\* currentPath;

std::vector<std::string> filesPath;

DAT(std::string path)

{

currentPath = new std::string{ path };

folderID = NULL;

folderName = NULL;

fileName = NULL;

fileOffset = NULL;

fileSize = NULL;

fileTSTAMPfst = NULL;

fileTSTAMPsnd = NULL;

fileFolderID = NULL;

fileFolderIDTemp = NULL;

}

~DAT()

{

if (folderID = NULL) delete[] folderID;

if (folderName = NULL) delete[] folderName;

if (fileName = NULL) delete[] fileName;

if (fileOffset = NULL) delete[] fileOffset;

if (fileSize = NULL) delete[] fileSize;

if (fileTSTAMPfst = NULL) delete[] fileTSTAMPfst;

if (fileTSTAMPsnd = NULL) delete[] fileTSTAMPsnd;

if (fileFolderID = NULL) delete[] fileFolderID;

if (fileFolderIDTemp = NULL) delete[] fileFolderIDTemp;

}

std::string ReadStringWithout(char\*& f, size\_t& pos, int size);

std::string ReadString(char\*& f, size\_t& pos, int size);

std::string WriteString(int size, std::string str);

std::string WriteLong(int num);

unsigned long readlong(char\*& f, size\_t& pos);

bool OpenFile(char\*& f, size\_t& fsize, std::string filename);

bool ReadAndWriteFolders(char\*& f, size\_t& pos, size\_t& fsize, std::string n);

void WriteFiles(char\*& f, size\_t& pos, size\_t& fsize);

void CreateArchive(std::string n, bool cBackup, bool cStartScreen);

};

**obj2inf.h**

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <cstring>

#include <iomanip>

#include <windows.h>

#include <shlwapi.h>

#include <vector>

#include <filesystem>

namespace fs = std::filesystem;

namespace otoi

{

public ref class obj2inf

{

public:

bool autoTex;

int facesNum = 0;

int verNum = 0;

int tverNum = 0;

int norNum = 0;

int texture = 0;

float\* vertices;

float\* tvertices;

float\* normals;

int\* verTri;

std::vector<std::string>\* polygons;

std::string\* path;

obj2inf()

{

polygons = new std::vector<std::string>;

verTri = NULL;

vertices = NULL;

tvertices = NULL;

normals = NULL;

}

~obj2inf()

{

delete[] vertices;

delete[] tvertices;

delete[] normals;

delete[] verTri;

delete polygons;

DeleteAllFiles();

}

void Reset(bool autoTex, int tex)

{

delete polygons;

polygons = new std::vector<std::string>;

verTri = NULL;

vertices = NULL;

tvertices = NULL;

normals = NULL;

facesNum = 0;

verNum = 0;

tverNum = 0;

norNum = 0;

texture = tex;

this->autoTex = autoTex;

}

void Convert()

{

std::string fileName;

std::string filePath;

filePath = OpenFileName();

fileName = GetFilenameFile(filePath);

if (!fs::exists(\*path + "\\Converted"))

fs::create\_directory(\*path + "\\Converted");

if (filePath != "")

{

ObjParse(filePath);

writeINF(fileName);

writeVER(fileName);

writeTRI(texture, fileName);

if (norNum > 0)

writeNOR(fileName);

DeleteAllFiles();

}

}

private:

std::string OpenFileName()

{

std::string filename(MAX\_PATH, '\0');

OPENFILENAME ofn = { };

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = NULL;

ofn.lpstrFilter = "Wavefront Files (\*.obj)\0\*.obj\0All Files (\*.\*)\0\*.\*\0";

ofn.lpstrFile = &filename[0];

ofn.nMaxFile = MAX\_PATH;

ofn.lpstrTitle = "Select a File";

ofn.Flags = OFN\_EXPLORER | OFN\_FILEMUSTEXIST | OFN\_HIDEREADONLY;

if (!GetOpenFileName(&ofn))

return "";

return filename;

}

std::string GetFilenameFile(const std::string str)

{

size\_t found;

std::string strt;

found = str.find\_last\_of("/\\");

if (found < str.size())

{

strt = str.substr(found + 1, -1);

found = strt.find(".");

if (found < strt.size())

strt = strt.substr(0, found);

}

else strt = str;

size\_t lastdot = strt.find\_last\_of(".");

if (lastdot == std::string::npos) return strt;

return strt.substr(0, lastdot);

}

void VandVN(std::string line, std::vector<float>& x, std::vector<float>& y, std::vector<float>& z)

{

int num = 1;

int i = 0;

std::string number = "";

if (line[1] == 32)

i = 2;

else

i = 3;

for (i; i < line.length(); i++)

{

switch (num)

{

case 1:

if (line[i] != 32)

number += line[i];

else

{

num++;

x.push\_back(stof(number));

number = "";

}

break;

case 2:

if (line[i] != 32)

number += line[i];

else

{

num++;

y.push\_back(stof(number));

number = "";

}

break;

case 3:

if (i != line.length() - 1)

number += line[i];

else

{

number += line[i];

z.push\_back(stof(number));

number = "";

}

break;

}

}

}

void VT(std::string line, std::vector<float>& x, std::vector<float>& y)

{

int num = 1;

std::string number = "";

for (int i = 3; i < line.length(); i++)

{

switch (num)

{

case 1:

if (line[i] != 32)

number += line[i];

else

{

num++;

x.push\_back(stof(number));

number = "";

}

break;

case 2:

if (i != line.length() - 1)

number += line[i];

else

{

number += line[i];

y.push\_back(stof(number));

number = "";

}

break;

}

}

}

void F(std::string line, std::vector<std::string>& x, std::vector<std::string>& y, std::vector<std::string>& z, std::vector<int>& t, int tex)

{

int num = 1;

int sl = 0;

std::string number = "";

for (int i = 2; i < line.length(); i++)

{

switch (num)

{

case 1:

if (line[i] != 32)

{

if (line[i] == 47)

sl++;

if (sl < 2)

number += line[i];

}

else

{

num++;

x.push\_back(number);

number = "";

sl = 0;

}

break;

case 2:

if (line[i] != 32)

{

if (line[i] == 47)

sl++;

if (sl < 2)

number += line[i];

}

else

{

num++;

y.push\_back(number);

number = "";

sl = 0;

}

break;

case 3:

if (i != line.length() - 1)

{

if (line[i] == 47)

sl++;

if (sl < 2)

number += line[i];

}

else

{

z.push\_back(number);

number = "";

sl = 0;

}

break;

}

}

t.push\_back(tex);

}

int US(std::string line, int tex)

{

if (autoTex)

{

std::string number = "";

for (int i = 7; i < line.length(); i++)

number += line[i];

tex = std::stoi(number);

}

return tex;

}

void ObjParse(std::string path)

{

std::ifstream obj(path);

std::vector<float> vx;

std::vector<float> vy;

std::vector<float> vz;

std::vector<float> vnx;

std::vector<float> vny;

std::vector<float> vnz;

std::vector<float> vtx;

std::vector<float> vty;

std::vector<std::string> fx;

std::vector<std::string> fy;

std::vector<std::string> fz;

std::vector<int> ft;

std::string line = "";

while (getline(obj, line))

{

if (line != "")

{

int type = line[0] + line[1];

switch (type)

{

//v

case 150:

VandVN(line, vx, vy, vz);

break;

//vt

case 234:

VT(line, vtx, vty);

break;

//vn

case 228:

VandVN(line, vnx, vny, vnz);

break;

//f

case 134:

F(line, fx, fy, fz, ft, texture);

break;

//us

case 232:

texture = US(line, texture);

break;

}

}

}

std::ofstream verFile(this->path[0] + "\\Vertex.txt");

for (int i = 0; i < vx.size(); i++)

verFile << vx[i] << std::endl

<< vy[i] << std::endl

<< vz[i] << std::endl;

verFile.close();

std::ofstream tverFile(this->path[0] + "\\Texture.txt");

for (int i = 0; i < vtx.size(); i++)

tverFile << vtx[i] << std::endl

<< vty[i] << std::endl;

tverFile.close();

if (vnx.size() > 0)

{

std::ofstream norFile(this->path[0] + "\\Normal.txt");

for (int i = 0; i < vnx.size(); i++)

norFile << vnx[i] << std::endl

<< vny[i] << std::endl

<< vnz[i] << std::endl;

norFile.close();

}

std::ofstream faceFile(this->path[0] + "\\Faces.txt");

for (int i = 0; i < fx.size(); i++)

faceFile << fx[i] << std::endl

<< fy[i] << std::endl

<< fz[i] << std::endl

<< ft[i] << std::endl;

faceFile.close();

obj.close();

}

int GetSymbolsCount(char sym, std::string str)

{

int c = 0;

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

if (str[i] == sym)

c++;

return c;

}

std::string writeShort(int num)

{

int arr[] = { 0, 0 };

while (num >= 256)

{

arr[1] += 1;

num -= 256;

}

arr[0] = num;

std::string hex;

for (int i = 0; i < 2; i++)

hex += char(arr[i]);

return hex;

}

std::string WriteLong(int num)

{

std::string Hex = "";

std::stringstream s;

s << std::hex << num;

int zeroes = 8 - size(s.str());

for (int i = 0; i < zeroes; i++)

Hex += "0";

Hex += s.str();

int arr[] = { 0, 0, 0, 0 };

int c = 3;

for (int i = 0; i < Hex.length() - 1; i += 2)

{

std::stringstream h;

h << Hex[i] << Hex[i + 1];

h >> std::hex >> arr[c];

c--;

}

Hex = "";

for (int i = 0; i < 4; i++)

Hex += char(uint8\_t(arr[i]));

return Hex;

}

std::string writeFloatLong(float num)

{

std::string strHEX = "";

unsigned long a = 0;

unsigned long a24 = 0;

float n = num;

memcpy(&a, &n, 4);

strHEX += char(uint8\_t(a));

strHEX += char((uint16\_t(a) - uint8\_t(a)) / 0x00000100);

memcpy(&a24, &n, 3);

strHEX += char((a24 - int(uint16\_t(a))) / 0x00010000);

strHEX += char((a - a24) / 0x01000000);

return strHEX;

}

std::string changeSymbol(std::string str, char ch1, char ch2)

{

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

if (str[i] == ch1) str[i] = ch2;

return str;

}

void writeINF(std::string name)

{

// read

std::string line;

std::ifstream facesFile(this->path[0] + "\\Faces.txt");

std::ifstream verFile(this->path[0] + "\\Vertex.txt");

std::ifstream tverFile(this->path[0] + "\\Texture.txt");

std::ifstream norFile(this->path[0] + "\\Normal.txt");

while (getline(facesFile, line))

{

(\*polygons).push\_back(line);

facesNum++;

}

while (getline(verFile, line))

verNum++;

while (getline(tverFile, line))

tverNum++;

while (getline(norFile, line))

norNum++;

verNum /= 3;

facesNum /= 4;

tverNum /= 2;

norNum /= 3;

norFile.close();

facesFile.close();

verFile.close();

tverFile.close();

// write

std::ofstream infFile(this->path[0] + "\\Converted\\" + name + ".inf", std::ios\_base::binary);

infFile << WriteLong(tverNum) << WriteLong(facesNum);

infFile.close();

}

void writeVER(std::string name)

{

//read

vertices = new float[verNum \* 3];

tvertices = new float[tverNum \* 2];

std::ifstream vertexFile(this->path[0] + "\\Vertex.txt");

std::ifstream textureFile(this->path[0] + "\\Texture.txt");

std::string str = "";

for (int i = 0; i < verNum; i++)

{

getline(vertexFile, str);

vertices[i \* 3] = stof(changeSymbol(str, ',', '.'));

getline(vertexFile, str);

vertices[(i \* 3) + 1] = stof(changeSymbol(str, ',', '.'));

getline(vertexFile, str);

vertices[(i \* 3) + 2] = stof(changeSymbol(str, ',', '.'));

}

for (int i = 0; i < tverNum; i++)

{

getline(textureFile, str);

tvertices[i \* 2] = stof(changeSymbol(str, ',', '.'));

getline(textureFile, str);

tvertices[(i \* 2) + 1] = stof(changeSymbol(str, ',', '.'));

}

vertexFile.close();

//write

std::ofstream file(this->path[0] + "\\Converted\\" + name + ".ver", std::ios\_base::binary);

for (int i = 0; i < tverNum; i++)

file << writeFloatLong(vertices[i \* 3])

<< writeFloatLong(vertices[(i \* 3) + 1])

<< writeFloatLong(vertices[(i \* 3) + 2])

<< writeFloatLong(tvertices[i \* 2])

<< writeFloatLong(tvertices[(i \* 2) + 1]);

file.close();

}

void writeTRI(int tex, std::string name)

{

//read

std::vector<int> verTriangles;

std::vector<int> norTriangles;

std::vector<int> texTriangles;

verTri = new int[facesNum \* 3];

for (int i = 0; i < facesNum \* 4; i++)

{

std::string tempFace = "";

if (GetSymbolsCount('/', (\*polygons)[i]) > 0)

{

for (int k = 0; k < size((\*polygons)[i]); k++)

{

if ((\*polygons)[i][k] != '/')

tempFace += (\*polygons)[i][k];

else

{

verTriangles.push\_back(stoi(tempFace) - 1);

tempFace = "";

for (int j = k + 1; j <= size((\*polygons)[i]); j++)

tempFace += (\*polygons)[i][j];

norTriangles.push\_back(stoi(tempFace) - 1);

tempFace = "";

break;

}

}

}

else

texTriangles.push\_back(stoi((\*polygons)[i]));

}

for (int i = 0; i < facesNum \* 3; i++)

verTri[i] = verTriangles[i];

//write

std::ofstream file(this->path[0] + "\\Converted\\" + name + ".tri", std::ios\_base::binary);

for (int i = 0; i < facesNum; i++)

{

file << writeShort(verTriangles[i \* 3]) << writeShort(norTriangles[i \* 3])

<< writeShort(verTriangles[(i \* 3) + 2]) << writeShort(norTriangles[(i \* 3) + 2])

<< writeShort(verTriangles[(i \* 3) + 1]) << writeShort(norTriangles[(i \* 3) + 1]);

if (autoTex)

file << writeShort(texTriangles[i]) << writeShort(8192);

else

file << writeShort(tex) << writeShort(8192);

}

file.close();

}

void writeNOR(std::string name)

{

//read

normals = new float[norNum \* 3];

std::ifstream normalsFile(this->path[0] + "\\Normal.txt");

std::string str = "";

for (int i = 0; i < norNum; i++)

{

getline(normalsFile, str);

normals[i \* 3] = stof(changeSymbol(str, ',', '.'));

getline(normalsFile, str);

normals[(i \* 3) + 1] = stof(changeSymbol(str, ',', '.'));

getline(normalsFile, str);

normals[(i \* 3) + 2] = stof(changeSymbol(str, ',', '.'));

}

normalsFile.close();

//write

std::ofstream file(this->path[0] + "\\Converted\\" + name + ".nor", std::ios\_base::binary);

for (int i = 0; i < norNum; i++)

file << writeFloatLong(normals[i \* 3])

<< writeFloatLong(normals[(i \* 3) + 1])

<< writeFloatLong(normals[(i \* 3) + 2]);

file.close();

}

bool FileIsExist(std::string path)

{

bool isExist = false;

std::ifstream fin(path.c\_str());

if (fin.is\_open())

isExist = true;

fin.close();

return isExist;

}

void DeleteAllFiles()

{

std::string pth = (\*path) + "\\Vertex.txt";

remove(pth.c\_str());

pth = (\*path) + "\\Faces.txt";

remove(pth.c\_str());

pth = (\*path) + "\\Texture.txt";

remove(pth.c\_str());

pth = (\*path) + "\\Normal.txt";

if (FileIsExist(pth))

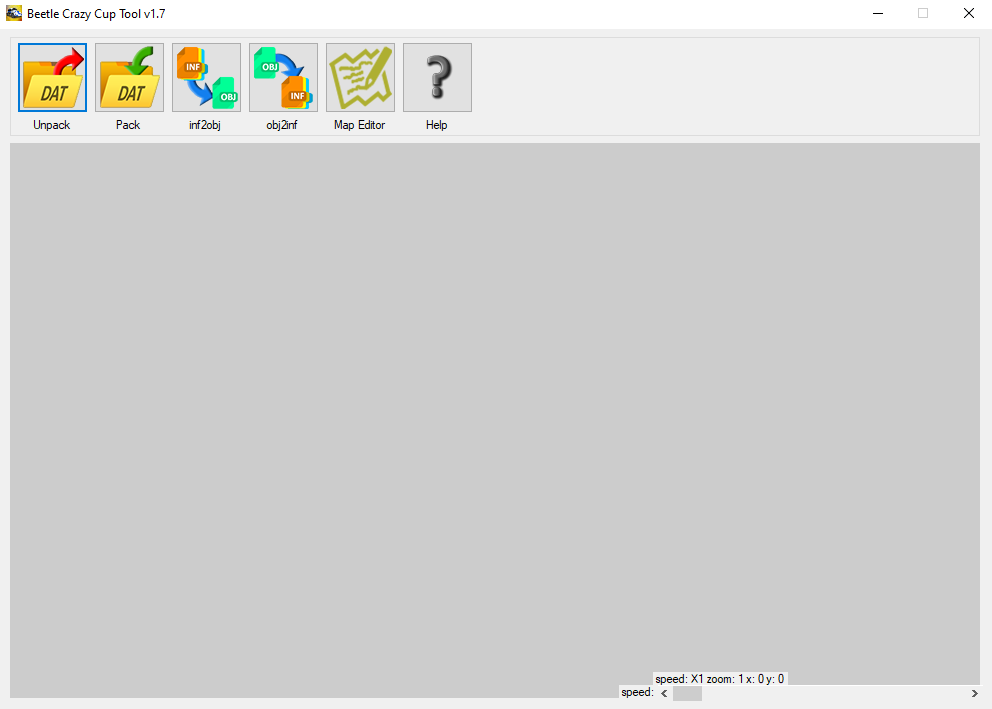
remove(pth.c\_str());

}

};

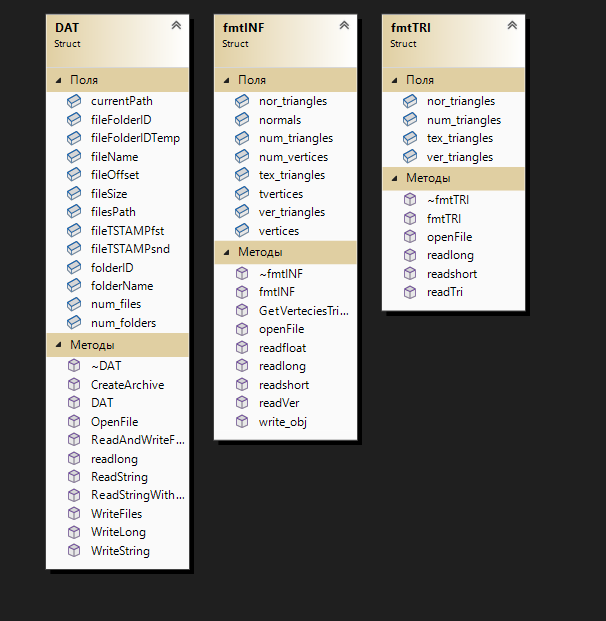
}

**Вывод программы**

****

***Рисунок 1 – вывод программы***

**UML диаграмма**

****

***Рисунок 2 – UML диаграмма***

**Вывод**

Программа выполняет свою задачу.

**Заключение**

Для решения задачи потребовались знания языка программирования, а конкретнее C++. Программа выполняет те условия, что были указаны в постановке задачи и работает без проблем.