**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Консольная игра.

|  |  |  |
| --- | --- | --- |
| Студент гр. 6382 |  | Камалеев Р.М. |
| Преподаватель |  | Филатов А.Ю. |

Санкт-Петербург

2018

**Цель работы.**

Изучение концепции ООП на примере реализации консольной игры. Получение навыков наследования классов.

**Постановка задачи.**

Дополнить проект, реализованный в первой лабораторной работе, следующим образом:

* Создать классы-наследники от класса Воин класс Мечник и класс Маг.
* Класс Мечник должен иметь

1. Функцию, переопределяющую функцию атаки так, чтобы был атакован объект с наименьшим количеством жизней на соседней клетке.

* Класс Маг должен иметь

1. Функцию, переопределяющую функцию атаки так, чтобы был атакован объект с наименьшим количеством жизней, а также все прочие объекты на линии атаки (луч атаки имеет ограниченную дальность)

* Класс Здание необходимо обновить, сделав его шаблонным, показывая воинов какого типа это здание производит.

**Пример работы программы.**

Входной файл:

10 10

7

m 5 0 1 80

s 5 2 121 80

m 0 4 1 80

m 9 4 1 80

s 5 5 172 80

m 5 9 1 80

b s 0 6 450 15 1 6 2 6 3 6 0 7 1 7 2 7 3 7 0 8 1 8 2 8 3 8 0 9 1 9 2 9 3 9

1

m 5 4 80 120

Результат работы:

Field:

x\_size = 10

y\_size = 10

Crown:

color = r

Crown.

Object:

x = 5

y = 0

hp = 1

crown = r

id = 0

Object.

Warrior:

strength = 80

Warrior.

Magician:

Magician.

Object:

x = 5

y = 2

hp = 121

crown = r

id = 1

Object.

Warrior:

strength = 80

Warrior.

Swordsman:

Swordsman.

Object:

x = 0

y = 4

hp = 1

crown = r

id = 2

Object.

Warrior:

strength = 80

Warrior.

Magician:

Magician.

Object:

x = 9

y = 4

hp = 1

crown = r

id = 3

Object.

Warrior:

strength = 80

Warrior.

Magician:

Magician.

Object:

x = 5

y = 5

hp = 172

crown = r

id = 4

Object.

Warrior:

strength = 80

Warrior.

Swordsman:

Swordsman.

Object:

x = 5

y = 9

hp = 1

crown = r

id = 5

Object.

Warrior:

strength = 80

Warrior.

Magician:

Magician.

Object:

x = 0

y = 6

hp = 450

crown = r

id = 6

Object.

Building:

x0 = 0

y0 = 6

x1 = 1

y1 = 6

x2 = 2

y2 = 6

x3 = 3

y3 = 6

x4 = 0

y4 = 7

x5 = 1

y5 = 7

x6 = 2

y6 = 7

x7 = 3

y7 = 7

x8 = 0

y8 = 8

x9 = 1

y9 = 8

x10 = 2

y10 = 8

x11 = 3

y11 = 8

x12 = 0

y12 = 9

x13 = 1

y13 = 9

x14 = 2

y14 = 9

Building.

Crown:

color = g

Crown.

Object:

x = 5

y = 4

hp = 80

crown = g

id = 7

Object.

Warrior:

strength = 120

Warrior.

Magician:

Magician.

Field.

0123456789

+----------+

0|.....m....|

1|..........|

2|.....s....|

3|..........|

4|m....m...m|

5|.....s....|

6|bbbb......|

7|bbbb......|

8|bbbb......|

9|bbbb.m....|

+----------+

Input army object you want to action (color, creature, x, y) [q to exit]:

**Основные теоретические положения.**

*Наследование* — это механизм создания нового класса на основе уже существующего. При этом к существующему классу могут быть добавлены новые элементы (данные и функции), либо существующие функции могут быть изменены. Основное назначение механизма наследования — повторное использование кодов, так как большинство используемых типов данных являются вариантами друг друга, и писать для каждого свой класс нецелесообразно.  
Объекты разных классов и сами классы могут находиться в отношении наследования, при котором формируется иерархия объектов, соответствующая заранее предусмотренной иерархии классов.

Иерархия классов позволяет определять новые классы на основе уже имеющихся. Имеющиеся классы обычно называют *базовыми* (иногда порождающими), а новые классы, формируемые на основе базовых, – *производными* (порожденными, классами-потомками или наследниками).

Производные классы «получают наследство» – данные и методы своих базовых классов, и могут пополняться собственными компонентами (данными и собственными методами). Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах. Сообщение, обработку которого не могут выполнить методы производного класса, автоматически передается в базовый класс. Если для обработки сообщения нужны данные, отсутствующие в производном классе, то их пытаются отыскать автоматически в базовом классе.

При наследовании некоторые имена методов (функций-членов) и данных-членов базового класса могут быть по-новому определены в производном классе. В этом случае соответствующие компоненты базового класса становятся недоступными из производного класса. Для доступа из производного класса к компонентам базового класса, имена которых повторно определены в производном, используется операция разрешения контекста ::.

Для порождения нового класса на основе существующего используется следующая общая форма

**сlass Имя: МодификаторДоступа  ИмяБазовогоКласса  
{ объявление\_членов;};**

При объявлении порождаемого класса **МодификаторДоступа** может принимать значения public, private, protected либо отсутствовать, по умолчанию используется значение private. В любом случае порожденный класс наследует все члены базового класса, но доступ имеет не ко всем. Ему доступны общие (public) члены базового класса и недоступны частные (private).

Для того, чтобы порожденный класс имел доступ к некоторым скрытым членам базового класса, в базовом классе их необходимо объявить со спецификацией доступа защищенные (protected). Члены класса с доступом protected видимы в пределах класса и в любом классе, порожденном из этого класса.

В качестве контейнеров, хранящих объекты, выступают двусвязные списки, используемые в предыдущей лабораторной работе.

Двусвязный список - это структура данных, которая состоит из узлов, которые хранят полезные данные, указатели на предыдущий узел и следующий узел.

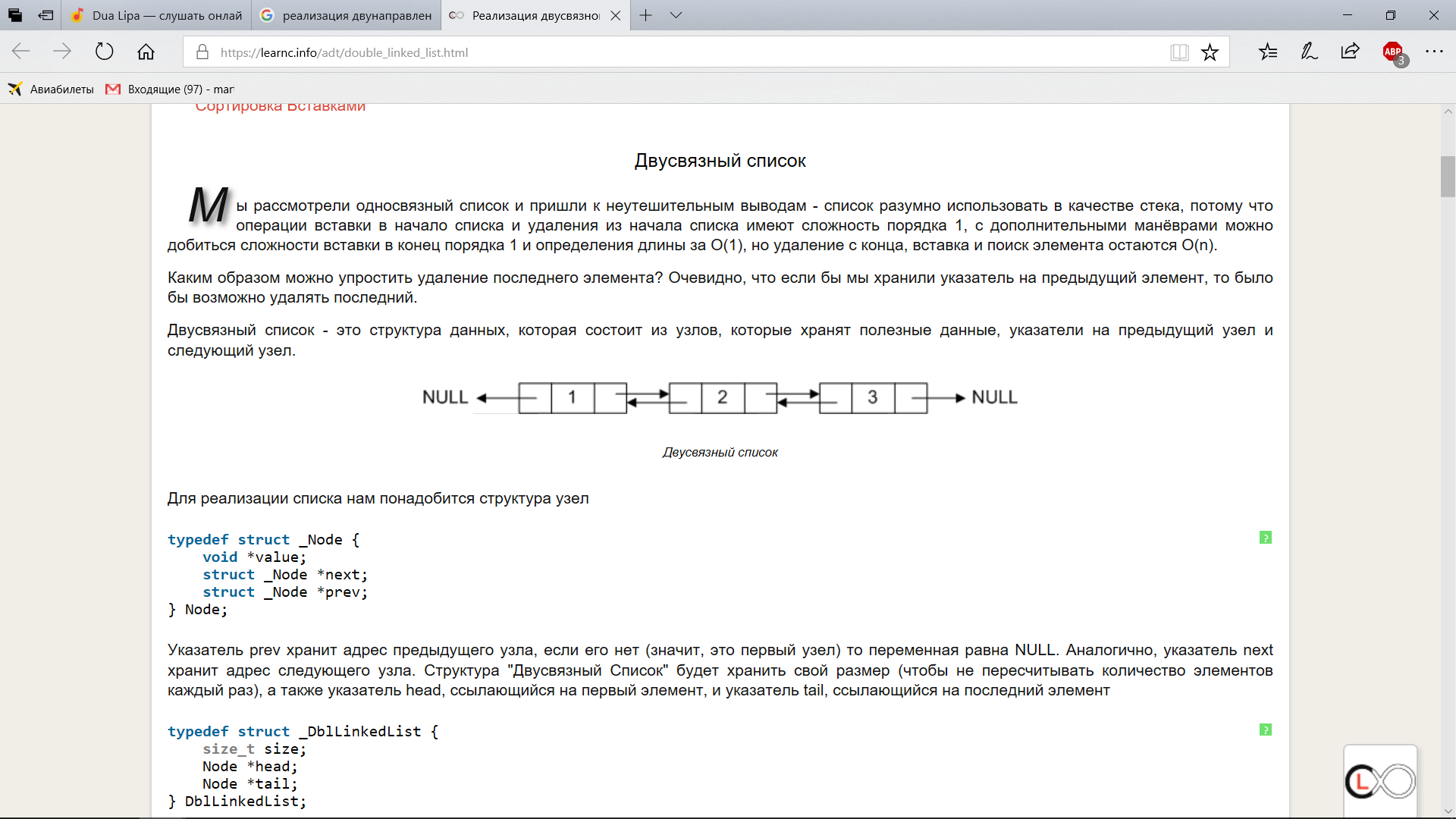


Рисунок 1. Двусвязный список.

**Спецификация программы.**

* Для класса Object определены следующие функции:

1. Говорящие конструктор Object(int x, int y, int hp, const shared\_ptr<Crown> crown) : x(x), y(y), hp(hp), crown(crown), id(count) и деструктор – и ~Object() соответственно.
2. Функция, инициализирующая поля объекта из файла - Object(ifstream& fl, const shared\_ptr<Crown>& clr) : id(count++), crown(clr).
3. Перегруженный оператор « - ostream& operator << (ostream& ostr, const Object& obj).
4. Перегруженный оператор » - istream& operator >> (istream &istr, Object &obj).
5. Функция, определяющая уровень здоровья объекта int& Get\_HP().
6. Функция, определяющая координату X объекта int& Get\_X().
7. Функция, определяющая координату Y объекта int& Get\_Y().
8. Функция, определяющая, есть ли объект на указанной позиции – bool Is\_Located(int x1, int y1).
9. Функция, принимающая урон – int Damage(int damage).
10. Функция, определяющая, убит ли объект – bool Is\_Dead().

* Для класса Warrior определены следующие функции:
  1. Говорящие конструктор и деструктор – Warrior(int x, int y, int hp, int strength, const shared\_ptr<Crown> crown) и ~Warrior() соответственно.
  2. Функция, инициализирующая поля война из файла Warrior(ifstream& fl,const shared\_ptr<Crown>& clr).
  3. Функции, отвечающие за передвижение война вверх, вниз, влево и вправо – void move\_up(), void move\_down(), void move\_left(), void move\_right() соответственно.
  4. Функция, определяющая силу война int& Get\_Strength().
  5. Функция, отвечающая за атаку объекта, совпадающего с воином координатами – List<tuple<int, int, int>>\* attack().
  6. Перегруженная функция базового класса Object, определяющая, есть ли воин на указанной позиции – bool Is\_Located(int x1, int y1).
* Для класса Building определены следующие функции:
  + 1. Говорящие конструктор и декструктор – Building(ifstream& fl, const shared\_ptr<Crown>& clr) и ~Building() соответственно.
    2. Перегруженная функция базового класса Object, определяющая, есть ли здание на указанной позиции – bool Is\_Located(int x1, int y1).
    3. Функция, создающая войнов армии здания – T\* create(int x, int y, int hp, int force, const shared\_ptr<Crown> crown).
    4. Функция, показывающая, воинов какого типа здание производит char Get\_C().
* Для класса Field определены следующие функции:

1. Говорящие конструктор и деструктор – Field(ifstream &fl) и ~Field() соответственно.
2. Функция, инициализирующая объекты-контейнеры красной армии из файла – Get\_Red\_Army(ifstream& fl, const shared\_ptr<Crown>& color).
3. Функция, инициализирующая объекты-контейнеры зелёной армии из файла – Get\_Green\_Army(ifstream& fl, const shared\_ptr<Crown>& color).
4. Функция, вызывающая меню перемещения для войнов – void Movement\_Menu().
5. Функция, отвечающая за перемещение воинов - void Movement(string clr, int entr, int x, int y).
6. Функция, отвечающая за атаку мечников и магов – bool Attack(string clr, string kind, int x1, int y1).
7. Функция, определяющая координаты война с минимальным количеством здоровья – tuple<int, int> Min\_Ray(string clr, List<tuple<int, int, int>>\* range).
8. Функция, определяющая здоровье война с минимальным количеством здоровья – Get\_Min\_Enemy\_HP(string clr, int x, int y).
9. Функция, выполняющая отрисовку поля боя – void Arena().
10. Функция, выполняющая создание объекта зданием, находящимся на позиции x и y – void Create\_Object(string clr, int x, int y).
11. Функция, определяющая, куда выполнено попадание (по объекту красной армии, зелёной армии или мимо) – bool Is\_Target\_Hit(string clr, int x1, int y1).
12. Перегруженный оператор « - ostream& operator << (ostream& ostr, const Object& obj).
13. Перегруженный оператор » - istream& operator >> (istream &istr, Object &obj).

* Для класса Swordsman определены следующие функции:

1. Говорящие конструктор и деструктор – Swordsman(int x, int y, int hp, int force, const shared\_ptr<Crown> crown) и ~Swordsman() соответственно.
2. Функция, инициализирующая поля мечника из файла – Swordsman(ifstream& fl, const shared\_ptr<Crown>& clr).
3. Функция, отвечающая за атаку соседнего объекта с наименьшим количеством жизни – List<tuple<int, int, int>>\* attack().
4. Перегруженная функция базового класса, определяющая, есть ли мечник на указанной позиции – bool Is\_Located(int x1, int y1).

* Для класса Magician определены следующие функции:

1. Говорящие конструктор и деструктор – Magician(int x, int y, int hp, int force, const shared\_ptr<Crown> crown) и ~Magician()соответственно.
2. Функция, инициализирующая поля мага из файла – Magician(ifstream& fl, const shared\_ptr<Crown>& clr, int x, int y).
3. Функция, отвечающая за атаку объекта с наименьшим количеством жизни на линии поражения, а также всех остальных объектов, попадающих в неё – List<tuple<int, int, int>>\* attack().
4. Перегруженная функция базового класса, определяющая, есть ли маг на указанной позиции – bool Is\_Located(int x1, int y1).
5. Функции, отвечающие за линии поражения мага: вверх, вниз, вправо и влево соответственно – List<tuple<int, int, int>>\* up\_distance(); List<tuple<int, int, int>>\* down\_distance(); List<tuple<int, int, int>>\* right\_distance(); List<tuple<int, int, int>>\* left\_distance().

* Для класса Crown определены следующие функции:

1. Говорящие конструктор и деструктор – Crown(bool flag) и ~Crown() соответственно.
2. Функция, задающая цвет короны – void Set\_Color(bool flag).
3. Функция, определяющая цвет короны – string Get\_Color(bool flag).

* Для класса Iterator определены следующие функции:

1. T& operator \*() - оператор разыменования.
2. T\* operator ->() - оператор доступа к полям по указателю.
3. T operator [](int index) - оператор доступа к полям по индексу.
4. Iterator operator ++() - оператор-инкремент.
5. Iterator operator --() - оператор-декремент.
6. friend bool operator !=(const Iterator& x, const Iterator& y) - оператор равенства.
7. friend bool operator ==(const Iterator& x, const Iterator& y) - оператор неравенства.

* Для класса List определены следующие функции:

1. ~List() – деструктор.
2. void Print() – выводит на экран список.
3. void Supl\_End(T info) – добавляет элемент в конец списка.
4. void Delete(const T& info) — удаляет элемент.
5. void Supl\_Begin(T info) – добавляет элемент в начало списка.
6. void Supl\_Pos(T info, int pos) – добавляет элемент по позиции.
7. void Erase\_End(T info) – удаляет элемент из конца списка.
8. void Erase\_Begin(T info) – удаляет элемент из начала списка.
9. void Erase\_Pos(T info, int pos) – удаляет элемент по позиции.

**Выводы.**

В ходе данной лабораторной работы были получены навыки работы с умными указателями, рассмотрены и изучены основные концепции объектно-ориентированного программирования (ООП), такие как наследование классов, дружественные классы, перегрузка операторов, шаблоны классов, вызов конструкторов и деструкторов. С использованием всех этих принципов была реализована консольная игра.