Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машинных

Дисциплина: Конструирование программ и языки программирования

КОД ПРОГРАММЫ

к курсовому проекту

на тему

«Расписание городского транспорта»

БГУИР КП 1-40 02 01 325 КП

Студент: гр. 950503 Сякачёв П.В.

Руководитель: старший преподаватель кафедры ЭВМ Ковальчук А.М.

Минск 2020

**Модуль main**

“main.cpp”

#include "Interface.h"

int main()

{

Interface menu;

menu.Choose();

}

**Модуль Interface**

“Interface.h”

#pragma once

#include "Header.h"

#include <conio.h>

using namespace std;

class Interface

{

vector<StopsInfo> gid, uid;

Ring<Underground\_Stops> us;

Ring<Ground\_Stops> gs;

Ring<Underground\_Trans> train;

Ring<Ground\_Trans> bus;

Ring<Ground\_Trans> trolleybus;

Ring<Ground\_Trans> tram;

public:

Interface()

{

open();

}

~Interface() {}

void open();

void save();

void Choose();

void Change();

void Add();

void ChangeSmth();

void ChangeStops();

void ChangeTrans();

void Stops();

void GrStops();

void UndStops();

void Trans();

void GrTrans();

void UndTrans();

};

“Interface.cpp”

#include "Interface.h"

void Interface::open()

{

gid.clear();

uid.clear();

us.Delete();

gs.Delete();

train.Delete();

bus.Delete();

trolleybus.Delete();

tram.Delete();

Text\_File<StopsInfo> GrId("Data\\GrId.txt");

Text\_File<StopsInfo> UndId("Data\\UndId.txt");

Text\_File<Underground\_Stops> UndStops("Data\\UndStops.txt");

Text\_File<Ground\_Stops> GrStops("Data\\GrStops.txt");

Text\_File<Ground\_Trans> Buses("Data\\Buses.txt");

Text\_File<Ground\_Trans> Trolleybuses("Data\\Trolleybuses.txt");

Text\_File<Ground\_Trans> Trams("Data\\Trams.txt");

Text\_File<Underground\_Trans> Trains("Data\\Trains.txt");

GrId.Open\_in();

if (!GrId.IsEmpty())

{

GrId.Remote();

for (int i = 0;!GrId.End(); i++)

{

StopsInfo temp;

GrId.Read(temp);

gid.push\_back(temp);

}

}

UndId.Open\_in();

if (!UndId.IsEmpty())

{

UndId.Remote();

for (int i = 0;!UndId.End(); i++)

{

StopsInfo temp;

UndId.Read(temp);

uid.push\_back(temp);

}

}

UndStops.Open\_in();

if (!UndStops.IsEmpty())

{

UndStops.Remote();

for (int i = 0;!UndStops.End(); i++)

{

int l;

Underground\_Stops temp;

UndStops.Read(temp);

for (l = 0;temp.getid() != uid[l].id;l++);

temp.setname(uid[l].name);

us.push(temp);

}

}

GrStops.Open\_in();

if (!GrStops.IsEmpty())

{

GrStops.Remote();

for (int i = 0;!GrStops.End(); i++)

{

int l;

Ground\_Stops temp;

GrStops.Read(temp);

for (l = 0;temp.getid() != gid[l].id;l++);

temp.setname(gid[l].name);

gs.push(temp);

}

}

Buses.Open\_in();

if (!Buses.IsEmpty())

{

Buses.Remote();

for (int i = 0;!Buses.End();i++)

{

int size = 0;

vector<int> st;

Ground\_Trans temp;

Buses.Read(temp);

size = (int)temp.getst().size();

st = temp.getst();

bus.push(temp);

for (int l = 0;l < size;l++)

{

int k;

for (k = 0;st[l] != gs[k].getid();k++);

bus[bus.getsize() - 1].pushstops(&gs[k]);

gs[k].pushtrans(1, &bus[bus.getsize() - 1]);

}

}

}

Trolleybuses.Open\_in();

if (!Trolleybuses.IsEmpty())

{

Trolleybuses.Remote();

for (int i = 0;!Trolleybuses.End();i++)

{

int size = 0;

vector<int> st;

Ground\_Trans temp;

Trolleybuses.Read(temp);

size = (int)temp.getst().size();

st = temp.getst();

trolleybus.push(temp);

for (int l = 0;l < size;l++)

{

int k;

for (k = 0;st[l] != gs[k].getid();k++);

trolleybus[trolleybus.getsize() - 1].pushstops(&gs[k]);

gs[k].pushtrans(2, &trolleybus[bus.getsize() - 1]);

}

}

}

Trams.Open\_in();

if (!Trams.IsEmpty())

{

Trams.Remote();

for (int i = 0;!Trams.End();i++)

{

int size = 0;

vector<int> st;

Ground\_Trans temp;

Trams.Read(temp);

size = (int)temp.getst().size();

st = temp.getst();

tram.push(temp);

for (int l = 0;l < size;l++)

{

int k;

for (k = 0;st[l] != gs[k].getid();k++);

tram[tram.getsize() - 1].pushstops(&gs[k]);

gs[k].pushtrans(3, &tram[bus.getsize() - 1]);

}

}

}

Trains.Open\_in();

if (!Trains.IsEmpty())

{

Trains.Remote();

for (int i = 0;!Trains.End(); i++)

{

int k;

int size = 0;

vector<Underground\_Stops\*> stations;

vector<int> st;

Underground\_Trans temp;

Trains.Read(temp);

size = (int)temp.getst().size();

st = temp.getst();

for (int l = 0;l < size;l++)

{

for (k = 0;st[l] != us[k].getid();k++);

stations.push\_back(&us[k]);

us[k].setline(temp.getnumber());

}

temp.setstops(stations);

train.push(temp);

}

}

}

void Interface::save()

{

Text\_File<StopsInfo> GrId("Data\\GrId.txt");

GrId.Open\_out();

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

GrId.Write(gid[i]);

Text\_File<StopsInfo> UndId("Data\\UndId.txt");

UndId.Open\_out();

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

UndId.Write(uid[i]);

Text\_File<Underground\_Stops> UndStops("Data\\UndStops.txt");

UndStops.Open\_out();

for (int i = 0;i < us.getsize();i++)

UndStops.Write(us[i]);

Text\_File<Ground\_Stops> GrStops("Data\\GrStops.txt");

GrStops.Open\_out();

for (int i = 0;i < gs.getsize();i++)

GrStops.Write(gs[i]);

Text\_File<Ground\_Trans> Buses("Data\\Buses.txt");

Buses.Open\_out();

for (int i = 0;i < bus.getsize();i++)

Buses.Write(bus[i]);

Text\_File<Ground\_Trans> Trolleybuses("Data\\Trolleybuses.txt");

Trolleybuses.Open\_out();

for (int i = 0;i < trolleybus.getsize();i++)

Trolleybuses.Write(trolleybus[i]);

Text\_File<Ground\_Trans> Trams("Data\\Trams.txt");

Trams.Open\_out();

for (int i = 0;i < tram.getsize();i++)

Trams.Write(tram[i]);

Text\_File<Underground\_Trans> Trains("Data\\Trains.txt");

Trains.Open\_out();

for (int i = 0;i < train.getsize();i++)

Trains.Write(train[i]);

}

void Interface::Choose()

{

bool flag = 0, key = 0;

int vflag = 1;

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "==================== MENU ======================" << endl;

cout << "||=======(Press Esc to end the program)=======||" << endl;

if (vflag == 1)

cout << "||---------[1. Show all the stations]---------||" << endl;

else

cout << "||--------- 1. Show all the stations ---------||" << endl;

if (vflag == 2)

cout << "||----------[2. Show all transport]-----------||" << endl;

else

cout << "||---------- 2. Show all transport -----------||" << endl;

if (vflag == 3)

cout << "||-----------[3. Do some changes]-------------||" << endl;

else

cout << "||----------- 3. Do some changes -------------||" << endl;

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::UP:

if (vflag == 1)

vflag = 3;

else vflag--;

key++;

break;

case Keys::DOWN:

if (vflag == 3)

vflag = 1;

else vflag++;

key++;

break;

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

switch (vflag)

{

case 1:

Stops();

flag = 0;

break;

case 2:

cout << vflag;

Trans();

break;

case 3:

cout << vflag;

Change();

break;

}

} while (vflag);

}

void Interface::Change()

{

bool flag = 0, key = 0;

int vflag = 1;

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "================== CHANGES =====================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

if (vflag == 1)

cout << "||------------[1. Add an element]-------------||" << endl;

else

cout << "||------------ 1. Add an element -------------||" << endl;

if (vflag == 2)

cout << "||-------[2. Change some information]---------||" << endl;

else

cout << "||------- 2. Change some information ---------||" << endl;

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::UP:

if (vflag == 1)

vflag = 2;

else vflag--;

key++;

break;

case Keys::DOWN:

if (vflag == 2)

vflag = 1;

else vflag++;

key++;

break;

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

switch (vflag)

{

case 1:

Add();

break;

case 2:

ChangeSmth();

break;

}

if (!vflag)

{

flag = 0;

do

{

cout << "Are you sure to save the changes?" << endl << endl;

cout << "0)Turn back." << endl;

cout << "1)Yes." << endl;

cout << "2)No." << endl;

switch (\_getch())

{

case '1':

save();

flag++;

break;

case '2':

open();

flag++;

break;

case '0':

vflag = 1;

flag++;

break;

}

} while (!flag);

}

} while (vflag);

}

void Interface::Add()

{

bool flag = 0, key = 0;

int vflag = 1;

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "================== CHANGES =====================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

if (vflag == 1)

cout << "||---------[1. Add a ground station]----------||" << endl;

else

cout << "||--------- 1. Add a ground station ----------||" << endl;

if (vflag == 2)

cout << "||------[2. Add an underground station]-------||" << endl;

else

cout << "||------ 2. Add an underground station -------||" << endl;

if (vflag == 3)

cout << "||----------[3. Add a ground route]-----------||" << endl;

else

cout << "||---------- 3. Add a ground route -----------||" << endl;

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::UP:

if (vflag == 1)

vflag = 3;

else vflag--;

key++;

break;

case Keys::DOWN:

if (vflag == 3)

vflag = 1;

else vflag++;

key++;

break;

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

switch (vflag)

{

case 1:

{

int numb = gs.getsize() + 1;

Ground\_Stops temp;

temp.setid(numb);

gs.push(temp);

StopsInfo info;

info.id = temp.getid();

gid.push\_back(info);

InName(gid, gs[gs.getsize() - 1]);

if (gs.getsize() > 1)

gs[gs.getsize() - 1].setnstops(gs);

break;

}

case 2:

{

Underground\_Stops temp;

int i, f = 0;

vector<Nearest\_St> stations;

Nearest\_St st;

cout << "Enter the number of the branch: ";

int numb = InInt(1, train.getsize() + 1);

temp.setline(numb);

i = us.getsize() + 1;

temp.setid(i);

us.push(temp);

StopsInfo info;

info.id = temp.getid();

uid.push\_back(info);

InName(uid, us[us.getsize() - 1]);

for (i = 0;i < us.getsize(); i++)

if (us[i].getline() == temp.getline())

f++;

if (f > 1)

us[us.getsize() - 1].setnstops(us);

if (numb > train.getsize())

{

Underground\_Trans ttemp;

ttemp.setnumber(numb);

cin >> ttemp;

ttemp.pushst(temp.getid());

ttemp.pushstops(&us[us.getsize() - 1]);

train.push(ttemp);

}

else

{

train[temp.getline() - 1].pushst(temp.getid());

train[temp.getline() - 1].pushstops(&us[us.getsize() - 1]);

}

break;

}

case 3:

{

Ground\_Trans temp;

int numb;

do

{

flag = 0;

cout << "Choose the type of the transport:" << endl;

cout << "1.Bus\n2.Trolleybus\n3.Tram\n";

switch (\_getch())

{

case '1':

cin >> temp;

for (numb = 0;numb < bus.getsize();numb++)

if (bus[numb].getnumber() != numb + 1)

break;

numb++;

temp.setnumber(numb);

bus.push(temp);

bus.sort();

bus[bus.getsize()].setstops(gs, 1);

flag++;

break;

case '2':

cin >> temp;

for (numb = 0;numb < trolleybus.getsize();numb++)

if (trolleybus[numb].getnumber() != numb + 1)

break;

numb++;

temp.setnumber(numb);

trolleybus.push(temp);

trolleybus.sort();

trolleybus[trolleybus.getsize()].setstops(gs, 2);

flag++;

break;

case '3':

cin >> temp;

for (numb = 0;numb < tram.getsize();numb++)

if (tram[numb].getnumber() != numb + 1)

break;

numb++;

temp.setnumber(numb);

tram.push(temp);

tram.sort();

tram[tram.getsize()].setstops(gs, 2);

flag++;

break;

}

} while (!flag);

}

}

} while (vflag);

}

void Interface::ChangeSmth()

{

bool flag = 0, key = 0;

int vflag = 1;

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "================== CHANGES =====================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

if (vflag == 1)

cout << "||---[1. Change information about stations]---||" << endl;

else

cout << "||--- 1. Change information about stations ---||" << endl;

if (vflag == 2)

cout << "||--[2. Change information about transport]---||" << endl;

else

cout << "||-- 2. Change information about transport ---||" << endl;

if (vflag == 3)

cout << "||--------------[3. Delete all]---------------||" << endl;

else

cout << "||-------------- 3. Delete all ---------------||" << endl;

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::UP:

if (vflag == 1)

vflag = 3;

else vflag--;

key++;

break;

case Keys::DOWN:

if (vflag == 3)

vflag = 1;

else vflag++;

key++;

break;

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

switch (vflag)

{

case 1:

ChangeStops();

break;

case 2:

ChangeTrans();

break;

case 3:

gid.clear();

uid.clear();

us.Delete();

gs.Delete();

train.Delete();

bus.Delete();

trolleybus.Delete();

tram.Delete();

cout << "Deleted!" << endl;

system("pause");

break;

}

} while (vflag);

}

void Interface::ChangeStops()

{

bool flag = 0, key = 0;

int vflag = 1;

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "================== CHANGES =====================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

if (vflag == 1)

cout << "||--[1. Change the name of ground station]----||" << endl;

else

cout << "||-- 1. Change the name of ground station ----||" << endl;

if (vflag == 2)

cout << "||[2. Change the name of underground station]-||" << endl;

else

cout << "|| 2. Change the name of underground station -||" << endl;

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::UP:

if (vflag == 1)

vflag = 2;

else vflag--;

key++;

break;

case Keys::DOWN:

if (vflag == 2)

vflag = 1;

else vflag++;

key++;

break;

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

switch (vflag)

{

case 1:

{

int f = 0, numb, num = 0;

string name;

numb = SearchStop(gs);

if (numb != -1)

{

cout << "Enter the new name: \n";

InName(gid, gs[numb]);

}

break;

}

case 2:

{

int f = 0, numb, num = 0;

string name;

numb = SearchStop(us);

if (numb != -1)

{

cout << "Enter the new name: \n";

InName(uid, us[numb]);

}

break;

}

}

} while (vflag);

}

void Interface::ChangeTrans()

{

bool flag = 0, key = 0;

int vflag = 1;

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "================== CHANGES =====================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

if (vflag == 1)

cout << "||----------[1. Change ground route]----------||" << endl;

else

cout << "||---------- 1. Change ground route ----------||" << endl;

if (vflag == 2)

cout << "||--------[2. Change ground timetable]--------||" << endl;

else

cout << "||-------- 2. Change ground timetable --------||" << endl;

if (vflag == 3)

cout << "||--------[3. Delete the ground route]--------||" << endl;

else

cout << "||-------- 3. Delete the ground route --------||" << endl;

if (vflag == 4)

cout << "||------[4. Change underground interval]------||" << endl;

else

cout << "||------ 4. Change underground interval ------||" << endl;

if (vflag == 5)

cout << "||-----[5. Delete the underground route]------||" << endl;

else

cout << "||----- 5. Delete the underground route ------||" << endl;

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::UP:

if (vflag == 1)

vflag = 5;

else vflag--;

key++;

break;

case Keys::DOWN:

if (vflag == 5)

vflag = 1;

else vflag++;

key++;

break;

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

switch (vflag)

{

case 1:

{

int f = 0, numb;

do {

cout << "1.Bus" << endl;

cout << "2.Trolleybus" << endl;

cout << "3.Tram" << endl;

switch (\_getch())

{

case '1':

if (numb = SearchGTrans(bus) > -1)

bus[numb].setstops(gs, 1);

f++;

break;

case '2':

if (numb = SearchGTrans(trolleybus) > -1)

trolleybus[numb].setstops(gs, 2);

f++;

break;

case '3':

if (numb = SearchGTrans(tram) > -1)

tram[numb].setstops(gs, 3);

f++;

break;

}

} while (!f);

break;

}

case 2:

{

int f = 0, numb;

do {

cout << "1.Bus" << endl;

cout << "2.Trolleybus" << endl;

cout << "3.Tram" << endl;

switch (\_getch())

{

case '1':

if (numb = SearchGTrans(bus) > -1)

cin >> bus[numb];

f++;

break;

case '2':

if (numb = SearchGTrans(trolleybus) > -1)

cin >> trolleybus[numb];

f++;

break;

case '3':

if (numb = SearchGTrans(tram) > -1)

cin >> tram[numb];

f++;

break;

}

} while (!f);

break;

}

case 3:

{

int f = 0, numb;

do {

cout << "1.Bus" << endl;

cout << "2.Trolleybus" << endl;

cout << "3.Tram" << endl;

switch (\_getch())

{

case '1':

if (numb = SearchGTrans(bus) > -1)

bus[numb].delstops(1);

bus.DeleteNode(numb);

f++;

break;

case '2':

if (numb = SearchGTrans(trolleybus) > -1)

trolleybus[numb].delstops(2);

trolleybus.DeleteNode(numb);

f++;

break;

case '3':

if (numb = SearchGTrans(tram) > -1)

tram[numb].delstops(3);

tram.DeleteNode(numb);

f++;

break;

}

} while (!f);

break;

}

case 4:

{

int numb;

if (!train.getsize())

{

cout << "There are no trains!" << endl;

system("pause");

break;

}

cout << "Enter the number of the route: ";

numb = InInt(1, train.getsize());

cin >> train[numb - 1];

break;

}

case 5:

{

int numb, i;

if (!train.getsize())

{

cout << "There are no trains!" << endl;

system("pause");

break;

}

cout << "Enter the number of the line: ";

numb = InInt(1, train.getsize());

for (i = 0;i < us.getsize();i++)

if (us[i].getline() == numb)

{

us.DeleteNode(i);

i--;

}

train.DeleteNode(numb);

while (numb <= train.getsize())

{

train[numb].setnumber(numb);

for (i = 0;i < us.getsize();i++)

if (us[i].getid() == numb + 1)

us[i].setline(numb);

numb++;

}

break;

}

}

if (vflag)

{

cout << "Success!\n";

system("pause");

}

} while (vflag);

}

void Interface::Stops()

{

bool flag = 0, key = 0;

int vflag = 1;

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "==================== STOPS =====================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

if (vflag == 1)

cout << "||-------------[1. Ground stops]--------------||" << endl;

else

cout << "||------------- 1. Ground stops --------------||" << endl;

if (vflag == 2)

cout << "||-----------[2. Underground stops]-----------||" << endl;

else

cout << "||----------- 2. Underground stops -----------||" << endl;

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::UP:

if (vflag == 1)

vflag = 2;

else vflag--;

key++;

break;

case Keys::DOWN:

if (vflag == 2)

vflag = 1;

else vflag++;

key++;

break;

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

switch (vflag)

{

case 1:

GrStops();

flag = 0;

break;

case 2:

cout << vflag;

UndStops();

break;

}

} while (vflag);

}

void Interface::GrStops()

{

bool flag = 0, key = 0;

int vflag = 1;

if (gs.getsize() != 0)

gs.sort();

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "================= GROUND STOPS =================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

cout << "||('Enter' - to choose the number of the stop)||" << endl;

for (int i = 0; i < gs.getsize(); i++)

cout << "||" << setw(3) << i + 1 << ". " << gs[i].getname() << setw(41 - gs[i].getname().size()) << "||" << endl;

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

if (vflag)

{

if (gs.getsize() != 0)

{

cout << "Enter the number of the station: ";

int numb = InInt(1, gs.getsize());

gs[numb - 1].info();

}

else

{

cout << "There are no stops!" << endl;

system("pause");

}

}

} while (vflag);

}

void Interface::UndStops()

{

bool flag = 0, key = 0;

int vflag = 1;

if (us.getsize() != 0)

us.sort();

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "============== UNDERGROUND STOPS ===============" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

cout << "||('Enter' - to choose the number of the stop)||" << endl;

for (int i = 0; i < us.getsize(); i++)

cout << "||" << setw(3) << i + 1 << ". " << us[i].getname() << setw(41 - us[i].getname().size()) << "||" << endl;

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

if (vflag)

{

if (us.getsize() != 0)

{

cout << "Enter the number of the station: ";

int numb = InInt(1, us.getsize());

numb--;

Time t = train[us[numb].getline()].getinterval();

Time wt = train[us[numb].getline()].getwinterval();

cout << "=========== STATION " << setw(15) << us[numb].getname() << " ============" << endl;

cout << "||========(Press any key to turn back)========||" << endl;

cout << "|| Line: " << setw(2) << us[numb].getline() << setw(37) << "||" << endl;

cout << "|| Interval on weekdays: " << t << setw(19) << "||" << endl;

cout << "|| Interval on weekends: " << wt << setw(19) << "||" << endl;

cout << "||============================================||" << endl;

}

else cout << "There are no metro stations!" << endl;

system("pause");

}

} while (vflag);

}

void Interface::Trans()

{

bool flag = 0, key = 0;

int vflag = 1;

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "================== TRANSPORT ===================" << endl;

cout << "||=======(Press Esc to end the program)=======||" << endl;

if (vflag == 1)

cout << "||-----------[1. Ground transport]------------||" << endl;

else

cout << "||----------- 1. Ground transport ------------||" << endl;

if (vflag == 2)

cout << "||---------[2. Underground transport]---------||" << endl;

else

cout << "||--------- 2. Underground transport ---------||" << endl;

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::UP:

if (vflag == 1)

vflag = 2;

else vflag--;

key++;

break;

case Keys::DOWN:

if (vflag == 2)

vflag = 1;

else vflag++;

key++;

break;

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

switch (vflag)

{

case 1:

GrTrans();

flag = 0;

break;

case 2:

cout << vflag;

UndTrans();

break;

}

} while (vflag);

}

void Interface::GrTrans()

{

bool flag = 0, key = 0;

int vflag = 1;

int hflag = 1;

if (bus.getsize() != 0)

bus.sort();

if (trolleybus.getsize() != 0)

trolleybus.sort();

if (tram.getsize() != 0)

tram.sort();

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "============== GROUND TRANSPORT ================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

cout << "||('Enter'- to choose the number of the route)||" << endl;

if (hflag == 1)

{

cout << "||---[1.Buses]-- 2.Trolleybuses -- 3.Trams ---||" << endl;

bus.PrintRing();

}

else if (hflag == 2)

{

cout << "||--- 1.Buses --[2.Trolleybuses]-- 3.Trams ---||" << endl;

trolleybus.PrintRing();

}

else if (hflag == 3)

{

cout << "||--- 1.Buses -- 2.Trolleybuses --[3.Trams]---||" << endl;

tram.PrintRing();

}

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::LEFT:

if (hflag == 1)

hflag = 3;

else hflag--;

key++;

break;

case Keys::RIGHT:

if (hflag == 3)

hflag = 1;

else hflag++;

key++;

break;

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

if (vflag)

{

int numb;

switch (hflag)

{

case 1:

numb = SearchGTrans(bus);

if (numb != -1)

bus[numb].info();

break;

case 2:

numb = SearchGTrans(trolleybus);

if (numb != -1)

trolleybus[numb].info();

break;

case 3:

numb = SearchGTrans(tram);

if (numb != -1)

tram[numb].info();

break;

}

}

} while (vflag);

}

void Interface::UndTrans()

{

bool flag = 0, key = 0;

int vflag = 1;

if (train.getsize() != 0)

train.sort();

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "==================== METRO =====================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

cout << "||('Enter'- to choose the number of the route)||" << endl;

train.PrintRing();

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

if (vflag)

{

if (train.getsize() != 0)

{

cout << "Enter the number of the line: ";

int numb = InInt(1, train.getsize());

train[numb - 1].info();

}

else cout << "There are no objects with the type 'train'!" << endl;

system("pause");

}

} while (vflag);

}

**Модуль Time**

“Time.h”

#pragma once

#include <iostream>

#include <iomanip>

#include <string>

#include <ctime>

#include <fstream>

using namespace std;

class Time

{

private:

int h, m;

public:

Time()

{

h = 0;

m = 0;

}

Time(double time)

{

h = (int)time;

m = (int)((time - h) \* 100);

}

Time(const Time& time)

{

h = time.h;

m = time.m;

}

~Time() {}

Time& now();

int geth();

int getm();

void seth(int newh);

void setm(int newm);

Time operator-(Time& sub);

Time operator+(Time& add);

Time& operator-=(Time& sub);

Time& operator+=(Time& add);

Time& operator=(double time);

Time& operator=(const Time& time);

bool operator>(Time time);

bool operator<(Time time);

bool operator>=(Time time);

bool operator<=(Time time);

bool operator==(Time time);

bool operator!=(Time time);

friend istream& operator>>(istream& stream, Time& T);

friend ostream& operator<<(ostream& stream, Time& T);

friend ifstream& operator>>(ifstream& stream, Time& T);

friend ofstream& operator<<(ofstream& stream, Time& T);

};

“Time.cpp”

#include "Time.h"

Time& Time::now()

{

struct tm ltm;

time\_t now = time(0);

localtime\_s(&ltm, &now);

h = ltm.tm\_hour;

m = ltm.tm\_min;

return \*this;

}

int Time::geth()

{

return h;

}

int Time::getm()

{

return m;

}

void Time::seth(int newh)

{

h = newh;

}

void Time::setm(int newm)

{

m = newm;

}

Time Time::operator-(Time& sub)

{

Time temp;

temp.m -= sub.m;

if (temp.m < 0)

{

temp.h--;

temp.m += 60;

}

temp.h -= sub.h;

if (temp.h < 0)

{

temp.h += 24;

}

return temp;

}

Time Time::operator+(Time& add)

{

Time temp = \*this;

temp.h += add.h;

temp.m += add.m;

if (temp.m >= 60)

{

temp.h++;

temp.m -= 60;

}

if (temp.h >= 24)

{

temp.h -= 24;

}

return temp;

}

Time& Time::operator-=(Time& sub)

{

m -= sub.m;

if (m < 0)

{

h--;

m += 60;

}

h -= sub.h;

if (h < 0)

{

h += 24;

}

return \*this;

}

Time& Time::operator+=(Time& add)

{

h += add.h;

m += add.m;

if (m >= 60)

{

h++;

m -= 60;

}

if (h >= 24)

{

h -= 24;

}

return \*this;

}

Time& Time::operator=(double time)

{

time += 0.001;

h = (int)time;

m = (int)((time - h) \* 100);

return \*this;

}

Time& Time::operator=(const Time& time)

{

h = time.h;

m = time.m;

return \*this;

}

bool Time::operator>(Time time)

{

bool flag = 0;

if (h > time.h && ((h > 5 && time.h > 5) || (h <= 5 && time.h <= 6)))

flag = 1;

else if (h < 5 && time.h > 5)

flag = 1;

else if (h == time.h && m > time.m)

flag = 1;

return flag;

}

bool Time::operator<(Time time)

{

bool flag = 0;

if (h < time.h && ((h > 5 && time.h > 5) || (h <= 5 && time.h <= 5)))

flag = 1;

else if (h > 5 && time.h < 5)

flag = 1;

else if (h == time.h && m < time.m)

flag = 1;

return flag;

}

bool Time::operator>=(Time time)

{

bool flag = 0;

if (h == time.h && m >= time.m)

flag = 1;

else if (h > time.h && ((h > 5 && time.h > 5) || (h <= 5 && time.h <= 5)))

flag = 1;

else if (h < 5 && time.h > 5)

flag = 1;

return flag;

}

bool Time::operator<=(Time time)

{

bool flag = 0;

if (h == time.h && m <= time.m)

flag = 1;

else if (h < time.h && ((h > 5 && time.h > 5) || (h <= 5 && time.h <= 5)))

flag = 1;

else if (h > 5 && time.h < 5)

flag = 1;

return flag;

}

bool Time::operator==(Time time)

{

bool flag = 0;

if (h == time.h && m == time.m)

flag = 1;

return flag;

}

bool Time::operator!=(Time time)

{

bool flag = 0;

if (h != time.h || m != time.m)

flag = 1;

return flag;

}

istream& operator>>(istream& stream, Time& T)

{

double time;

stream >> time;

T.h = (int)time;

T.m = (int)((time - T.h) \* 100);

return stream;

}

ostream& operator<<(ostream& stream, Time& T)

{

stream << T.h << ':';

if (T.m < 10)

stream << 0;

stream << T.m;

return stream;

}

ifstream& operator>>(ifstream& stream, Time& T)

{

double time;

stream >> time;

T.h = (int)time;

T.m = (int)((time - T.h) \* 100);

return stream;

}

ofstream& operator<<(ofstream& stream, Time& T)

{

stream << setw(4) << T.h << '.';

if (T.m < 10)

stream << 0;

stream << T.m;

return stream;

}

**Модуль Exceptions**

“Exceptions.h”

#include <iostream>

#include <conio.h>

#include <string>

#pragma once

using namespace std;

enum errors

{

UncorrectType = 1,

NegativeNum,

TooLittleNum,

TooBigNum,

NotEngLetters,

TooShortStr,

TooLongStr,

IsNotWord,

NotBigLetter,

UncorrectHour,

UncorrectMin,

TooBigInterval,

IsNotExist,

AllreadyIncl

};

class Exception

{

protected:

string message;

public:

Exception(string msg) : message(msg) { }

void what()

{

cout << "\nERROR! " << message << endl;

}

~Exception() {}

};

**Модуль Input\_Except**

“Input\_Except.h”

#pragma once

#include "Exceptions.h"

#include "Structs.h"

#include "Time.h"

class Input\_Except : public Exception

{

protected:

int code;

public:

Input\_Except(string msg, int type) : Exception(msg)

{

code = 1000 + type;

}

void what()

{

cout << "\nERROR " << code << "! " << message << endl;

}

~Input\_Except() {}

};

static int InInt(int min, int max)

{

int par;

bool flag = false;

do

{

rewind(stdin);

cin.clear();

try

{

if (!(cin >> par) || cin.get() != '\n') throw Input\_Except("Uncorrect type of input!!!", UncorrectType);

if (par < min) throw Input\_Except("Too litle number!!!", TooLittleNum);

if (par > max) throw Input\_Except("Too big number!!!", TooBigNum);

flag = true;

}

catch (Input\_Except& ex)

{

ex.what();

cout << "It must be from " << min << " to " << max << endl;

cout << "\nPlease try again: ";

}

catch (...)

{

cout << "\nUnknown ERROR!\nPlease try again: ";

}

} while (!flag);

return par;

}

static double InDouble(double min, double max)

{

double par;

bool flag = false;

do

{

rewind(stdin);

cin.clear();

try

{

if (!(cin >> par) || cin.get() != '\n') throw Input\_Except("Uncorrect type of input!!!", UncorrectType);

if (par < min) throw Input\_Except("Too litle number!!!", TooLittleNum);

if (par > max) throw Input\_Except("Too big number!!!", TooBigNum);

flag = true;

}

catch (Input\_Except& ex)

{

ex.what();

cout << "It must be from " << min << " to " << max << endl;

cout << "\nPlease try again: ";

}

catch (...)

{

cout << "\nUnknown ERROR!\nPlease try again: ";

}

} while (!flag);

return par;

}

static string InString()

{

string str;

int j;

bool flag = false;

do

{

rewind(stdin);

cin.clear();

j = 0;

try

{

cin >> str;

if (str.size() < 3) throw Input\_Except("Too short string!!!\nThe length of the string should be from 2 to 10 letters.", TooShortStr);

if (str.size() > 15) throw Input\_Except("Too long string!!!\nThe length of the string should be from 2 to 10 letters.", TooLongStr);

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

if ((str[i] < 'A' || str[i] > 'z') && (str[i] < '0' || str[i] > '9')) throw Input\_Except("Uncorrect input!!!\nThe name should contain only capital or little letters of English alphabet or digits.", NotEngLetters);

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

if (str[i] >= '0' && str[i] <= '9')

j++;

if (j == str.size()) throw Input\_Except("Uncorrect input!!!\nThe name shouldn't contain only digits.", IsNotWord);

flag = true;

}

catch (Input\_Except& ex)

{

ex.what();

cout << "\nPlease try again: ";

}

catch (...)

{

cout << "\nUnknown ERROR!\nPlease try again: ";

}

} while (!flag);

return str;

}

template <class T>

static void InName(vector<StopsInfo>& id, T& st)

{

string name;

int j, num, flag = false;

for (j = 0; j < id.size();j++)

if (id[j].id == st.getid())

break;

num = j;

cout << "Enter the name of the station: \n";

do

{

rewind(stdin);

cin.clear();

try

{

name = InString();

for (j = 0; j < id.size();j++)

if (id[j].name == name)

if (id[j].id != st.getid())

throw Input\_Except("This name is allready used!", AllreadyIncl);

id[num].name = name;

st.setname(name);

flag = true;

}

catch (Input\_Except& ex)

{

ex.what();

cout << "\nPlease try again: ";

}

catch (...)

{

cout << "\nUnknown ERROR!\nPlease try again: ";

}

} while (!flag);

}

static Time InTime(double max)

{

float t;

Time time;

bool flag = false;

do

{

rewind(stdin);

cin.clear();

try

{

if (!(cin >> t) || cin.get() != '\n') throw Input\_Except("Uncorrect type of input!!!", UncorrectType);

if (t > max) throw Input\_Except("To bid time interval!!!", TooBigInterval);

if ((int)t > 23 || (int)t < 0) throw Input\_Except("Incorrect format for entering hours!!!", UncorrectHour);

if ((int)((t -(int)t)\*100) > 59 || (int)((t - (int)t) \* 100) < 0) throw Input\_Except("Incorrect format for entering minutes!!!", UncorrectMin);

time = t;

flag = true;

}

catch (Input\_Except& ex)

{

ex.what();

cout << "Hours must be from 0 to 23, minuters must be from 0 to 59. Time interval must be less than " << max << '.' << endl;

cout << "\nPlease try again: ";

}

catch (...)

{

cout << "\nUnknown ERROR!\nPlease try again: ";

}

} while (!flag);

return time;

}

**Модуль File\_Except**

“File\_Except.h”

#pragma once

#include <fstream>

#include "Exceptions.h"

class File\_Except : public Exception

{

protected:

int code;

public:

File\_Except(string msg, int type) : Exception(msg)

{

code = 2000 + type;

}

void what()

{

cout << "\nERROR " << code << "! " << message << endl;

}

~File\_Except() {}

};

static ifstream InFile(string adress)

{

ifstream file;

file.open(adress, ios::in);

try

{

if (!file.is\_open()) throw File\_Except("Cannot open the file!", UncorrectType);

}

catch (File\_Except& ex)

{

ex.what();

exit(-1);

}

return file;

}

static ofstream OutFile(string adress)

{

ofstream file;

file.open(adress, ios::out);

try

{

if (!file.is\_open()) throw File\_Except("Cannot open the file!", UncorrectType);

}

catch (File\_Except& ex)

{

ex.what();

exit(-1);

}

return file;

}

**Модуль Search\_Except**

“Search\_Except.h”

#pragma once

#include "Exceptions.h"

#include "Ground\_Trans.h"

#include "Template.h"

class Search\_Except : public Exception

{

protected:

int code;

public:

Search\_Except(string msg, int type) : Exception(msg)

{

code = 3000 + type;

}

void what()

{

cout << "\nERROR " << code << "! " << message << endl;

}

~Search\_Except() {}

};

static int SearchGTrans(Ring<Ground\_Trans>& trans)

{

int numb, i, f = 0;

bool flag = false;

do

{

rewind(stdin);

cin.clear();

try

{

if (!trans.getsize())

{

cout << "There are no route with this type of transport!" << endl;

system("pause");

return - 1;

}

cout << "Enter the number of the route: ";

numb = InInt(1, trans[trans.getsize() - 1].getnumber());

for (i = 0; i < trans.getsize();i++)

if (trans[i].getnumber() == numb)

{

f = 1;

break;

}

if (f != 1) throw Input\_Except("There is no route with this number!!!", IsNotExist);

flag = true;

}

catch (Input\_Except& ex)

{

ex.what();

cout << "\nPlease try again: ";

}

catch (...)

{

cout << "\nUnknown ERROR!\nPlease try again: ";

}

} while (!flag);

return i;

}

template <class T>

static int SearchStop(T& st)

{

int i, f = 0;

bool flag = false;

string name;

do

{

rewind(stdin);

cin.clear();

try

{

if (!st.getsize())

{

cout << "There are no stops!" << endl;

system("pause");

return -1;

}

cout << "Enter the name of the station: \n";

name = InString();

for (i = 0; i < st.getsize();i++)

if (st[i].getname() == name)

{

f = 1;

break;

}

if (f != 1) throw Input\_Except("There is no route with this number!!!", IsNotExist);

flag = true;

}

catch (Input\_Except& ex)

{

ex.what();

cout << "\nPlease try again: ";

}

catch (...)

{

cout << "\nUnknown ERROR!\nPlease try again: ";

}

} while (!flag);

return i;

}

**Модуль Header**

“Header.h”

#pragma once

#include "Underground\_Trans.h"

#include "Underground\_Stops.h"

#include "Ground\_Trans.h"

#include "Ground\_Stops.h"

#include "Search\_Except.h"

#include "Input\_Except.h"

#include "File\_Except.h"

#include "Template.h"

#include "Text\_File.h"

#include "Time.h"

#include "Keys.h"

**Модуль Structs**

“Structs.h”

#pragma once

#include <conio.h>

#include <iostream>

#include <iomanip>

#include <string>

#include <ctime>

#include <fstream>

#include "Time.h"

#include "vector"

#include "Keys.h"

struct Nearest\_St

{

int id = 0;

Time time;

friend istream& operator>>(istream& stream, Nearest\_St& S)

{

cout << "ID: ";

stream >> S.id;

cout << "Time to get to: ";

stream >> S.time;

return stream;

}

friend ostream& operator<<(ostream& stream, Nearest\_St& S)

{

stream << setw(5) << S.id << S.time;

return stream;

}

friend ifstream& operator>>(ifstream& stream, Nearest\_St& S)

{

stream >> S.id >> S.time;

return stream;

}

friend ofstream& operator<<(ofstream& stream, Nearest\_St& S)

{

stream << setw(5) << S.id;

stream << S.time;

return stream;

}

};

struct StopsInfo

{

int id = 0;

string name = "\0";

friend istream& operator>>(istream& stream, StopsInfo& Info)

{

stream >> Info.id >> Info.name;

return stream;

}

friend ostream& operator<<(ostream& stream, StopsInfo& Info)

{

stream << setw(5) << Info.id << " " << Info.name;

stream << endl;

return stream;

}

friend ifstream& operator>>(ifstream& stream, StopsInfo& Info)

{

stream >> Info.id;

stream >> Info.name;

return stream;

}

friend ofstream& operator<<(ofstream& stream, StopsInfo& Info)

{

stream << setw(5) << Info.id;

stream << setw(15) << Info.name;

return stream;

}

};

**Модуль Keys**

“Keys.h”

#pragma once

enum Keys

{

LEFT = 75,

RIGHT = 77,

UP = 72,

DOWN = 80,

ESC = 27,

SPACE = 32,

ENTER = 13,

};

**Модуль Template**

“Template.h”

#pragma once

#include <iostream>

#include <iomanip>

#include <string>

#include "Iterator.h"

using namespace std;

template <typename T>

class Ring

{

private:

Node<T>\* head;

Node<T>\* current;

int size;

public:

Ring()

{

head = NULL;

current = NULL;

size = 0;

}

Ring(Ring& r)

{

head = r.head;

current = r.current;

size = r.size;

}

Ring(T x)

{

head = NULL;

current = NULL;

size = 0;

push(x);

}

Ring(const Ring<T>& C)

{

this->head = NULL;

this->current = NULL;

this->size = 0;

Node<T>\* p = C.current;

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

{

push(p->obj);

p = p->next;

}

}

~Ring()

{

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

}

int IsNotEmpty()

{

if (current == NULL) return 0;

else return 1;

}

int getsize()

{

return size;

}

void PrintRing()

{

if (IsNotEmpty())

{

Iterator<T> temp = head;

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

{

cout << \*temp;

temp++;

}

}

else cout << "|| List is empty! ||\n";

}

void push(T data)

{

Node<T>\* inserted;

inserted = new Node<T>;

inserted->obj = data;

if (!IsNotEmpty())

{

head = inserted;

current = inserted;

current->next = inserted;

current->prev = inserted;

}

else

{

inserted->next = current->next;

inserted->next->prev = inserted;

current->next = inserted;

inserted->prev = current;

}

size++;

current = inserted;

}

T pop()

{

T tag;

Node<T>\* temp = current;

tag = temp->obj;

if (size == 1)

{

head = NULL;

current = NULL;

}

else

{

current->next->prev = current->prev;

current->prev->next = current->next;

current = current->next;

}

if (temp == head) head = head->next;

size--;

delete temp;

return tag;

}

void DeleteNode(int n)

{

Iterator<T> temp = head; Node<T>\* del;

if (IsNotEmpty())

{

n--; temp + n;

current = temp.GetCurrent();

del = current;

if (size == 1)

{

head = NULL;

current = NULL;

}

else

{

current->next->prev = current->prev;

current->prev->next = current->next;

current = current->next;

}

if (temp.GetCurrent() == head) head = head->next;

size--;

delete del;

}

}

void Delete()

{

for (int i = 0; i < size;) DeleteNode(i);

}

void sort()

{

for (int k = 0; k < 2; k++) {

Node<T>\* uns, \* min, \* temp;

uns = head;

do

{

temp = uns->next;

min = uns;

do

{

if (temp->obj < uns->obj) min = temp;

temp = temp->next;

} while (temp != head);

if (min != uns)

{

if (head == uns) head = min;

min->prev->next = min->next;

min->next->prev = min->prev;

uns->prev->next = min;

min->next = uns;

min->prev = uns->prev;

uns->prev = min;

}

else uns = uns->next;

} while (uns->next != head);

}

}

T& operator[](int numb)

{

Iterator<T> temp = head;

temp + numb;

return temp.GetCurrent()->obj;

}

Node<T>\* begin()

{

return head;

}

Node<T>\* tail()

{

return head->prev;

}

};

**Модуль Iterator**

“Iterator.h”

#pragma once

template<class T>

struct Node

{

T obj;

Node\* next;

Node\* prev;

};

template<class T>

class Iterator

{

private:

Node<T>\* current;

public:

Iterator(Node<T>\* first = nullptr) :current(first) { }

Iterator(const Iterator& obj)

{

current = obj.current;

}

void operator+(const int n)

{

for (int i = 0; i < n; i++) current = current->next;

}

void operator-(const int n)

{

for (int i = 0; i < n; i++) current = current->prev;

}

void operator++(int)

{

current = current->next;

}

void operator--(int)

{

current = current->prev;

}

void operator=(T data)

{

current->obj = data;

}

void operator=(Node<T>\* obj)

{

current = obj;

}

T& operator\*()

{

return current->obj;

}

bool operator ==(Node<T>\* obj)

{

return (current == obj);

}

Node<T>\* GetCurrent()

{

return current;

}

};

**Модуль File**

“File.h”

#pragma once

#include <iostream>

#include <string>

#include <iomanip>

#include <fstream>

#include<cstdio>

using namespace std;

class File

{

protected:

string fname;

ifstream fin;

ofstream fout;

public:

File()

{

fname = "\n";

}

File(string fn)

{

fname = fn;

}

~File()

{

}

};

**Модуль Text\_File**

“Text\_File.h”

#pragma once

#include <iostream>

#include <string>

#include <iomanip>

#include <fstream>

#include "File.h"

#include "File\_Except.h"

using namespace std;

template<class T>

class Text\_File : public File

{

public:

Text\_File() {}

Text\_File(string fn) : File(fn) { }

~Text\_File()

{

fout.close();

fin.close();

}

void Open\_in()

{

fin = InFile(fname);

}

void Open\_out()

{

fout = OutFile(fname);

}

void Read(T& obj)

{

fin >> obj;

}

void Remote()

{

fin.seekg(0, std::ios::beg);

fout.seekp(0, std::ios::beg);

}

void Write(T& obj)

{

fout << endl;

fout << obj;

}

bool End()

{

if (fin.eof()) return true;

else return false;

}

bool IsEmpty()

{

fin.peek();

if (fin.eof()) return true;

else return false;

}

};

**Модуль Company**

“Company.h”

#pragma once

#include <conio.h>

#include <iostream>

#include <iomanip>

#include <string>

#include <ctime>

#include <fstream>

#include "Time.h"

#include "vector"

#include "Input\_Except.h"

#include "Keys.h"

using namespace std;

class Company

{

private:

string CompanyName;

public:

Company()

{

CompanyName = "MyTrans";

}

Company(string name)

{

CompanyName = name;

}

void setcname(string nm)

{

CompanyName = nm;

}

string getcname()

{

return CompanyName;

}

};

**Модуль Transport**

“Transport.h”

#pragma once

#include "Company.h"

class Transport : public Company

{

protected:

vector<int> st;

int number;

public:

Transport()

{

number = 0;

}

Transport(int num, vector<int> s)

{

number = num;

st = s;

}

Transport(Transport& trans)

{

number = trans.number;

st = trans.st;

}

~Transport() { }

void setnumber(int num);

void setst(vector<int> st);

void pushst(int st);

unsigned int getnumber();

vector<int> getst();

Transport& operator=(Transport& trans);

friend istream& operator>>(istream& stream, Transport& T);

friend ostream& operator<<(ostream& stream, Transport& T);

friend ifstream& operator>>(ifstream& stream, Transport& T);

friend ofstream& operator<<(ofstream& stream, Transport& T);

};

“Transport.cpp”

#include "Transport.h"

void Transport::setnumber(int num)

{

number = num;

}

void Transport::setst(vector<int> s)

{

st = s;

}

void Transport::pushst(int s)

{

st.push\_back(s);

}

unsigned int Transport::getnumber()

{

return number;

}

vector<int> Transport::getst()

{

return st;

}

Transport& Transport::operator=(Transport& trans)

{

number = trans.number;

return \*this;

}

istream& operator>>(istream& stream, Transport& T)

{

return stream;

}

ostream& operator<<(ostream& stream, Transport& T)

{

stream << "||================ Route №" << setw(3) << T.number <<" ================||" << endl;

return stream;

}

ofstream& operator<<(ofstream& stream, Transport& T)

{

int size;

stream << setw(5) << T.number;

size = T.st.size();

stream << setw(5) << size;

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

stream << setw(5) << T.st[i];

return stream;

}

ifstream& operator>>(ifstream& stream, Transport& T)

{

int size;

stream >> T.number;

stream >> size;

T.st.resize(size);

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

stream >> T.st[i];

return stream;

}

**Модуль Ground\_Trans**

“Ground\_Trans.h”

#pragma once

#include "Transport.h"

#include "Ground\_Stops.h"

#include "Template.h"

using namespace std;

class Ground\_Trans : public Transport

{

protected:

Ring<class Ground\_Stops\*> stops;

vector<Time> time1, time2;

vector<Time> wtime1, wtime2;

public:

Ground\_Trans() : Transport() { }

Ground\_Trans(int num, Ring<class Ground\_Stops\*> st, vector<Time> t1, vector<Time> t2, vector<Time> wt1, vector<Time> wt2, vector<int> s) : Transport(num, s)

{

stops = st;

time1 = t1;

time2 = t2;

wtime1 = wt1;

wtime2 = wt2;

}

Ground\_Trans(Ground\_Trans& trans)

{

st = trans.st;

number = trans.number;

stops = trans.stops;

time1 = trans.time1;

time2 = trans.time2;

wtime1 = trans.wtime1;

wtime2 = trans.wtime2;

}

~Ground\_Trans() { }

void setstops(Ring<class Ground\_Stops\*> stops);

void settime1(vector<Time> t1);

void settime2(vector<Time> t2);

void setwtime1(vector<Time> t1);

void setwtime2(vector<Time> t2);

void pushstops(class Ground\_Stops\* st);

void delstops(int type);

void setstops(Ring<Ground\_Stops>& gs, int type);

vector<Time> fromtotime(bool a, bool b, int c);

Ring<class Ground\_Stops\*> getstops();

vector<Time> gettime1();

vector<Time> gettime2();

vector<Time> getwtime1();

vector<Time> getwtime2();

Ground\_Trans& operator=(Ground\_Trans& trans);

bool operator>(Ground\_Trans& T);

bool operator<(Ground\_Trans& T);

void info();

void stopinfo(int id);

friend istream& operator>>(istream& stream, Ground\_Trans& T);

friend ostream& operator<<(ostream& stream, Ground\_Trans& T);

friend ifstream& operator>>(ifstream& stream, Ground\_Trans& T);

friend ofstream& operator<<(ofstream& stream, Ground\_Trans& T);

};

“Ground\_Trans.cpp”

#include "Ground\_Trans.h"

using namespace std;

void Ground\_Trans::setstops(Ring<Ground\_Stops\*> st)

{

stops = st;

}

void Ground\_Trans::settime1(vector<Time> t1)

{

time1 = t1;

}

void Ground\_Trans::settime2(vector<Time> t2)

{

time2 = t2;

}

void Ground\_Trans::setwtime1(vector<Time> t1)

{

wtime1 = t1;

}

void Ground\_Trans::setwtime2(vector<Time> t2)

{

wtime2 = t2;

}

void Ground\_Trans::pushstops(class Ground\_Stops\* st)

{

stops.push(st);

}

void Ground\_Trans::delstops(int type)

{

for (int i = 0;i < stops.getsize();i++)

{

int j;

Ring<Ground\_Trans\*> tr = stops[i]->gettrans(type);

for (j = 0;j < tr.getsize();j++)

if (tr[j]->getnumber() == number)

break;

tr.DeleteNode(j);

stops[i]->settrans(type, tr);

}

stops.Delete();

st.clear();

}

void Ground\_Trans::setstops(Ring<Ground\_Stops>& gs, int type)

{

int num, i, k = 0, t = 0, f;

delstops(type);

cout << "Enter the id of the first station(from " << 1 << " to " << gs.getsize() << "): ";

for (i = 1; i < gs.getsize();i++)

if (gs[i].getid() > k) k = gs[i].getid();

do {

f = 0;

num = InInt(1, k);

for (i = 0; i < gs.getsize();i++)

if (num == gs[i].getid())

{

f++;

break;

}

if (!f) cout << "There are no stops with the same id, please try again: ";

} while (!f);

do

{

int id;

if (t)

id = gs[t].getid();

else id = -1;

t = i;

stops.push(&gs[i]);

st.push\_back(num);

gs[t].pushtrans(type, this);

vector<Nearest\_St> ns = gs[t].getnstops();

cout << "Enter the id of the next station (";

for (int j = 0;j < ns.size();j++)

{

f = 0;

for (int k = 0; k < st.size(); k++)

if (st[k] == ns[j].id)

{

f = 1;

break;

}

if (!f)

cout << ns[j].id << ' ';

}

cout << ")\n(0 - end of entering): ";

do {

f = 0;

num = InInt(0, k);

for (int j = 0; j < st.size(); j++)

if (st[j] == num)

{

f = -1;

break;

}

if (f != -1)

for (i = 0;i < ns.size();i++)

if (num == ns[i].id)

{

f++;

for (i = 0;gs[i].getid() != num;i++);

break;

}

if ((!f) & num) cout << "There are no nearest stops with the same id, please try again: ";

} while ((!f) & num);

if (num == 0) break;

} while (num);

}

Ring<Ground\_Stops\*> Ground\_Trans::getstops()

{

return stops;

}

vector<Time> Ground\_Trans::gettime1()

{

return time1;

}

vector<Time> Ground\_Trans::gettime2()

{

return time2;

}

vector<Time> Ground\_Trans::getwtime1()

{

return wtime1;

}

vector<Time> Ground\_Trans::getwtime2()

{

return wtime2;

}

vector<Time> Ground\_Trans::fromtotime(bool a, bool b, int c)

{

vector<Time> NewTime;

Time t;

int d, i;

if (a == 1)

{

d = -1;

i = stops.getsize() - 1;

if (b == 0)

NewTime = time1;

else NewTime = wtime1;

}

else

{

d = 1;

i = 0;

if (b == 0)

NewTime = time2;

else NewTime = wtime2;

}

for (i; stops[i]->getid() != c; i += d)

{

for (int l = 0; l < stops[i]->getnstops().size();l++)

if (stops[i]->getnstops()[l].id == stops[i + d]->getid())

{

t = stops[i]->getnstops()[l].time;

break;

}

for (int k = 0; k < NewTime.size();k++)

NewTime[k] = NewTime[k] + t;

}

return NewTime;

}

Ground\_Trans& Ground\_Trans::operator=(Ground\_Trans& trans)

{

number = trans.number;

st = trans.st;

stops = trans.stops;

time1 = trans.time1;

time2 = trans.time2;

wtime1 = trans.wtime1;

wtime2 = trans.wtime2;

return \*this;

}

bool Ground\_Trans::operator>(Ground\_Trans& T)

{

if (number > T.number)

return true;

else return false;

}

bool Ground\_Trans::operator<(Ground\_Trans& T)

{

if (number < T.number)

return true;

else return false;

}

void Ground\_Trans::info()

{

int wday = 1;

bool flag = 0, key = 0;

int dir = 1;

int d, i, size;

Time now;

now.now();

struct tm ltm;

time\_t nw = time(0);

localtime\_s(&ltm, &nw);

if (((ltm.tm\_wday == 0) || ((ltm.tm\_wday == 6) & (now.geth() > 5))) || ((ltm.tm\_wday == 1) & (now.geth() < 5)))

wday = 1;

do {

do {

if (dir == 2)

{

d = -1;

i = st.size() - 1;

size = -1;

}

else

{

d = 1;

i = 0;

size = st.size();

}

system("cls");

cout << "================= Route # " << setw(3) << number << " ==================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

cout << "||('Enter' - to choose the number of the stop)||" << endl;

if (wday == 1)

cout << "||== ('Left', 'Right' - [weekday]/weekend) ===||" << endl;

else

cout << "||== ('Left', 'Right' - weekday/[weekend]) ===||" << endl;

cout << "||== ('Up', 'Down' - change the direction) ===||" << endl;

cout << "|| ||" << endl;

for (i;i != size;i += d)

{

int k;

int f = 0;

cout << "|| " << setw(3) << i + 1 << ". " << stops[i]->getname() << setw(40 - stops[i]->getname().size()) << "||" << endl;

vector<Time> t = fromtotime(dir - 1, wday - 1, stops[i]->getid());

for (k = 0;k < t.size();k++)

if (t[k] > now)

{

f = 1;

break;

}

cout << "||";

if (!f)

cout << " The route is over ||" << endl;

else

{

int flag = 0;

for (k;k < t.size() && flag != 3;k++, flag++)

cout << setw(4) << t[k];

cout << setw(46 - 7 \* flag) << "||" << endl;

}

}

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::LEFT:

if (wday == 1)

wday = 2;

else wday--;

key++;

break;

case Keys::RIGHT:

if (wday == 2)

wday = 1;

else wday++;

key++;

break;

case Keys::UP:

if (dir == 1)

dir = 2;

else dir--;

key++;

break;

case Keys::DOWN:

if (dir == 2)

dir = 1;

else dir++;

key++;

break;

case Keys::ESC:

dir = 0;

flag++;

key++;

break;

case Keys::ENTER:

{

int id, flag = 0;

cout << "Enter the number of the stop: ";

id = InInt(1, st.size());

stopinfo(stops[i]->getid());

key++;

break;

}

}

}

} while (!key);

} while (!flag);

} while (dir);

}

void Ground\_Trans::stopinfo(int id)

{

int numb;

int wday = 1;

bool flag = 0, key = 0;

int dir = 1;

int d, i, size;

Time now;

now.now();

struct tm ltm;

time\_t nw = time(0);

localtime\_s(&ltm, &nw);

if ((ltm.tm\_wday == 0 || ltm.tm\_wday == 6 & now.geth() > 5) || (ltm.tm\_wday == 1 & now.geth() < 5))

wday = 1;

for (i = 0; i < stops.getsize();i++)

if (stops[i]->getid() == id)

break;

numb = i;

do {

do {

if (dir == 2)

{

d = -1;

i = st.size() - 1;

size = 0;

}

else

{

d = 1;

i = 0;

size = st.size() - 1;

}

system("cls");

cout << "================= Route # " << setw(3) << number << " ==================" << endl;

cout << "============== " << stops[numb]->getname() << setw(33 - stops[numb]->getname().size()) << " ================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

if (wday == 1)

cout << "||== ('Left', 'Right' - [weekday]/weekend) ===||" << endl;

else

cout << "||== ('Left', 'Right' - weekday/[weekend]) ===||" << endl;

cout << "||== ('Up', 'Down' - change the direction) ===||" << endl;

cout << "|| " << setw(15) << stops[i]->getname() << " - " << stops[size]->getname();

cout << setw(26 - stops[size]->getname().size()) << " ||" << endl;

int k;

int f = 0;

vector<Time> t = fromtotime(dir - 1, wday - 1, id);

for (k = 0; k < t.size(); k++)

{

f = 0;

cout << "|| ";

for (k;k < t.size() && f != 5;k++, f++)

cout << setw(4) << t[k];

cout << setw(45 - 7 \* f) << "||" << endl;

}

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::LEFT:

if (wday == 1)

wday = 2;

else wday--;

key++;

break;

case Keys::RIGHT:

if (wday == 2)

wday = 1;

else wday++;

key++;

break;

case Keys::UP:

if (dir == 1)

dir = 2;

else dir--;

key++;

break;

case Keys::DOWN:

if (dir == 2)

dir = 1;

else dir++;

key++;

break;

case Keys::ESC:

dir = 0;

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

} while (dir);

}

istream& operator>>(istream& stream, Ground\_Trans& T)

{

Time t, first, last;

int flag;

T.time1.clear();

T.time2.clear();

T.wtime1.clear();

T.wtime2.clear();

{

system("cls");

cout << "Enter the departure times of the transport from the first station on weekdays:" << endl;

cout << "Enter the time of the first time departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

first = InTime(24);

T.time1.push\_back(first);

cout << "\nEnter the time of the last time departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

last = InTime(24);

while (last < first)

{

cout << "This time must be more than" << first << endl;

cout << "Please, try again: ";

last = InTime(24);

}

flag = 0;

cout << "\n1.Enter the interval";

cout << "\n2.Enter the other times";

do {

switch (\_getch())

{

case '1':

flag = 1;

break;

case'2':

flag = 2;

break;

}

} while (!flag);

if (flag == 1)

{

cout << "\n\nEnter the interval of departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

t = InTime(12);

while (last > (first + t))

{

first += t;

T.time1.push\_back(first);

}

T.time1.push\_back(last);

}

else

{

cout << "\nEnter the other times(the last time - the end of the entering):";

cout << "\n(The input format must be like: hh.mm)" << endl;

do {

t = InTime(24);

if (t > first && t <= last)

{

T.time1.push\_back(t);

first = t;

}

else

{

cout << "Uncorrect input! Please, try again." << endl;

cout << "The time must be more than " << first << " and less than " << last << endl;

}

} while (t != last);

}

system("cls");

cout << "Enter the departure times of the transport from the last station on weekdays:" << endl;

cout << "Enter the time of the first time departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

first = InTime(24);

T.time2.push\_back(first);

cout << "\nEnter the time of the last time departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

last = InTime(24);

while (last < first)

{

cout << "This time must be more than" << first << endl;

cout << "Please, try again: ";

last = InTime(24);

}

flag = 0;

cout << "\n1.Enter the interval";

cout << "\n2.Enter the other times";

do {

switch (\_getch())

{

case '1':

flag = 1;

break;

case'2':

flag = 2;

break;

}

} while (!flag);

if (flag == 1)

{

cout << "\n\nEnter the interval of departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

t = InTime(12);

while ((first + t) < last)

{

first += t;

T.time2.push\_back(first);

}

T.time2.push\_back(last);

}

else

{

cout << "\nEnter the other times(the last time - the end of the entering):";

cout << "\n(The input format must be like: hh.mm)" << endl;

do {

t = InTime(24);

if (t > first && t <= last)

{

T.time2.push\_back(t);

first = t;

}

else

{

cout << "Uncorrect choise! Please, try again." << endl;

cout << "The time must be more than " << first << " and less than " << last << endl;

}

} while (t != last);

}

}

{

system("cls");

cout << "Enter the departure times of the transport from the first station on weekends:" << endl;

cout << "Enter the time of the first time departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

first = InTime(24);

T.wtime1.push\_back(first);

cout << "\nEnter the time of the last time departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

last = InTime(24);

while (last < first)

{

cout << "This time must be more than" << first << endl;

cout << "Please, try again: ";

last = InTime(24);

}

flag = 0;

cout << "\n1.Enter the interval";

cout << "\n2.Enter the other times";

do {

switch (\_getch())

{

case '1':

flag = 1;

break;

case'2':

flag = 2;

break;

}

} while (!flag);

if (flag == 1)

{

cout << "\n\nEnter the interval of departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

t = InTime(12);

while ((first + t) < last)

{

first += t;

T.wtime1.push\_back(first);

}

T.wtime1.push\_back(last);

}

else

{

cout << "\nEnter the other times(the last time - the end of the entering):";

cout << "\n(The input format must be like: hh.mm)" << endl;

do {

t = InTime(24);

if (t > first && t <= last)

{

T.wtime1.push\_back(t);

first = t;

}

else

{

cout << "Uncorrect input! Please, try again." << endl;

cout << "The time must be more than " << first << " and less than " << last << endl;

}

} while (t != last);

}

system("cls");

cout << "Enter the departure times of the transport from the last station on weekdays:" << endl;

cout << "Enter the time of the first time departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

first = InTime(24);

T.wtime2.push\_back(first);

cout << "\nEnter the time of the last time departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

last = InTime(24);

while (last < first)

{

cout << "This time must be more than" << first << endl;

cout << "Please, try again: ";

last = InTime(24);

}

flag = 0;

cout << "\n1.Enter the interval";

cout << "\n2.Enter the other times";

do {

switch (\_getch())

{

case '1':

flag = 1;

break;

case'2':

flag = 2;

break;

}

} while (!flag);

if (flag == 1)

{

cout << "\n\nEnter the interval of departure: ";

cout << "\n(The input format must be like: hh.mm)" << endl;

t = InTime(12);

while ((first + t) < last)

{

first += t;

T.wtime2.push\_back(first);

}

T.wtime2.push\_back(last);

}

else

{

cout << "\nEnter the other times(the last time - the end of the entering):";

cout << "\n(The input format must be like: hh.mm)" << endl;

do {

t = InTime(24);

if (t > first && t <= last)

{

T.wtime2.push\_back(t);

first = t;

}

else

{

cout << "Uncorrect choise! Please, try again." << endl;

cout << "The time must be more than " << first << " and less than " << last << endl;

}

} while (t != last);

}

}

return stream;

}

ostream& operator<<(ostream& stream, Ground\_Trans& T)

{

stream << "||" << setw(4) << T.getnumber() << '.' << T.stops[0]->getname() << " - " << T.stops[T.stops.getsize() - 1]->getname();

stream << setw(38 - T.stops[0]->getname().size() - T.stops[T.stops.getsize() - 1]->getname().size()) << "||" << endl;

return stream;

}

ofstream& operator<<(ofstream& stream, Ground\_Trans& T)

{

int size;

stream << dynamic\_cast<Transport&> (T);

size = T.time1.size();

stream << endl << setw(5) << size;

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

stream << T.time1[i];

size = T.time2.size();

stream << endl << setw(5) << size;

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

stream << T.time2[i];

size = T.wtime1.size();

stream << endl << setw(5) << size;

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

stream << T.wtime1[i];

size = T.wtime2.size();

stream << endl << setw(5) << size;

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

stream << T.wtime2[i];

return stream;

}

ifstream& operator>>(ifstream& stream, Ground\_Trans& T)

{

int size;

stream >> dynamic\_cast<Transport&> (T);

stream >> size;

T.time1.resize(size);

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

stream >> T.time1[i];

stream >> size;

T.time2.resize(size);

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

stream >> T.time2[i];

stream >> size;

T.wtime1.resize(size);

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

stream >> T.wtime1[i];

stream >> size;

T.wtime2.resize(size);

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

stream >> T.wtime2[i];

return stream;

}

**Модуль Underground\_Trans**

“Underground\_Trans.h”

#pragma once

#include "Transport.h"

#include "Underground\_Stops.h"

class Underground\_Trans : public Transport

{

protected:

vector<class Underground\_Stops\*> stops;

Time interval, winterval;

public:

Underground\_Trans() { }

Underground\_Trans(int num, Time t, Time wt, vector<int> s, vector<class Underground\_Stops\*> st) : Transport(num, s)

{

stops = st;

interval = t;

winterval = wt;

}

Underground\_Trans(Underground\_Trans& trans)

{

stops = trans.stops;

number = trans.number;

st = trans.st;

interval = trans.interval;

winterval = trans.winterval;

}

void setstops(vector<class Underground\_Stops\*> stops);

void setinterval(Time t);

void setwinterval(Time t);

void info();

void pushstops(class Underground\_Stops\* st);

vector<class Underground\_Stops\*> getstops();

Time getinterval();

Time getwinterval();

Underground\_Trans& operator=(Underground\_Trans& trans);

bool operator>(Underground\_Trans& T);

bool operator<(Underground\_Trans& T);

friend istream& operator>>(istream& stream, Underground\_Trans& T);

friend ostream& operator<<(ostream& stream, Underground\_Trans& T);

friend ifstream& operator>>(ifstream& stream, Underground\_Trans& T);

friend ofstream& operator<<(ofstream& stream, Underground\_Trans& T);

};

“Underground\_Trans.cpp”

#include "Underground\_Trans.h"

void Underground\_Trans::setstops(vector<class Underground\_Stops\*> st)

{

stops = st;

}

void Underground\_Trans::pushstops(class Underground\_Stops\* st)

{

stops.push\_back(st);

}

void Underground\_Trans::setinterval(Time t)

{

interval = t;

}

void Underground\_Trans::setwinterval(Time t)

{

winterval = t;

}

vector<class Underground\_Stops\*> Underground\_Trans::getstops()

{

return stops;

}

Time Underground\_Trans::getinterval()

{

return interval;

}

Time Underground\_Trans::getwinterval()

{

return winterval;

}

void Underground\_Trans::info()

{

cout << "||=============== Line # " << setw(3) << number << " ================||" << endl;

cout << "||---------------- Interval: ----------------||" << endl;

cout << "|| on weekdays: " << interval << " on weekends: " << winterval << setw(7) << "||" << endl;

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

cout << "||" << setw(4) << i + 1 << '.' << stops[i]->getname() << setw(40 - stops[i]->getname().size()) << "||" << endl;

cout << "||===========================================||" << endl;

}

Underground\_Trans& Underground\_Trans::operator=(Underground\_Trans& trans)

{

stops = trans.stops;

number = trans.number;

st = trans.st;

interval = trans.interval;

winterval = trans.winterval;

return \*this;

}

bool Underground\_Trans::operator>(Underground\_Trans& T)

{

if (number > T.number)

return true;

else return false;

}

bool Underground\_Trans::operator<(Underground\_Trans& T)

{

if (number < T.number)

return true;

else return false;

}

istream& operator>>(istream& stream, Underground\_Trans& T)

{

stream >> dynamic\_cast<Transport&> (T);

cout << "Enter the interval of arriving on weekdays: \n";

cout << "\n(The input format must be like: hh.mm)" << endl;

T.interval = InTime(1.00);

cout << "Enter the interval of arriving on weekends: \n";

cout << "\n(The input format must be like: hh.mm)" << endl;

T.winterval = InTime(1.00);

return stream;

}

ostream& operator<<(ostream& stream, Underground\_Trans& T)

{

stream << "||" << setw(4) << T.getnumber() << '.' << T.stops[0]->getname() << " - " << T.stops[T.stops.size() - 1]->getname();

stream << setw(38 - T.stops[0]->getname().size() - T.stops[T.stops.size() - 1]->getname().size()) << "||" << endl;

return stream;

}

ofstream& operator<<(ofstream& stream, Underground\_Trans& T)

{

stream << dynamic\_cast<Transport&> (T);

stream << T.interval;

stream << T.winterval;

return stream;

}

ifstream& operator>>(ifstream& stream, Underground\_Trans& T)

{

stream >> dynamic\_cast<Transport&> (T);

stream >> T.interval;

stream >> T.winterval;

return stream;

}

**Модуль Stops**

“Stops.h”

#pragma once

#include "Company.h"

#include "Structs.h"

#include "Template.h"

class Stops : public Company

{

protected:

StopsInfo Info;

vector<Nearest\_St> nearest;

public:

Stops()

{

Info.id = 0;

Info.name = '\0';

}

Stops(int i, string n, vector<Nearest\_St> nr)

{

Info.id = i;

Info.name = n;

nearest = nr;

}

Stops(Stops& st)

{

Info = st.Info;

Info.name = st.Info.name;

nearest = st.nearest;

}

~Stops() { }

void setid(int i);

void setname(string n);

void setnstops(vector<Nearest\_St> nr);

void pushnstops(Nearest\_St st);

bool operator>(Stops& S);

bool operator<(Stops& S);

int getid();

string getname();

vector<Nearest\_St> getnstops();

Stops& operator=(Stops& st);

friend istream& operator>>(istream& stream, Stops& S);

friend ostream& operator<<(ostream& stream, Stops& S);

friend ifstream& operator>>(ifstream& stream, Stops& S);

friend ofstream& operator<<(ofstream& stream, Stops& S);

};

“Stops.cpp”

#include "Stops.h"

void Stops::setid(int i)

{

Info.id = i;

}

void Stops::setname(string n)

{

Info.name = n;

}

void Stops::setnstops(vector<Nearest\_St> gs)

{

nearest = gs;

}

void Stops::pushnstops(Nearest\_St st)

{

nearest.push\_back(st);

}

int Stops::getid()

{

return Info.id;

}

string Stops::getname()

{

return Info.name;

}

vector<Nearest\_St> Stops::getnstops()

{

return nearest;

}

bool Stops::operator>(Stops& S)

{

if (getname() > S.getname())

return 1;

return 0;

}

bool Stops::operator<(Stops& S)

{

if (getname() < S.getname())

return 1;

return 0;

}

Stops& Stops::operator=(Stops& st)

{

Info = st.Info;

Info.name = st.Info.name;

nearest = st.nearest;

return \*this;

}

istream& operator>>(istream& stream, Stops& S)

{

cout << "\nEnter the name of the station: ";

S.Info.name = InString();

return stream;

}

ostream& operator<<(ostream& stream, Stops& S)

{

stream << S.Info.name << endl;

return stream;

}

ofstream& operator<<(ofstream& stream, Stops& S)

{

stream << setw(5) << S.Info.id;

return stream;

}

ifstream& operator>>(ifstream& stream, Stops& S)

{

stream >> S.Info.id;

return stream;

}

**Модуль Ground\_ Stops**

“Ground\_Stops.h”

#pragma once

#include "Stops.h"

#include "Ground\_Trans.h"

#include "Template.h"

class Ground\_Stops : public Stops

{

private:

Ring<class Ground\_Trans\*> buses;

Ring<class Ground\_Trans\*> trams;

Ring<class Ground\_Trans\*> trolleybuses;

public:

Ground\_Stops() : Stops() {};

Ground\_Stops(int i, string n, vector<Nearest\_St> nr, Ring<class Ground\_Trans\*> bus, Ring<class Ground\_Trans\*> tram, Ring<class Ground\_Trans\*> trolleybus) : Stops(i, n, nr)

{

buses = bus;

trams = tram;

trolleybuses = trolleybus;

}

Ground\_Stops(Ground\_Stops& st)

{

Info = st.Info;

buses = st.buses;

trams = st.trams;

trolleybuses = st.trolleybuses;

nearest = st.nearest;

}

void settrans(int flag, Ring<class Ground\_Trans\*> trans);

void pushtrans(int flag, Ground\_Trans\* trans);

void setnstops(Ring<Ground\_Stops>& gs);

Ring<class Ground\_Trans\*> gettrans(int flag);

void info();

Ground\_Stops& operator=(Ground\_Stops& st);

friend istream& operator>>(istream& stream, Ground\_Stops& S);

friend ostream& operator<<(ostream& stream, Ground\_Stops& S);

friend ifstream& operator>>(ifstream& stream, Ground\_Stops& S);

friend ofstream& operator<<(ofstream& stream, Ground\_Stops& S);

};

“Ground\_Stops.cpp”

#include "Ground\_Stops.h"

void Ground\_Stops::settrans(int flag, Ring<Ground\_Trans\*> trans)

{

switch (flag)

{

case 1:

buses = trans;

break;

case 2:

trolleybuses = trans;

break;

case 3:

trams = trans;

break;

}

}

void Ground\_Stops::pushtrans(int flag, Ground\_Trans\* trans)

{

switch (flag)

{

case 1:

buses.push(trans);

break;

case 2:

trolleybuses.push(trans);

break;

case 3:

trams.push(trans);

break;

}

}

void Ground\_Stops::setnstops(Ring<Ground\_Stops> &gs)

{

int num, i, k;

vector<Nearest\_St> stations;

Nearest\_St st;

system("cls");

cout << "Enter the amount of the nearest ground stations:\n";

if (gs.getsize() > 4)

num = InInt(1, 4);

else num = InInt(1, getid() - 1);

for (i = 0;i < num;i++)

{

int f;

cout << i + 1 << ")ID(from " << 1 << " to " << gs.getsize() - 1 << "): ";

do {

f = 0;

st.id = InInt(1, gs.getsize() - 1);

for (int j = 0; j < stations.size();j++)

if (st.id == stations[j].id || st.id == getid())

{

f = -1;

cout << "This station is allready included, please, try again: ";

}

if (f == 0)

for (k = 0; k < gs.getsize();k++)

if (st.id == gs[k].getid())

{

f = 1;

break;

}

if (!f) cout << "There are no stops with the same id, please try again: ";

} while (f <= 0);

cout << "Time to get to: ";

cin >> st.time;

stations.push\_back(st);

st.id = getid();

gs[k].pushnstops(st);

}

nearest = stations;

}

Ring<Ground\_Trans\*> Ground\_Stops::gettrans(int flag)

{

switch (flag)

{

case 1:

return buses;

case 2:

return trolleybuses;

case 3:

return trams;

}

return 0;

}

void Ground\_Stops::info()

{

bool flag = 0, key = 0;

int vflag = 1;

int hflag = 1;

if (buses.getsize() != 0)

buses.sort();

if (trolleybuses.getsize() != 0)

trolleybuses.sort();

if (trams.getsize() != 0)

trams.sort();

do {

do {

flag = 0;

rewind(stdin);

system("cls");

cout << "============== " << getname() << setw(33 - getname().size());

cout << " ================" << endl;

cout << "||==========(Press Esc to turn back)==========||" << endl;

cout << "||('Enter'- to choose the number of the route)||" << endl;

if (hflag == 1)

{

cout << "||---[1.Buses]-- 2.Trolleybuses -- 3.Trams ---||" << endl;

for (int i = 0; i < buses.getsize();i++)

cout << \*buses[i];

if (!buses.getsize())

cout << "|| The list is empty! ||" << endl;

}

else if (hflag == 2)

{

cout << "||--- 1.Buses --[2.Trolleybuses]-- 3.Trams ---||" << endl;

for (int i = 0; i < trolleybuses.getsize();i++)

cout << \*trolleybuses[i];

if (!trolleybuses.getsize())

cout << "|| The list is empty! ||" << endl;

}

else if (hflag == 3)

{

cout << "||--- 1.Buses -- 2.Trolleybuses --[3.Trams]---||" << endl;

for (int i = 0; i < trams.getsize();i++)

cout << \*trams[i];

if (!trams.getsize())

cout << "|| The list is empty! ||" << endl;

}

cout << "||============================================||" << endl;

do {

key = 0;

if (\_kbhit())

{

switch (static\_cast<Keys>(\_getch()))

{

case Keys::LEFT:

if (hflag == 1)

hflag = 3;

else hflag--;

key++;

break;

case Keys::RIGHT:

if (hflag == 3)

hflag = 1;

else hflag++;

key++;

break;

case Keys::ESC:

vflag = 0;

flag++;

key++;

break;

case Keys::ENTER:

flag++;

key++;

break;

}

}

} while (!key);

} while (!flag);

if (vflag)

{

int flag = 0;

switch (hflag)

{

case 1:

{

if (buses.getsize() != 0)

{

Time t;

int numb, i;

cout << "Enter the number of the route: ";

do

{

numb = InInt(1, 100);

for (i = 0; i < buses.getsize() && buses[i]->getnumber() != numb;i++);

if (buses[i]->getnumber() == numb)

{

numb = i;

flag++;

}

else

cout << "There are no transport with this number.\nPlease, try again: ";

} while (!flag);

buses[numb]->stopinfo(getid());

}

else

{

cout << "There are no objects with the type 'bus'!" << endl;

system("pause");

}

break;

}

case 2:

{

if (trolleybuses.getsize() != 0)

{

int numb, i;

cout << "Enter the number of the route: ";

do

{

numb = InInt(1, trolleybuses[trolleybuses.getsize()]->getnumber());

for (i = 0; i < trolleybuses.getsize() && trolleybuses[i]->getnumber() != numb;i++);

if (trolleybuses[i]->getnumber() == numb)

{

numb = i;

flag++;

}

else

cout << "There are no transport with this number.\nPlease, try again: ";

} while (!flag);

trolleybuses[numb]->stopinfo(getid());

}

else

{

cout << "There are no objects with the type 'trolleybus'!" << endl;

system("pause");

}

break;

}

case 3:

{

if (trams.getsize() != 0)

{

int numb, i;

cout << "Enter the number of the route: ";

do

{

numb = InInt(1, trams[trams.getsize()]->getnumber());

for (i = 0; i < trams.getsize() && trams[i]->getnumber() != numb;i++);

if (trams[i]->getnumber() == numb)

{

numb = i;

flag++;

}

else

cout << "There are no transport with this number.\nPlease, try again: ";

} while (!flag);

trams[numb]->stopinfo(getid());

}

else

{

cout << "There are no objects with the type 'tram'!" << endl;

system("pause");

}

break;

}

}

}

} while (vflag);

}

Ground\_Stops& Ground\_Stops::operator=(Ground\_Stops& st)

{

Info = st.Info;

buses = st.buses;

trams = st.trams;

trolleybuses = st.trolleybuses;

nearest = st.nearest;

return \*this;

}

istream& operator>>(istream& stream, Ground\_Stops& S)

{

stream >> dynamic\_cast<Stops&> (S);

return stream;

}

ostream& operator<<(ostream& stream, Ground\_Stops& S)

{

stream << "||" << setw(3) << S.getid() << ". " << S.getname() << setw(41 - S.getname().size()) << "||" << endl;

return stream;

}

ofstream& operator<<(ofstream& stream, Ground\_Stops& S)

{

stream << dynamic\_cast<Stops&> (S);

stream << setw(5) << S.nearest.size();

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

{

stream << setw(5);

stream << S.nearest[i];

}

return stream;

}

ifstream& operator>>(ifstream& stream, Ground\_Stops& S)

{

int size;

stream >> dynamic\_cast<Stops&> (S);

stream >> size;

S.nearest.resize(size);

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

stream >> S.nearest[i];

return stream;

}

**Модуль Underground\_ Stops**

“Underground\_Stops.h”

#pragma once

#include "Stops.h"

class Underground\_Stops : public Stops

{

private:

int line;

public:

Underground\_Stops() : Stops() { line = 0; }

Underground\_Stops(int i, string n, vector<Nearest\_St> nr, int l) : Stops(i, n, nr)

{

line = l;

}

Underground\_Stops(Underground\_Stops& st)

{

Info = st.Info;

line = st.line;

nearest = st.nearest;

}

~Underground\_Stops() { }

void setline(int l);

void setnstops(Ring<Underground\_Stops>& us);

int getline();

Underground\_Stops& operator=(Underground\_Stops& st);

friend istream& operator>>(istream& stream, Underground\_Stops& S);

friend ostream& operator<<(ostream& stream, Underground\_Stops& S);

friend ifstream& operator>>(ifstream& stream, Underground\_Stops& S);

friend ofstream& operator<<(ofstream& stream, Underground\_Stops& S);

};

“Underground\_Stops.cpp”

#include "Underground\_Stops.h"

void Underground\_Stops::setline(int l)

{

line = l;

}

void Underground\_Stops::setnstops(Ring<Underground\_Stops>& us)

{

int id1, id2, j, f, k;

vector<Nearest\_St> stations;

Nearest\_St st;

cout << "Enter the information about the near station: " << endl;

id1 = 0;

id2 = 0;

for (j = 0;j < us.getsize() - 1;j++)

if (us[j].getline() == line && us[j].getnstops().size() <= 1)

{

id1 = us[j].getid();

j++;

break;

}

for (j;j < us.getsize() - 1;j++)

if (us[j].getline() == line && us[j].getnstops().size() <= 1)

{

id2 = us[j].getid();

break;

}

if (id2 == 0) id2 = id1;

cout << "ID(" << id1 << " or " << id2 << "): ";

do {

f = 0;

st.id = InInt(1, us[us.getsize() - 2].getid());

for (k = 0; k < us.getsize();k++)

if ((st.id == us[k].getid()) & (st.id == id1 || st.id == id2))

{

f++;

break;

}

if (!f) cout << "There are no stops with the same id or this\nstop doesen`t suit, please try again: ";

} while (!f);

cout << "Time to get to: ";

cin >> st.time;

stations.push\_back(st);

nearest = stations;

st.id = getid();

us[k].pushnstops(st);

}

int Underground\_Stops::getline()

{

return line;

}

Underground\_Stops& Underground\_Stops::operator=(Underground\_Stops& st)

{

Info = st.Info;

line = st.line;

nearest = st.nearest;

return \*this;

}

istream& operator>>(istream& stream, Underground\_Stops& S)

{

stream >> dynamic\_cast<Stops&> (S);

return stream;

}

ostream& operator<<(ostream& stream, Underground\_Stops& S)

{

stream << "||" << setw(4) << S.getid() << '.' << S.getname() << setw(41 - S.getname().size()) << "||" << endl;

return stream;

}

ofstream& operator<<(ofstream& stream, Underground\_Stops& S)

{

stream << dynamic\_cast<Stops&> (S);

stream << setw(5) << S.line;

stream << setw(5) << S.nearest.size();

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

{

stream << setw(5);

stream << S.nearest[i];

}

return stream;

}

ifstream& operator>>(ifstream& stream, Underground\_Stops& S)

{

int size;

stream >> dynamic\_cast<Stops&> (S);

stream >> S.line;

stream >> size;

S.nearest.resize(size);

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

stream >> S.nearest[i];

return stream;

}