**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 |  | Мартыненко П.П. |
| Преподаватель |  | Филатов А.Ю. |

Санкт-Петербург

2018

**Цель работы.**

Изучить концепции объектно-ориентированного программирования на примере реализации консольной игры.

**Формулировка задания.**

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.

**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 ::.

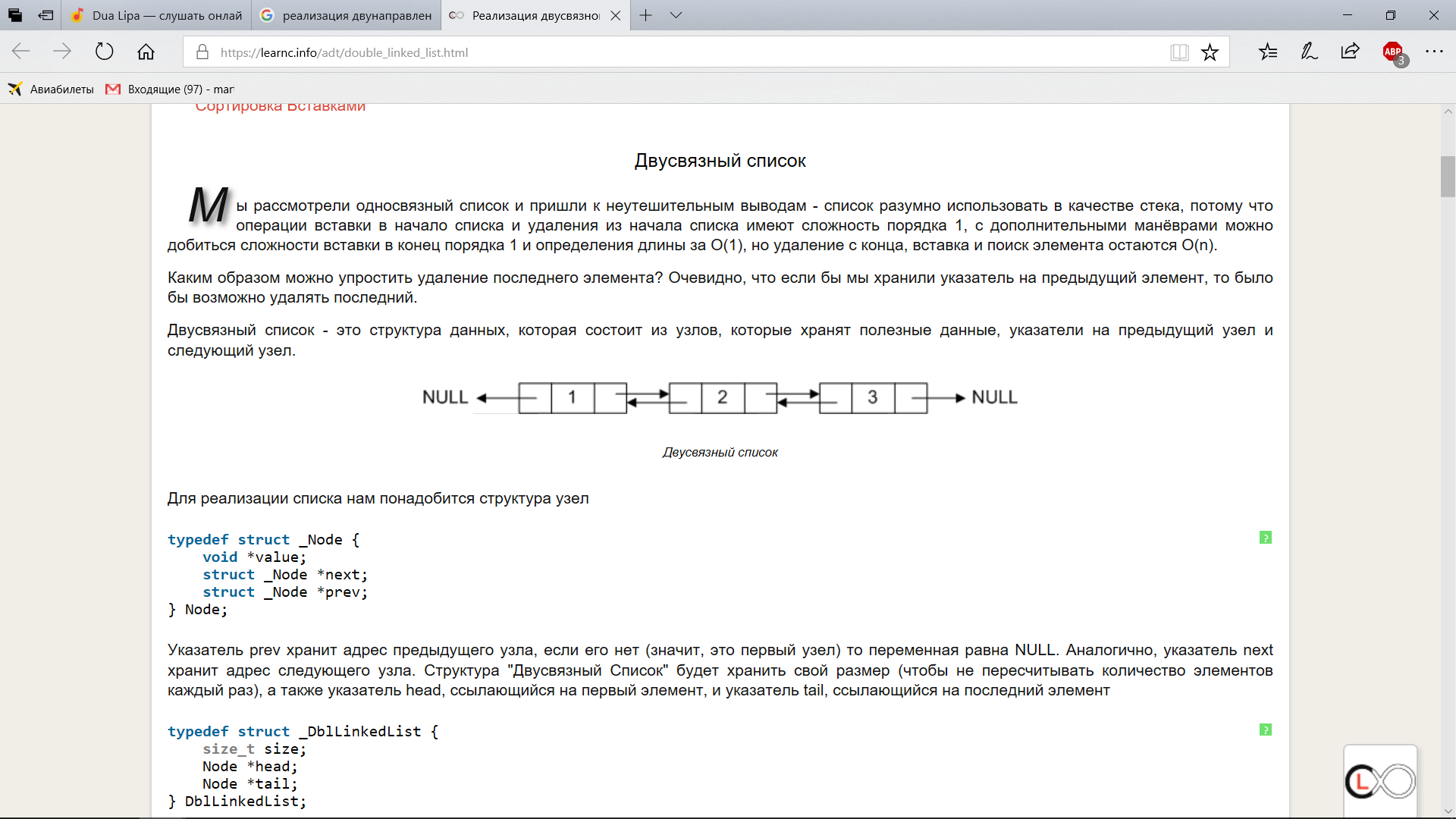
Для порождения нового класса на основе существующего используется следующая общая форма

**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.

**Спецификация программы.**

Для класса Object определены следующие операции:

1. Говорящие конструктор и деструктор – Object(int x, int y, int hp, const shared\_ptr<Crown> crown) и virtual ~Object() соответственно.
2. Функция, которая принимает урон – int damage(int dmg).
3. Функция, отвечающая на вопрос, есть ли Объект на указанной позиции – virtual bool isObject(int x1, int y1).
4. Считывание из файла – Object(ifstream& f, const shared\_ptr<Crown>& clr);
5. Функция, отвечающая на вопрос, «убит» ли объект – bool isDead().
6. Перегруженные операторы >> и <<, производящие ввод и вывод из потока ввода/вывода соответственно.

Для класса Warrior определены следующие операции:

1. Говорящие конструктор и деструктор – Warrior(int x, int y, int hp, const shared\_ptr<Crown> crown) и ~Warrior() соответственно.
2. Функции передвижения вверх, вниз, вправо, влево – void move\_up(), void move\_down(), void move\_right() и void move\_left() соответственно.
3. Функция атаки противника, находящегося на той же позиции – List<tuple<int,int,int>>\* attack();
4. Перегруженная функция базового класса Object, отвечающая на вопрос, если объект класса Воин в заданной позиции, – int isObject(int x1, int y1).

Для класса Building определены следующие операции:

1. Говорящие конструктор и деструктор – Building(int x, int y, int hp, const shared\_ptr<Crown> crown) и ~Building() соответственно.
2. Перегруженная функция базового класса Object, отвечающая на вопрос, если объект класса Здание в заданной позиции, – int isObject(int x1, int y1).

Для класса Field определены следующие операции:

1. Говорящие конструктор и деструктор – Field(ifstream &f) и ~Field() соответственно.
2. Функция, атакующая объект на заданной позиции – void attack(int x,int y,int dmg).
3. Отрисовка поля боя на экране (разными цветами) – void paint().
4. Загружать объекты из файла – void getA(ifstream& f, const shared\_ptr<Crown>& clr) и void getB(ifstream& f, const shared\_ptr<Crown>& clr).
5. Определять, располагается ли на заданных координатах объект той или иной армии – bool isIt(int x1, int y1).
6. Перегруженные операторы >> и <<, производящие ввод и вывод из потока ввода/вывода соответственно.

Для класса Iterator определены следующие операции:

* Обращение к первому элементу списка. Возвращает класс Iterator, указывающий на первый элемент – метод begin().
* Обращение к «последнему» элементу списка. Возвращает класс Iterator, указывающий на нулевой элемент – метод end(). Используется для обозначения конца списка.
* Обращение к последнему элементу списка. Возвращает класс Iterator, указывающий на последний элемент – метод last().
* Разыменовывание. Возвращает ссылку на элемент, хранящийся в узле – **operator \***,
* Инкремент. Переход к следующему элементу (берет из узла и сохраняет адрес следующего узла) – **operator ++**,
* Декремент. Переход к предыдущему элементу (берет из узла и сохраняет адрес предыдущего узла) – **operator --**,
* Равенство и неравенство (простое сравнение указателей) – **operator ==** & **operator !=** ,
* Переход к узлу, на который указывает Iterator– **operator ->.**

Для класса List определены следующие операции:

* Вставка в начало и конец списка, а также вставка по номеру – методы pushFront(…), pushBack(…) и Insert(…) соответственно.
* Удаление из начала и конца списка, а также удаление по номеру – методы popFront(…), popBack(…) и Delete(…) соответственно.
* Показ списка – метод show().

**Выводы.**

В данной лабораторной работе были рассмотрены и изучены основные концепции объектно-ориентированного программирования, такие как наследование классов, дружественные классы, перегрузка операторов, шаблоны классов и т.д. Также был реализована консольная игра в рамках ООП.