**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**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 класс Воин, который должен иметь:

1. Поле, указывающее силу удара.
2. Функцию перемещения вверх/вниз/вправо/влево.
3. Функцию атаки объектов, которые располагаются на тех же координатах, что и он сам.

* создать класс Здание, который должен иметь:

1. Набор координат, в которых это здание располагается.

* реализовать пошаговую игру до уничтожения одной командой другой.

**Пример работы программы.**

Входной файл:

10 10

4

w 6 9 150 60

w 2 5 50 50

w 5 5 10 150

b 1 7 450 9 2 7 3 7 1 8 2 8 3 8 1 9 2 9 3 9

4

w 1 0 80 120

w 2 2 100 100

w 8 1 120 76

b 5 2 300 4 6 2 5 3 6 3

Результат работы:

Field:

x\_size = 10

y\_size = 10

Crown:

color = red

Crown.

Object:

x = 6

y = 9

hp = 150

crown = red

id = 0

Object.

Warrior:

strength = 60

Warrior.

Object:

x = 2

y = 5

hp = 50

crown = red

id = 1

Object.

Warrior:

strength = 50

Warrior.

Object:

x = 5

y = 5

hp = 10

crown = red

id = 2

Object.

Warrior:

strength = 150

Warrior.

Object:

x = 1

y = 7

hp = 450

crown = red

id = 3

Object.

Building:

x0 = 1

y0 = 7

x1 = 2

y1 = 7

x2 = 3

y2 = 7

x3 = 1

y3 = 8

x4 = 2

y4 = 8

x5 = 3

y5 = 8

x6 = 1

y6 = 9

x7 = 2

y7 = 9

x8 = 3

y8 = 9

Building.

Crown:

color = green

Crown.

Object:

x = 1

y = 0

hp = 80

crown = green

id = 4

Object.

Warrior:

strength = 120

Warrior.

Object:

x = 2

y = 2

hp = 100

crown = green

id = 5

Object.

Warrior:

strength = 100

Warrior.

Object:

x = 8

y = 1

hp = 120

crown = green

id = 6

Object.

Warrior:

strength = 76

Warrior.

Object:

x = 5

y = 2

hp = 300

crown = green

id = 7

Object.

Building:

x0 = 5

y0 = 2

x1 = 6

y1 = 2

x2 = 5

y2 = 3

x3 = 6

y3 = 3

Building.

Field.

0123456789

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

0|.w........|

1|........w.|

2|..w..bb...|

3|.....bb...|

4|..........|

5|..w..w....|

6|..........|

7|.bbb......|

8|.bbb......|

9|.bbb..w...|

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

Input army object you want to action (color, 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.

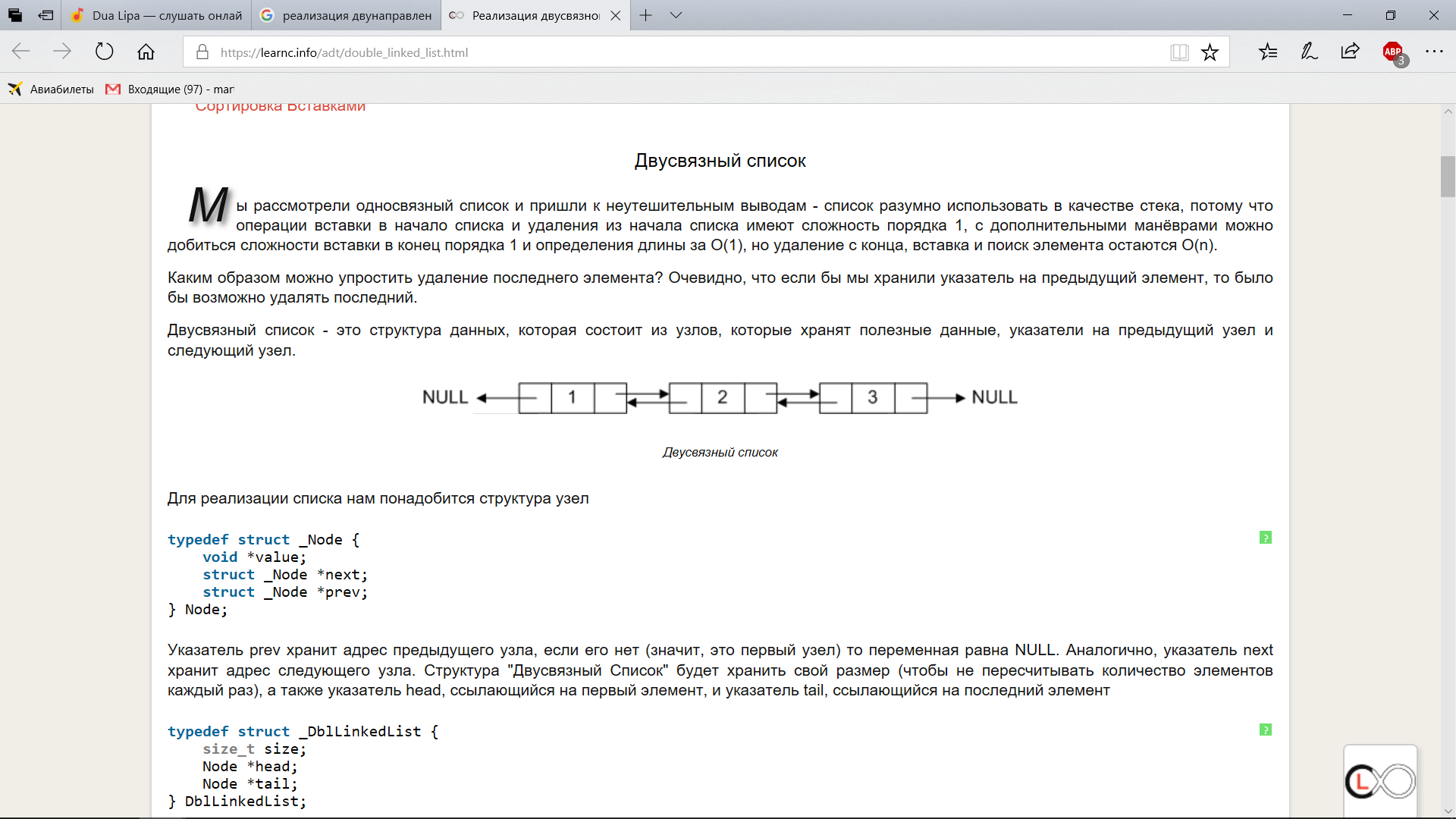


Рисунок 1. Двусвязный список.

**Спецификация программы.**

* Для класса Object определены следующие функции:

1. Говорящие конструктор деструктор – Object(ifstream& fl, const shared\_ptr<Crown>& clr) и ~Object() соответственно.
2. Функция, определяющая координату X объекта int& Get\_X();
3. Функция, определяющая координату Y объекта int& Get\_Y();
4. Функция, определяющая, есть ли объект на указанной позиции – bool Is\_Located(int x1, int y1).
5. Функция, принимающая урон – int Damage(int damage).
6. Функция, определяющая, убит ли объект – bool Is\_Dead().

* Для класса Warrior определены следующие функции:
  1. Говорящие конструктор и деструктор – Warrior(ifstream& fl,const shared\_ptr<Crown>& clr) и ~Warrior() соответственно.
  2. Функции, отвечающие за передвижение война вверх, вниз, влево и вправо - void move\_up(), void move\_down(), void move\_left(), void move\_right() соответственно.
  3. Функция, отвечающая за атаку объекта, совпадающего с воином координатами - List<tuple<int, int, int>>\* attack().
  4. 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).
* 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 воинов – bool Attack(string clr, int x1, int y1, int x2, int y2).
7. Функция, выполняющая отрисовку поля боя – void Arena().
8. Функция, определяющая, куда выполнено попадание (по объекту красной армии, зелёной армии или мимо) – bool Is\_Target\_Hit(string clr, int x1, int y1).

* 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.