МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Объектно-ориентированное программирование»

Тема: Создание консольной игры на C++

Студент гр. 6382 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кацер И.С.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Филатов А.Ю.

Санкт-Петербург

2018

**Цель работы.**

Реализовать класс Объект, а также класс Поле боя, являющийся обёрткой над контейнерами-армиями, хранящим экземпляры класса Объект.

Объект должен иметь:

1. Двумерные координаты.

2. Очки жизней.

3. Говорящий конструктор, говорящий деструктор.

4. Функцию, которая принимает урон.

5. Функцию, отвечающую на вопрос, есть ли Объект на указан-ной позиции.

6. Считываться из файла.

Поле боя должно иметь:

1. Контейнеры, хранящие объекты.

2. Отрисовку Поля боя на экране (разными цветами).

3. Загружать объекты из файла

4. Определять, располагается ли на заданных координатах объ-ект той или иной армии.

**Описание программы.**

Программа содержит класс Pole, который описывает поле игры и включа-ет в себя два контейнера, содержащие объекты разных армий. Объекты описа-ны в классе Object и имеют координаты в игровом поле и очки жизней.

Поля класса Pole:

int maxx, maxy – размеры поля

List<obj> red, green; – списки для хранения объектов красной и зеленой армий

В классе Pole представлены следующие функции:

1. int prov(const int& x, const int& y) - проверка на наличие объекта из той или иной армии

2. int damage (const int& x, const int& y,const int& d) -функция, которая наносит урон(объект обязательно должен быть)

3. bool fromfile ( std::ifstream &f) – функция для чтения данных из файла

4. void showPole() – функция для вывода

**Пример работы программы:**

Размер поля: 10 10

Объект: х=0 у=0 hp=100

~Объект: х=0 у=0 hp=80

Объект: х=1 у=1 hp=100

~Объект: х=1 у=1 hp=100

Объект: х=2 у=2 hp=100

~Объект: х=2 у=2 hp=100

Объект: х=0 у=2 hp=100

~Объект: х=0 у=2 hp=100

Объект: х=9 у=9 hp=100

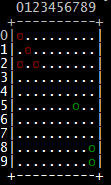
~Объект: х=9 у=9 hp=100

Объект: х=9 у=8 hp=100

~Объект: х=9 у=8 hp=100

Объект: х=7 у=5 hp=100

~Объект: х=7 у=5 hp=100



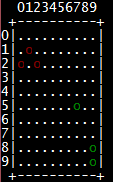
Введите координаты цели(через пробел)(q для выхода)

0 0

В выбранной точке поля находится боец красной армии

Введите урон: 100

Оставшиеся очки здоровья: ~Объект: х=0 у=0 hp=0



Введите координаты цели(через пробел)(q для выхода)

~Объект: х=1 у=1 hp=100

~Объект: х=2 у=2 hp=100

~Объект: х=0 у=2 hp=100

~Объект: х=9 у=9 hp=100

~Объект: х=9 у=8 hp=100

~Объект: х=7 у=5 hp=100

**Выводы.**

В результате выполнения данной лабораторной работы был реализован класс Объект, а также класс Поле\_боя, являющийся обёрткой над контейнера-ми-армиями, хранящим экземпляры класса Объект.

**ПРИЛОЖЕНИЕ A (ИСХОДНЫЙ КОД)**

**MAKEFILE**

all: main.o main.cpp

g++ main.o -o out -static -std=c++11

main.o: main.cpp list.h obj.h Pole.h

g++ main.cpp -c -o main.o -static -std=c++11

**POLE.H**

#ifndef PoleH

#define PoleH

#include "List.h"

#include "color.h"

#include "obj.h"

#include <iostream>

#include <fstream>

class Pole{

private:

int maxx, maxy;

List<obj> red, green; //армия

public:

Pole()

{

}

//декструктор

~Pole()

{

red.~List();

green.~List();

}

bool fromfile ( std::ifstream &f)//из файла

{

int kol;

if (!f.is\_open()) return false;

f >> maxx;

f >> maxy;

std::cout<<"Размер поля: "<<maxx<<" "<<maxy<< std::endl;

f >> kol; //кол-во бойцов в красной армии

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

{

obj \*r = new obj(f);//создаём объект и передаем в конструктор указатель на файл

red.AddEnd(\*r);//добавляем объект в красную армию

delete r;

}

f >> kol; //кол-во бойцов в зеленой армии

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

{

obj \*r = new obj(f);//создаём объект и передаем в конструктор указатель на файл

green.AddEnd(\*r);//добавляем объект в красную армию

delete r;

}

}

int prov(const int& x, const int& y) //проверка на наличие объекта из той или иной армии

{

List<obj>::Iterator p;

if ((x>maxx) || (y>maxy) || (x<0) || (y<0)) return -1; //если вне поля

for(p=red.Begin(); p!=red.End(); p++)

{

//возвращает элемент нашего списка по итератору >>

if ((\*p).issure(x,y)) return 1; //для наличия из красной армии возвращаем 1

}

for(p=green.Begin(); p!=green.End(); p++)

{

if ((\*p).issure(x,y)) return 2; //для наличия из зеленой армии возвращаем 2

}

return 0; //если объекта нет

}

int damage (const int& x, const int& y,const int& d) //функция, которая наносит урон(объект обязательно должен быть)

{

int h;

List<obj>::Iterator p;

for(p=red.Begin(); p!=red.End(); p++)

{

if ((\*p).issure(x,y))

{

h=(\*p).damage(d);//наносим урон и сохраняем текущее здоровье в переменную h

if (h>0)

return h; //для наличия из красной армии возвращаем здоровье

else

{

red.Delete(p);

return 0; //возвращаем 0 если боец убит

}

}

}

for(p=green.Begin(); p!=green.End(); p++)

{

if ((\*p).issure(x,y))

{

h=(\*p).damage(d); //наносим урон и сохраняем текущее здоровье в переменную h

if (h>0)

return h; //для наличия из зеленый армии возвращаем здоровье

else

{

red.Delete(p);

return 0; //возвращаем 0 если боец убит

}

}

}

}

void showPole() //вывод

{

std::cout << " ";

int o;

for(int a=0; a<maxy; a++)

{

std::cout<< a;

}

std::cout << std::endl<< " +";

for(int a=0; a<maxy; a++)

{

std::cout << "-" ;

}

std::cout << "+" << std::endl;

for (int j=0; j<maxy; j++)

{

std::cout<< j <<"|";

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

{

o=prov(i,j);

if (o==0) std::cout<<".";

if (o==1)

{

SetColor(Red , 0);

std::cout<<"o";

SetColor(White , Black);

}

if (o==2)

{

SetColor(Green , 0);

std::cout<<"o";

SetColor(White , Black);

}

}

std::cout<< "|"<<std::endl;

}

std::cout << " +";

for(int a=0; a<maxy; a++)

{

std::cout << "-" ;

}

std::cout << "+" << std::endl;

}

};

#endif

**LIST.H**

//----------------------------------------------------------------------------

#ifndef ListH

#define ListH

#include <iostream>

//----------------------------------------------------------------------------

template <class T>

class List {

private:

struct Node

{

T Data;

Node \*Prev, \*Next;

Node(const T& data) // конструктор для удобства

: Data(data), // копируем данные;

Prev(NULL), // предыдущего и

Next(NULL) // последующего узлов нет

{

}

};

Node \*Head, // первый узел списка

\*Tail; // последний узел

// Делаем конструктор копирования и оператор присваивания закрытыми.

List(const List&) { }

void operator=(const List&) { }

public:

class Iterator {

// Класс список объявляется дружественным для доступа к указателю на узел.

friend List<T>;

Node\* P; // ук-ль на узел списка, содержащий значение эл-та, на кот-й указывает итератор

public:

Iterator(Node\* p = NULL)

: P(p)

{

}

// разыменовывание для константного итератора.

// Возвращает константую ссылку на значение эл-та.

const T& operator \*() const

{

return P->Data;

}

// разыменовывание для не константного итератора.

// Возвращает не константного ссылку на значение эл-та, по которой

// оно может быть изменено.

T& operator \*()

{

return P->Data;

}

// Префиксная операция инкремента. Перемещает итератор к следующему эл-ту списка.

Iterator operator ++()

{

P = P->Next; // переходим к следующему эл-ту

return \*this; // возвращаем "увеличенный" итератор

}

// Постфиксная операция инкремента. Перемещает итератор к следующему эл-ту списка.

// Аргумент указывает компилятору, что данная функция-операция operator ++ является постфиксной.

Iterator operator ++(int)

{

Iterator t(\*this);

P = P->Next;

return t;

}

// Префиксная операция декремента. Перемещает итератор к предыдущему эл-ту списка.

Iterator operator --()

{

P = P->Prev;

return \*this;

}

// Постфиксная операция декремента. Перемещает итератор к предыдущему эл-ту списка.

// Аргумент указывает компилятору, что данная функция-операция operator -- является постфиксной.

Iterator operator --(int)

{

Iterator t(\*this);

P = P->Prev;

return t;

}

// Операция равенства.

friend bool operator ==(const Iterator& x,const Iterator& y)

{

return y.P == x.P;

}

// Операция неравенства.

friend bool operator !=(const Iterator& x,const Iterator& y)

{

return y.P != x.P;

}

};

// Конструктор пустого списка

List()

: Head(NULL), Tail(NULL)

{

}

// Деструктор (освобождает память).

~List()

{

Clear();

}

// Возвращает константный итератор на первый элемент (для константного списка).

const Iterator Begin() const

{

return Iterator(Head);

}

// Возвращает итератор на первый элемент (для не константного списка).

Iterator Begin()

{

return Iterator(Head);

}

// Возвращает константный итератор на последний элемент (для константного списка).

const Iterator Last() const

{

return Iterator(Tail);

}

// Возвращает итератор на последний элемент (для не константного списка).

Iterator Last()

{

return Iterator(Tail);

}

// Возвращает константный итератор на "конец" (для константного списка).

const Iterator End() const

{

return Iterator(NULL);

}

// Возвращает итератор на "конец" (для не константного списка).

Iterator End()

{

return Iterator(NULL);

}

// Проверка пустоты списка.

bool Empty() const

{

return !Head;

}

// Вставляет элемент в список перед элементом, на который указывает номер.

// Если список меньше, то эл-т добавляется в конец списка.

void Insert(const int nomer, const T& data);

// Удаляет эл-т, на который указывает итератор номер.

// Если список меньше, то эл-т удаляется с конца списка.

void Delete(const int nomer);

// Удаляет эл-т, на который указывает итератор iter.

void Delete(const Iterator& iter);

// Удаляет все эл-ты из списка.

void Clear();

// Добавление эл-та в начало списка.

void AddBegin(const T& data) ;

// Добавление эл-та в конец списка.

void AddEnd(const T& data) ;

// Удаление первого эл-та.

void DeleteFirst();

// Удаление последнего эл-та.

void DeleteLast();

// операция вывода списка

show()

{

Iterator i;

std::cout << '[';

// Выводим все элементы кроме последнего, разделяя их запятой.

for (i = Begin(); i != Last(); i++)

std::cout << \*i << ", ";

// Выводим последний элемент (если список не пуст).

if (i != End()) std::cout << \*i;

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

}

};

//---------------------------------------------------------------------------

// Определение не встраиваемых функций-элементов

//---------------------------------------------------------------------------

template <class T>

void List<T>::Insert(const int nomer, const T& data)

{

// Если список не пуст, то

if ((Head) && (nomer!=0))

{

int m;

Iterator i;

m=0;

i=Begin();

// Ищем узел X, перед которым нужно вставить.

while( (m<nomer) && (i!= End()) )

{

m++;

i++;

}

// Если iter указывает на "конец", то добавляем эл-т в конец списка.

if (i==End())

{

AddEnd(data);

}

// Иначе вставляем эл-т в "середину" списка.

else

{

Node \*t,

\*n = new Node(data); // создаем отдельный узел

t = i.P->Prev; // в t заносим предшествующий X

t->Next = n; // связываем предшествующий X с новым

n->Prev = t; // связываем новый с предшествующим X

n->Next = i.P; // связываем новый с X

i.P->Prev = n; // связываем X с новым

}

}

// Иначе вставляем эл-т в начало

else

{

AddBegin(data);

}

}

//---------------------------------------------------------------------------

template <class T>

void List<T>::Delete(const int nomer)

{

// Если список не пуст, то

if (Head)

{

int m=0;

Iterator i;

i=Begin();

// Ищем узел X, который нужно удалить.

while( (m<nomer) && (i!= End()) )

{

m++;

i++;

}

// Если X первый.

if (i == Begin())

{

DeleteFirst();

}

// Иначе, если X последний.

else if ((i==Last()) || (i==End()))

{

void DeleteLast();

}

// Иначе X средний.

else

{

Node \*t, \*pred, \*sled;

t=i.P; //удаляемый элемент

pred=t->Prev; //предыдущий

sled=t->Next; //следующй

pred->Next = t->Next; // связываем предыдущий X со следующим за X,

sled->Prev = t->Prev; // отделяя X от списка

delete t; // Удаляем (уничтожаем и освобождаем память).

}

}

}

//---------------------------------------------------------------------------

template <class T>

void List<T>::Delete(const Iterator& iter)

{

Node \*t;

Iterator i;

// Ищем узел X, содержащий удаляемый эл-т.

for (i = Begin(); i != End(); i++)

if (i == iter) break;

// Если не найден, то iter не корректен: ошибка.

//assert(i != End());

t = i.P;

// Если X первый.

if (i == Begin()) {

// Смещаем первый узел на следующий за ним.

Head = t->Next;

// Если он отсутсвует, то список состоит из одного эл-та и

// нужно сбросить указатель на последний.

if (!Head) Tail = NULL;

}

else

// Иначе, если X последний.

if (i == Last()) {

// Смещаем последний узел на ему предшествующий.

Tail = t->Prev;

// Если он отсутсвует, то список состоит из одного эл-та и

// нужно сбросить указатель на первый.

if (!Tail) Head = NULL;

}

else {

// Иначе X средний.

t->Prev->Next = t->Next; // связываем предыдущий X со следующим за X,

t->Next->Prev = t->Prev; // отделяя X от списка

}

// Удаляем X (уничтожаем и освобождаем память).

delete t;

}

//---------------------------------------------------------------------------

template <class T>

void List<T>::Clear()

{

// Если список не пуст.

if (!Empty()) {

Node \*n;

Iterator i;

// Удаляем все узлы.

for (i = Begin(); i != End(); ) {

n = i.P; // сохраняем адрес узла

i++; // переходим к следующему

delete n; // и удаляем текущий

}

// Сбрасываем указатели на первый и последний.

Head = Tail = NULL;

}

}

//---------------------------------------------------------------------------

//Добавление эл-та в начало списка.

template <class T>

void List<T>::AddBegin(const T& data)

{

Node \*n=new Node(data); // создаем отдельный узел

// Если список не пуст, то

if (Head)

// связываем первый узел с новым;

Head->Prev = n;

else

// иначе новый первый узел является и последним.

Tail = n;

// Связываем новый узел с первым,

n->Next = Head;

// и делаем его первым.

Head = n;

}

//---------------------------------------------------------------------------

//Добавление эл-та в конец списка.

template <class T>

void List<T>::AddEnd(const T& data)

{

Node \*n=new Node(data); // создаем отдельный узел

// Если список не пуст, то

if (Head)

// связываем последний узел с новым;

Tail->Next = n;

else

// иначе новый последний узел является и первым.

Head = n;

// Связываем новый узел с последним,

n->Prev = Tail;

// и делаем новый последним.

Tail = n;

}

//---------------------------------------------------------------------------

// Удаление первого эл-та.

template <class T>

void List<T>::DeleteFirst()

{

Node \*t=Head;

// Смещаем первый узел на следующий за ним.

Head = t->Next;

// Если он отсутсвует, то список состоит из одного эл-та и

// нужно сбросить указатель на последний.

if (!Head) Tail = NULL;

// Удаляем (уничтожаем и освобождаем память).

delete t;

}

//---------------------------------------------------------------------------

// Удаление последнего эл-та.

template <class T>

void List<T>::DeleteLast()

{

Node \*t=Tail;

// Смещаем последний узел на ему предшествующий.

Tail = t->Prev;

// Если он отсутсвует, то список состоит из одного эл-та и

// нужно сбросить указатель на первый.

if (!Tail) Head = NULL;

// Удаляем (уничтожаем и освобождаем память).

delete t;

}

//---------------------------------------------------------------------------

#endif

**MAIN.CPP**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include "List.h"

#include "obj.h"

#include "Pole.h"

using std::cout;

using std::cin;

using std::endl;

typedef List<int> L;

int main()

{

setlocale(LC\_ALL, "Russian");

std::ifstream fin("1.txt");

int x,y,temp, damag;

Pole map;

map.fromfile(fin);

map.showPole();

x=y=-1;

std::cout<<"ведите координаты цели (через пробел)(q для выхода) \n";

std::cin>>x>>y;

while(y!=-1)

{

temp=map.prov(x, y);

if(temp==0) std::cout<< "в выбранной точке поля нет боцов \n";

if(temp==1) std::cout<< "в выбранной точке поля находится боец красной армии \n";

if(temp==2) std::cout<< "в выбранной точке поля находится боец зеленой армии \n";

if(temp==-1) std::cout<< "выбранная точка находится вне поля \n";

if ((temp!=-1) && (temp!=0))

{

std::cout<<"Введите урон: ";

std::cin>>damag;

std::cout<<"\nОставшиеся очки здоровья: ";

std::cout<<map.damage(x,y,damag);

std::cout<<std::endl;

map.showPole();

}

x=y=-1;

std::cout<<"ведите координаты цели (через пробел)(q для выхода) \n ";

std::cin>>x>>y;

}

}

**OBJ.H**

#ifndef objH

#define objH

#include <iostream>

#include <fstream>

class obj {

private:

int x, y, heal; //кооординаты и кол-во очков

public:

obj (const int& a, const int& b, const int& h) //конструктор

{

x=a;

y=b;

heal=h;

std::cout<<"Объект: х="<<x<<" у="<<y<<" hp="<<heal<< std::endl;

}

obj (std::ifstream &f) //конструктор

{

if (!f.is\_open()) exit(1); // если файл не открыт

f >> x;

f >> y;

f >> heal;

std::cout<<"Объект: х="<<x<<" у="<<y<<" hp="<<heal<< std::endl;

}

obj()

{

x=-1;

y=-1;

heal=-1;

std::cout<<"Объект: х="<<x<<" у="<<y<<" hp="<<heal<< std::endl;

}

~obj() //декструкор

{

std::cout<<"~Объект: х="<<x<<" у="<<y<<" hp="<<heal<< std::endl;

}

int damage (const int& d) //функция, которая наносит урон

{

//std::cout<<heal<<" урон" <<d<<"\n";

heal=heal-d;

return heal;

}

bool issure(const int& x1, const int& y1) //функция, которая проверяет наличие объекта на координатах

{

if ((x1==x) && (y1==y))

return true;

else

return false;

}

bool file(std::ifstream &f) //для считывания из файла

{

if (!f.is\_open()) return false; // если файл не открыт

f >> x;

f >> y;

f >> heal;

//std::cout<<"Объект: х="<<x<<" у="<<y<<" hp="<<heal<< std::endl;

return true; //указатель на себя

}

void show() //вывод

{

std::cout << "Координаты x= " << x << std::endl;

std::cout << "Координаты y= " << y << std::endl;

std::cout << "Количество очков/здоровье heal= " << heal << std::endl;

}

};

#endif

**COLOR.H**

#ifndef colorH

#define colorH

#include <iostream>

#include <fstream>

#include <Windows.h>

enum ConsoleColor

{

Black = 0,

Blue = 1,

Green = 2,

Cyan = 3,

Red = 4,

Magenta = 5,

Brown = 6,

LightGray = 7,

DarkGray = 8,

LightBlue = 9,

LightGreen = 10,

LightCyan = 11,

LightRed = 12,

LightMagenta = 13,

Yellow = 14,

White = 15

};

void SetColor(int text, int background) //функция, которая переключает цвет текста и фона(принимает int)

{

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, (WORD)((background << 4) | text));

}

void SetColor(ConsoleColor text, ConsoleColor background) //функция, которая переключает цвет текста и фона(принимает название цвета)

{

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, (WORD)((background << 4) | text));

}

#endif