Московский государственный технический университет им. Н.Э.Баумана

Большаков С.А. , к.т.н., доц. Каф. ИУ5

Методические указания для выполнения ДЗ/КЛР по дисциплине Программирование на основе классов и шаблонов кафедры ИУ5

Москва - 2016

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ

1. Введение

Данные методические указания предназначены для студентов кафедры ИУ5 обучающихся во 2-м семестре и изучающих дисциплину “Программирование на основе классов и шаблонов”. Для выполнения комплексной лабораторной работы (КЛР) или, подругому, домашнего задания (ДЗ) по курсу необходим определенный набор базовых знаний и умений, но, к сожалению, не все студенты ими обладают. Я надеюсь, что для большинства студентов в этот материал не будет содержать много новых сведений по курсу: основные знания они должны быть получены на лекциях, при выполнении лабораторных работ и при самостоятельной проработке литературы. С другой стороны я надеюсь, что для части студентов данная информация будет, несомненно, полезна и они с успехом будут использовать данный материал для выполнения и защиты лабораторных работ и домашнего задания по дисциплине.

В данных методических указаниях рассмотрены основные теоретические вопросы, необходимые студентам для выполнения задания (это не весь курс!), шаги выполнения задания, рассмотрен пример выполнения задания и требования к заданию. Кроме этого, рассмотрены требования к документации для выполнения задания и даны примеры ее оформления.

2. Цель и задачи КЛР/ДЗ

Целями КЛР (домашнего задания) **является**: освоение технологии объектно-ориентированного проектирования программного обеспечения; самостоятельная разработка системы классов; получение навыков работы с системами программирования, ориентированными на объектный подход, в частности с системой программирования на языке C++. Получение навыков разработки и оформления технической документации для программ ориентированных на объекты. Домашнее задание выполняется студентами в учебные часы самостоятельной работы (17 академических часов) и цикла специальных лабораторных работ (ЛР12 – ЛР15). Методические указания по ЛР КЛР размещены на сайте дисциплины.

Темы заданий

Каждый студент выполняет КЛР (домашнее задание) **индивидуально** (варианты тем заданий представлены в отдельном файле). Кроме этого для каждой из групп предъявляются специальные требования. Студенту назначается тема задания в соответствии с вариантом по журналу группы. Сильные студенты могут выполнить домашние задания с учетом дополнительных требований. Варианты заданий студентов и варианты групп представлены на сайте в отдельном файле. По договоренности с преподавателем вариант может быть изменен на любой, соответствующий тематике задания, но не входящий в список вариантов групп (утвердить новую тему можно не позднее 4-й недели текущего семестра). Эту тему можно выбрать из списка тем с дополнительными требованиями или придумать самому. Степень сложности таких новых тем должен соответствовать степени сложности тем с дополнительными требованиями.

В каждом задании разрабатывается минимум один контейнерный класс и один элементный класс (см. ниже п.3).

3. Классы, наследование и перегрузка операций

В объектно-ориентированном программировании центральное место занимает понятие класса. Классы служат своеобразным эталоном (шаблоном) для создания объектов. Объекты являются основными конструктивными элементами (“строительными блоками”), из которых строиться программа.

3.1. КЛАССЫ. Класс и его составляющие

Класс это новый тип данных, который объявлен пользователем. Класс описывает новое понятие предметной области для программирования задач на основе объектов. Класс включает описания переменных класса (“членов-данных класса”) и методов класса (“членов функций класса”). Члены-данные класса копируются в каждом новом объекте и составляют его сущность. Методы класса (функции) описаны один раз и служат для работы с членами-данными класса. Иногда говорят, что методы класса определяют поведение объектов.

Формально описание класса имеет вид (здесь приводиться упрощенное описание):

class <имя класса >: <список базовых классов> {
< тело класса>
};

Таким образом, описание класса состоит из заголовка описания и тела класса. Более детально формальное описание класса выглядит так:

class <имя класса > <список базовых классов> {
public:
< описания членов класса>
…
private:
< описания членов класса>
…
protected:
< описания членов класса>
…
< повторение предыдущих описаний в любом порядке>
};

Под членами класса понимаются члены-данные класса и члены-функции класса. Список базовых классов строиться на основе следующего формального правила:

<список базовых классов> ::= <пусто> | :<[private | public | protected] базовый класс> … [, <[private | public | protected] базовый класс>] …

3.2. Члены-данные класса

Члены-данные класса(свойства объектов класса) – это любые описание переменных за исключением описания переменных класса, описываемого в данном случае (однако, можно описывать указатели и ссылки на типы этого класса). Например, класс, содержащий только данные:

```
class Student {
public:
 char Name[14]; // Фамилия студента
 int kurs; // Курс обучения
```

```
 bool pol; // true - women, false – men
private:
 float Stipen; // Размер стипендии
};
```

3.3. Методы класса

Члены-функции класса это любые функции, которые получают доступ непосредственно к членам-данным класса. В теле класса могут размещаться описания функций класса и их прототипов. Во втором случае производится внешнее описание метода с указанием того, к какому классу это описание принадлежит. Пример описания методов класса:

```
class Student {
public:
        char Name[14]; // Фамилия студента
int GetKurs();
void Print(){ // Метод описан в теле класса
cout<< "Фамилия: " << Name <<endl;
cout<< "Курс: " << kurs <<endl; };
private:
int kurs; // Курс обучения
...
        };
...
// Внешнее описание метода класса
int Student::GetKurs(){ return kurs;};
```

Описания методов может быть произведено и в другом исходном модуле, но данные модули должны быть подключены в проект и исходный модуль с описанием классов.

3.4. Описание объектов класса

После описания классов мы можем производить описания объектов данного класса наравне с описаниями стандартных типов. Объект - это переменная данного типа класса (экземпляр класса). Можно описывать массивы объектов, указатели и ссылки на объекты. Примеры описания объектов и массивов:

```
Student Petrov; // Описание объекта типа Student
Student Group22[30];  // Описание массива объектов типа Student в 30 элементов
Student * pStud; // Описание указателя на тип Student
Student & rStud = Petrov; // Описание ссылки на тип Student и инициализация
...
pStud = new Student; // Динамическое создание объекта типа Student
```

3.5. Конструкторы объектов

Для создания объектов класса используются специальные методы, которые называются конструкторами. Эти методы используются для вычисления начальных значений членов данных объекта, выделения дополнительной динамической памяти под данные объекта и других действий. Все конструкторы возвращают по-умолчанию тип ссылка на объект (Student &), поэтому в их описаниях и прототипах тип возврата не задается. Название функции конструктора совпадает с именем класса. Конструкторов может быть много, но они должны отличаться числом и типами параметров (перегрузка функций). По умолчанию в каждом класса существует конструктор без параметров. Он не содержит операторов. Если такой конструктор описан явно, то используется пользовательский конструктор. Примеры описания конструкторов:

```
class Student {
public:
...
Student(){ kurs = 1; Stipen = 1500.0f; pol = true;};
Student(int k, float s){ kurs = k; Stipen = s; };
Student(int k, float s, bool p){ kurs = k; Stipen = s; pol = p;};
…};
```

Конструкторы целесообразно размещать в области public, так как только в этом случае доступно внешнее описание объектов. Примеры описаний с параметрами:

```
Student Petrov; // Описание объекта типа Student без параметров
Student Ivaniv(2 , 2000.0f); // Описание объекта типа Student с двумя параметрами
Student Sidorov(5, 3000.0f, true); // Описание объекта типа Student с тремя параметрами
...
Student  * pS = new Student (5, 3000.0f, true); // Динамический вызов конструктора
```

При каждом описании объекта конструкторы вызываются до его первого использования.

3.6. Деструкторы объектов

При удалении объектов также могут быть использованы специальные методы класса. Они называются деструкторами. В деструкторе параметров не задается, поэтому можно объявить только один деструктор. По умолчанию предусмотрен деструктор, который удаляет объект и используется в том случае, когда явно деструктор не объявлен. В деструкторе может освобождаться память, выделенная динамически, или производиться другие действия, связанные с удалением данного объекта (например, изменяться счетчики объектов данного типа). Название функции деструктора также совпадает с именем класса, но ему предшествует специальный знак тильда (“~”). Пример описания деструктора:

```
class Student {
public:
...
~Student(){ GlobalCount--;};
…};
```

Если объект создан динамически, то выполняется вызов деструктора явно при выполнении операции **delete**. Пример:

```
Student  * pS = new Student (5, 3000.0f, true); // Динамический вызов конструктора
...
delete pS;
```

В других случаях (при явном описании объекта), деструкторы объектов вызываются при выходе из блока (составного оператора) в котором эти объекты описаны. Можно считать, что они вызываются при прохождении закрывающей скобки блока (“}”). Деструкторы глобальных объектов вызываются после завершения функции **main**, а конструкторы вызываются до ее начала!

3.7. Работа с данными и методами объектов вне класса

Доступ к данным класса в методах класса выполняется непосредственно, то есть мы можем использовать имена переменных без дополнительной ссылки на объект. Вне класса для доступа нужно использовать имя объекта (либо ссылка) или указатель на объект. Рассмотрим примеры для класса Student, описанного выше:

```
Student Sidorov(5, 3000.0f, true); // Описание объекта типа Student с тремя параметрами
Student  * pS = new Student (5, 3000.0f, true); // Указатель на динамический объект
Student  & rS = *new Student (5, 3000.0f, true); // Ссылка на динамический объект
Student Group22[30];  // Описание массива объектов типа Student в 30 элементов
```

```
...
cout << "Name:  " << Sidorov.Name << endl; //Квалицированная ссылка
cout << "Name:  " << pS->Name << endl;  // Указатель
cout << "Name:  " << rS.Name << endl;  // Динамическая ссылка
cout << "Name:  " << Group22[5].Name << endl; // Элемент массива 5
cout << "Kurs:  " << Sidorov.kurs << endl; // !!! Ошибка, т.к. это поле объекта private
```

Для вызова методов класса они должны быть в области объекта **public**:

```
cout << "Kurs:  " << Sidorov.GetKurs() << endl; // Объект
cout << "Kurs:  " << pS->GetKurs() << endl; // Указатель
cout << "Kurs:  " << rS.GetKurs() << endl; //  Ссылка
cout << "Name:  " << Group22[5]. GetKurs()<< endl; // Элемент массива 5
Sidorov.Print(); // Вызов метода печати
```

3.8. ИНКАПСУЛЯЦИЯ

Для эффективного программирования и надежности программы в классах определяются различные режимы доступа (по-другому, области видимости) к данным и функциям. Если данные и методы описаны в области видимости **private**, то они недоступны (невидимы) для внешнего использования вне класса. С такими данными могут работать только методы класса. Если данные описаны в области **public**, то они доступны для доступа вне класса. Инкапсуляцией называется технология скрытия данных и методов от внешнего пользователя, Существует еще один режим доступа **protected**, но мы поясним его в разделе наследования. Примеры открытого доступа и ошибки инкапсуляции были показаны выше для класса Student и объектов данного класса. В частности доступ к члену данных **kurs** запрещен, так как он расположен в области **private**.

3.9. НАСЛЕДОВАНИЕ

Наследованием называется создание новых типов данных (классов) на основе существующих классов. Исходные классы для наследования должны быть предварительно описаны или определены. При наследовании различают понятия базового и порожденного класса. Базовым классом является тот, на основе которого создается другой класс. Никаких ограничений на базовый класс не накладывается кроме: базовый класс должен быть предварительно определен и имя базового класса не должно совпадать с именем порожденным. Порожденным считается новый класс. Порождение (или наследование) может быть 2-х видов: **public** и **private**. Этот спецификатор доступа должен предшествовать имени базового класса. Формально

<список базовых классов>::= [public | private | protected] <имя базового класса>

[,<имя базового класса> …]

При наследовании классов типа **public** в порожденном классе доступны все члены базового класса из разделов **public** и **protected**. При наследовании типа **private** в порожденном классе не доступны все члены базового класса. При наследовании типа **protected** в порожденном классе все открытые и защищенные члены базового класса становятся защищенными в классе наследнике. Простое наследование – это наследование от одного базового класса. Если базовых классов несколько, но наследование называется множественным. Отметим, что в объект порожденного класса всегда включены все члены базового класса, хотя они могут быть невидимы (недоступны).

При наследовании в порожденном классе могут быть:

- Добавлены новые члены данные.
- Добавлены новые методы.
- Заново описаны данные с тем же именем.
- Заново описаны методы с тем же именем.

– Изменены методы доступа к данным и методам.

Главное это то, что в порожденном классе создается новый тип данных с новыми свойствами и новым поведением (фактически новое понятие предметной области). Пример наследования:

```
class Magistr : public Student {
...
public:
char TemaDiss[MAX]; // Тема магистерской работы
...
viod SetNewTema(char * pNewTema); // Задание новой темы
...
};
```

В данном примере мы добавили новый член - данных – тема работы и метод изменения темы работы – новый метод. Отметим, что порожденном классе конструкторы порожденного должны вызывать конструкторы базового класса, если в них передаются параметры. Например:

```
...
Magistr (int k, float s , char * t) :Student(k, s){ strcpy(TemaDiss, t); };
...
```

Вызов конструкторов базовых классов выполняется с помощью фактических параметров передаваемых в конструктор порожденного класса. Более детально с вопросами наследования вы познакомитесь на лекциях и можете самостоятельно изучить в литературе [1]. Здесь затронуты только главные вопросы и понятия необходимые для выполнения ЛР.

3.10. Статические члены класса

В классе могут быть объявлены члены со специальным спецификатором **static**. Такие члены называются статическими. Фактически они являются общими для всех создаваемых объектов данного класса и могут рассматриваться как глобальные переменные программы. Обращение к таким статическим членам вне класса выполнятся так:

<имя класса>::<имя статической переменной>

Например, описана статическая переменная в качестве счетчика объектов класса:

```
Class Student {
 ...
Static int CountObj; // Статическая переменная счетчик
...
Student(){CountObj++ ;}; // Конструктор
~Student(){CountObj-- ;}; // Деструктор
...
};
```

Статические переменные должны быть обязательно проинициализированы (задано начальное значение). Это выполняется в программе после описания класса следующим образом:

```
int Student::CountObj = 0; // Инициализация статического чледа класса
...
Student::CountObj = 10; // Доступ к статической переменной вне класса как глобальной
```

3.11. Объявления класса

Для описания указателей и ссылок внутри классов на текущий описываемый класс или на классы, которые в данный момент компиляции еще не известны, используется объявление класса (не надо путать его с описанием или определением класса). С помощью такого объявления компилятору предписывается, что такой класс будет описан ниже. Объявление задается в следующей форме:

class <имя класса> ;

Например в новом классе A используется описание указателя на объект типа Student:

```
class Student;
class A {
...
Student * pS;
...
};
```

Для выполнения наследования (использования в виде базового класса) можно применять повторное описание класса базового класса в другом модуле.

3.12. Классы и структуры

В языке C++ понятие структуры данных расширено до понятия класса. В структурах могут определяться методы, условия защиты данных и так далее. Основное отличие структур данных от классов заключается в том, что в них по-умолчанию все члены является открытыми (**public**). В классах наоборот, по-умолчанию, когда не указан режим доступа, все члены класса считаются закрытыми (**private**).

3.13. Операции в классах

В классах могут быть перегружены стандартные операции (“+”, “-” и т.д.). Понятие перегрузки операций заключается в том, что для объектов нового тапа задается новый смысл операции. Этот смысл (или действия при выполнении операции) определяется специальной функцией, написанной пользователем. Более подробно механизм перегрузки операций будет изложен в методических указаниях к лабораторной работе по перегрузке. Например, если мы имеем объект типа группа студентов, то мы можем сложить две группы:

```
Group G1;
Group G2;
Group G3;
G3 = G1 + G2;
```

3.14. Статическое и динамическое связывание

Как сделать программу более наглядной, универсальной, понятной и компактной одновременно? Этот вопрос очень важен при построении программных систем. При построении сложных программных систем применяются разные методы и механизмы для обеспечения такого результата. Наглядность и компактность, а также универсальность, способствуют сокращению сроков разработки и отладки программного обеспечения (ПО), в значительной мере влияют на его качество (надежность ПО, сопровождение ПО, мобильность ПО) и обеспечивают увеличение сроков службы ПО. В принципе возможны два подхода достижения этих результатов: статическое и динамическое связывание.

Связывание в программировании означает внутреннюю настройку программы на классы, функции и переменные. Способ связывания существенно влияет на универсальность программы. Они отличаются временем связывания. Другими словами, это определение того, на каком этапе формирования и выполнения программы, эти связи устанавливаются. Принято рассматривать три возможных этапа: этап работы макропроцессора (предварительной обработки исходного текста), этап компиляции и этап выполнения программ. Первые два этапа, обычно, называют статическим связыванием (связи можно изменять только до компиляции). Третий способ обеспечивает динамическое связывание,

такое связывание обеспечивается на этапе выполнения программы. Хотя последний способ является более сложным в реализации, он обеспечивает большую степень универсальности и компактности программных систем. Простым примером динамического связывания является использование указателей и динамической памяти в программах, а более сложным использование виртуальных функций в классах.

Данная ЛР посвящена изучению механизмов статического связывания: перегрузки функций и операций, параметров по умолчанию т.д.

3.15. Параметры по-умолчанию

В современных языках программирования допускается задание параметров функций по-умолчанию. Такая возможность обеспечивает использование вызова конкретной функции без задания всего перечня фактических параметров функции. Если параметр не задается, то в этом случае используется значение параметра по-умолчанию, которое может быть задано либо в прототипе функции, либо в описании функции (определении функции). Одновременно и в прототипе функции и в ее описании нельзя задавать параметры по-умолчанию.

Формат задания параметров по-умолчанию в прототипе функции:

<имя функции> (<тип> [=<значение>], <тип> [=<значение>], …);

Или возможно так:

<имя функции> (<тип> <тег>[=<значение>], <тип> <тег> [=<значение>], …);

Нетерминальная переменная **<тег>** - это любое имя, фактически комментарий, его обозначение может не совпадать с именами формальных параметров из описания функций. Значение - это переменная или выражение этапа компиляции (**#define**), либо константа, либо переменная доступная на этом этапе компиляции перед вызовом функции (глобальная переменная, объявленная в этом или другом модуле). При использовании в качестве значений глобальных переменных их можно и нужно вычислить заново перед вызовом функции. Суть использования параметра по-умолчанию следующий: если фактический параметр не задан в строке вызова (пропущен), то он заменяется значением этого параметра по-умолчанию. В этом случае, не сокращается число параметров функции, а просто подставляются стандартные значения. Пример использования параметров по-умолчанию в прототипах дан ниже:

```
int  Summa1( int a = 10 , int b = 20 , int c = 30); // три параметра по-умолчанию
int  Summa2( int a  , int b = 20 , int c = 30);  // два параметра по-умолчанию
int  Summa3( int  , int  , int c = 30); // один параметр по-умолчанию
```

Если задано три параметра по-умолчанию, то возможны следующие вызовы данных функций:

```
cout << "Сумма = "<< dec << Summa1 ( ) <<endl;
cout << "Сумма = "<< Summa1 ( 3 ) <<endl;
cout << "Сумма = "<< Summa1 (3 , 3) <<endl;
cout << "Сумма = "<< Summa1 (3 , 3 , 3 ) <<endl;
```

Если данный позиционный параметр не может быть задан по-умолчанию, пропуск его при вызове функций является ошибкой:

```
cout << "Сумма = "<< Summa2 ( 3 ) <<endl; // Правильно, только 1-й параметр
cout << "Сумма = "<< Summa3 ( 3 ) <<endl; // Ошибка второго параметра
```

Параметры по умолчанию можно пропускать только последовательно справа налево. Такое задание в примере ниже тоже является ошибкой:

```
cout << "Сумма = "<< dec << Summa1 (, 3, 5 ) <<endl; //Ошибка!
```

При задании параметров по умолчанию в описании функции (определении функции) синтаксис записи выглядит так:

<имя функции> (<тип> <ФП>[=<значение>], <тип> <ФП> [=<значение>], …) { …};

Здесь терминальная переменная **<ФП>** - это имя формального параметра, его упускать нельзя. Остальные правила задания параметров по-умолчанию сходны. Пример:

```
int  Summa( int a = 10 , int b = 20 , int c = 30) // Три параметра по умолчанию.
```

```
{
 return ( a + b + c ); };
```

Для такой функции также допустимы следующие четыре вызова, в зависимости от числа параметров взятых по-умолчанию:

```
cout << "Сумма = "<< dec << Summa1 ( ) <<endl;
cout << "Сумма = "<< Summa( 3 ) <<endl;
cout << "Сумма = "<< Summa(3 , 3) <<endl;
cout << "Сумма = "<< Summa(3 , 3 , 3 ) <<endl;
```

Рассмотрим использование глобальных переменных для задания начальных значений:

```
int iGlob = 5; // Глобальная переменная
// ...
int  Summa( int a = 10 , int b = 20 , int c = iGlob)
{  return ( a + b + c ); };
```

Вызов функции с предварительным изменением глобальной переменной:

```
cout << "Сумма = "<< Summa1 (3 , 3) <<endl;
iGlob = 15;
cout << "Сумма = "<< Summa(3 , 3) <<endl; // Третий фактический параметр  = 15
```

Глобальная переменная должна быть описана перед описанием прототипа или определения функции, то есть находиться в области видимости.

3.16. Перегрузка функций (Статическое связывание)

Перегрузка функций заключается в возможности вызывать функции с одинаковыми именами, но для разных типов параметров, точнее функций с разной сигнатурой. Уменьшение числа разнообразных имен для разных типов объектов значительно увеличивает степень понятности, читаемости и наглядности программ. Сравните варианты названий для функции перемещения разных объектов (**Move** – при перегрузке можно использовать одно имя): **MoveRect**, **MoveLine**, **MovePoint** и т.д. Число таких названий функций в программе может быть большим. При использовании механизма перегрузки мы можем воспользоваться одним именем **Move** для объектов разных типов, в этом случае не надо помнить различные имена!

Поясню, что понимается под сигнатурой функции. Это понятие не просто определить одним предложением. В сигнатуру функции входит:

- Название (имя) функции,
- Определения типов параметров.
- Число параметров.
- Тип данных возврата функции.
- Вариант последовательности (порядка) типов параметров.

Функции, которые имеют разные имена, имеют разную сигнатуру. Функции, которые имеют разное число фактических параметров, будут иметь разную сигнатуру. Функции, в которых параметры по типам и номерам не совпадают, имеют разную сигнатуру. Правило перегрузки функций звучит так: **если функции имеют разную сигнатуру, исключая имя функции, то их можно описывать и использовать повторно**. Другими словами, можно использовать функции с одинаковыми именами для разных совокупностей формальных параметров. Рассмотрим пример для функции Swap, применительно к ссылкам и указателям. Описание функций:

```
void Swap(int * x , int * y) // Указатели
{ int Temp = *x;
*x=*y;
*y=Temp;};
//
void Swap(int & x , int & y) // ссылки
{ int Temp = x;
x=y;
```

```
y=Temp;};
```
При вызове функции с параметрами указателями, будет вызвана 1-я функция:
```
int x = 5, y = 10;
Swap (&x , &y);
```
При вызове функции с параметрами ссылками, будет вызвана 2-я функция:
```
Swap (x , y);
```
Если в программе есть описание другой функции Swap с другими типами, например типом float для ссылок:
```
void Swap(float & x , float & y) // ссылки
{ float Temp = x;
x=y;
y=Temp;};
```
То ее можно вызвать, задавая параметры других типов (float):
```
float x = 5, y = 10;
Swap (x , y);
```
На этапе компиляции (статическое связывание) компилятором определяется описание функции, которое будет использоваться в каждом конкретном случае. Вспомним из предыдущих ЛР, что для этого случая целесообразнее использовать шаблоны функции, когда число параметров совпадает (Напомним, что шаблон функции позволяет компилятору фактически сгенерировать описание перегруженных функций и одинаковыми алгоритмами). Если число параметров не совпадает, то перегрузка функций незаменима, например:
```
int  Summa( int a  , int b  , int c ) // Три параметра
{
 return ( a + b + c ); };
int  Summa( int a  , int b   ) // Два параметра
{
 return ( a + b  ); };
```
Тогда при вызове описанных выше функций таким образом:
```
// Число параметров при перегрузке
cout << "Сумма = "<< Summ (3 , 3 , 3 ) <<endl;  // Вызов первой функции
cout << "Сумма = "<< Summ (3 , 3) <<endl;  // Вызов второй функции
```
Примечание. В некоторых случаях при одновременном использовании механизма перегрузки функций и задания параметров по-умолчанию может возникнуть неоднозначность, что в данном случае применяется. Обычно компилятор выдает диагностическое сообщение, но все-таки нужно быть внимательными и стараться не допускать такой противоречивой трактовки проекта.

Напомню, что при описании классов может быть задано произвольное число конструкторов (см. ЛР по классам), они также должны обязательно отличаться сигнатурой и являются перегруженными функциями-членами класса.

3.17. ПЕРЕГРУЗКА. Суть механизма перегрузки операций

Механизм перегрузки операций в языках ООП позволяет задать для новых типов объектов новый смысл (порядок, алгоритм) выполнения стандартных операций. Например, для объектов типа строка можно записать:
```
String S1("Первая ");
String S2("Вторая ");
String S3();
S3 = S1 + S2;
```
Такая запись является наглядной и компактной (сравните с вариантом вызова функции RTL - strcat). Алгоритмы выполнения перегруженных операций могут быть любыми, так как они реализуются через функцию, которую описывает сам программист. Можно придумать самые разнообразные операции (даже фантастические) и реализовать в своей программе. Например, сложение двух улиц, домов и т.д (см. пособие[6]). Для пото-

кового ввода/вывода, например, перегружены стандартные операции двоичного сдвига: “**>>**” и “**<<**”.

Повторю в итоге еще раз: перегрузка операций – **это задание новых способов выполнения операций для новых типов объектов**.

3.18. Ограничения перегрузки операций

В языке C++ не все стандартные операции можно перегружать, есть ограничения. Это обуславливается особенностями синтаксиса языка и конструкцией компилятора. Так запрещено перегружать следующие операции:

- Операцию разрешения области видимости – “**::**”
- Условную операцию – “**() ? :** ”
- Операцию доступа к членам класса через объект и указатель на член класса– “***.**”
- Операцию разрешения квалифицированной ссылки “**.**”

Все остальные операции, включая и операцию индексирования (“[]”) можно перегружать. Использование приемов перегрузки операций позволяет сделать программный проект на основе классов более наглядным и качественным.

Для задания типа операции может использоваться один символ (знак) или два символа (знака). Так для бинарной операции сложения – один знак “+”, а для операции инкремента – два знака “++”. При перегрузке операций нужно это учитывать. Перегрузка операций выполняется в классах в виде функций класса или функций дружественных данному классу. Перегрузка операций для стандартных типов переменных не допускается. Рассмотрим разновидности перегрузки операций.

3.19. Разновидности типов перегрузки операций

Перегрузка операций имеет несколько разновидностей. В зависимости от числа операндов (значений, участвующих в операции), они подразделяются на одноместные и двуместные. Одноместная операция выполняется над одним операндом, например вывод переменной в поток (”<<”). Для выполнения двуместной операции необходимо два однородных (одного типа) операнда, например сложение двух строк (”+”). В некоторых операциях, например, индексирования указываются два разнородных операнда: тип контейнерного объекта и номер элемента в объекте.

Кроме того, выделяют механизмы с использованием внутренней перегрузки (метод является членом класса) и внешней перегрузки с помощью дружественной функции (более подробно см. ниже). При разных видах перегрузки задается разное число параметров функции-оператора. В результате полная картина различных типов перегруженных операций выглядит так:

- Внутренняя одноместная перегрузка операций (без параметров)
- Внутренняя двуместная перегрузка операций (нужен один параметр)
- Внешняя одноместная перегрузка операций (нужен один параметр)
- Внешняя двуместная перегрузка операций (нужны два параметра)

3.20. Формальное описание перегруженной операции

Формальное описание заголовка перегруженной функции-операции выглядит так:

<тип возврата> operator <знаки операции>(<список формальных параметров>)
{<тело функции>};

Знаки операции это любая доступная стандартная операция (“+”, “-” и т.д.), кроме исключений перечисленных выше. Тип возврата определяет тип результата, который дол-

жен быть возвращен при выполнении операции. Чаще всего это тип объекта (точнее ссылка на тип объекта), который участвует в операции и для которого перегружается данная операция. Список формальных параметров и тело функции соответствуют обычному описанию функции. Служебное слово **operator** указывает на то, что выполняется описание перегруженной операции. Пример описания операций с разным числом параметров:

```
String operator + (String & s) {}; // Перегрузка операции прибавления строки
Test operator ++ () {}; // Перегрузка операции инкремента (без параметров)
friend String operator + (String & s1 , String & s1){};// Перегрузка операции сложения строк
```

В теле класса описывается сам метод перегруженной операции или его прототип. Если используется перегрузка внешняя (дружественная функция), то при описании прототипа задается служебное слово **friend**.

3.21. Внутренняя одноместная перегрузка операций

Для описания внутренней одноместной перегрузки операций используется метод без формальных параметров. Пусть у нас есть класс **Test**, отвлечемся от его содержания и назначения. Для тестового класса зададим следующее описание:

```
class Test {
public:
int Num; // Переменная целого типа
Test(){ Num = 0; };  //  Конструктор без параметров
Test(int i){ Num = i; }; //  Конструктор с параметром
...    };
```

Для операции инкремента можно описать этого класса следующую перегруженную операцию:

```
Test & operator ++() {Num++; return *this; };
```

Для операции декремента можно описать вне описания класса следующую перегруженную операцию (при описании вне класса указание имени **Test::** области видимости нового оператора обязательно):

```
Test & Test::operator --() {--Num; return *this; };
```

Примечание. Перегрузка одноместных операций может быть префиксной и постфиксной. В этом случае для постфиксной перегрузке в описание функции добавляется специальный параметр целого типа (**int**). Подробнее смотри в [1]. Эти вопросы нужно самостоятельно учить в литературе [1].

При этом в классе должен быть обязательно задан прототип перегруженного декремента:

```
Test & operator --(); // Прототип в описании класса
```

Тогда в основной программе мы можем проверить использование наших перегруженных одноместных операций:

```
Test a2(5);
cout << "++a2 = "<< ++a2 <<endl; // справедлива постфиксная записи
cout << "a2++ = "<< a2++ <<endl; // справедлива запись после переменной
cout << "--a2 = "<< --a2 <<endl;
```

Результат работы программы , полученный на консоли, выглядит так:

```
++a2 = 6
a2++ = 7
--a2 = 6
```

3.22. Внутренняя двуместная перегрузка операций

При описании внутренней двуместной операции необходимо указать один параметр, указывающий на второй операнд. Первый операнд двуместной операции задается объектом класса (указатель - **this**), для которого вызвана функция - операция. Для нашего

простого класса можем описать следующий метод операции “+” , описанный внутри класса:

```
Test operator +(Test &  X )
{ Test T;
T.Num = this->Num + X.Num ;
return T;     };
```

Или метод, описанный вне класса:

```
Test2 & Test2::operator -(Test2 &  X) {
Test2 T;
T.Num = this->Num - X.Num ;
return T; };
```

В самом же классе опишем прототип операции “-”:

```
Test2 & operator - (Test2 &  X);
```

Тогда в основной программе мы можем проверить использование определенных выше перегруженных двуместных операций:

```
Test a2(5) , b2(3), c2;
c2 = a2 + b2;
        cout << "c2 = "<< c2 <<endl;
c2 = a2 - b2;
        cout << "c2 = "<< c2 <<endl;
```

Результат работы программы, полученный на консоли, выглядит так:

```
c2 = 7
c2 = 2
```

Нужно знать, что для сложных выражений с операциями, следующую особенность: передача выполняется через стек, который по завершению операции затирается. Если операция планируется для использования в сложных выражениях операции “+” и “–“ нужно описать возврат метода-операции по-другому:

```
Test & operator +(Test &  X )
{
Test *pT = new Test;
pT->Num = this->Num + X.Num ;
return (Test &)*pT;
   };
// Или
Test & Test::operator -(Test &  X) {
Test * pT = new Test;
pT->Num = this->Num - X.Num ;
return (Test &)*pT;
};
```

В примере описания операций для промежуточного результата операции выделяется динамическая память. При сложной операции вида:

```
Test a(3), b(1), c, d(3);
c = a - b + d;
cout << "c = "  << c   <<endl;
```

Получим результат в консольном окне:

```
c = 5
```

Кроме того. Необходимо учесть, что для сложных объектов, особенно объектов использующих динамическую память, необходимо корректно описать операцию присваивания и конструктор копирования, которые создаются для объектов по-умолчанию и обеспечивают только простое копирования содержание объекта (памяти). Эти вопросы нужно самостоятельно учить в литературе [1].

3.23. Дружественные функции и классы

Внешние (глобальные) функции называются дружественными (friend), если в теле описания класса описан их прототип со служебным словом **friend**. Формально это можно записать так:

friend <прототип глобальной функции>;

Глобальная функция должна быть описана после описания класса и доступна в классе. Дружественная функция получает доступ к скрытым членам класса, тех объектов класса, которые ей переданы в качестве параметров или описаны в виде локальных объектов самой функции.

В качестве дружественной функции может выступать и функция член другого класса и весь другой класс в целом (все методы класса дружественные данному классу). В этом случае прототипы членов функций или объявления класса описываются со специальным ключевым словом **friend**. Пример описания дружественных функций и классов:

```
class Test {
...
friend void Print(Test &t); // Глобальная дружественная функция
friend class Test2; // Дружественный класс Test2
friend Test1::Change(Test1 t); // Дружественная функция член класса Test1

};
```

Глобальные дружественные функции могут выполнять перегрузку операций для объектов данного класса. В некоторых случаях такая перегрузка является более удобной и наглядной. Использование механизма дружественности на самом деле является нарушением важного принципа ООП - инкапсуляции, и, по возможности, должно исключаться из программ.

3.24. Внешние одноместная и двуместная перегрузки операций

Формальное описание внешней функции для перегрузки операций приведено ниже:

<тип возврата> operator <знаки операции>(<список формальных параметров>)

{<тело функции>};

Для примера опишем простейший класс **Test1,** содержащий прототипы внешних функций для перегрузки операций:

```
class Test1 {
public:
int Num;
Test1(){ Num = 0; };  //  Конструктор без параметров
Test1(int i){ Num = i; }; //  Конструктор с параметром
friend Test1 & operator ++(Test1 & T); // Дружественная функция перегрузки
friend Test1 operator +( Test1 &  X ,  Test1 &  Y ); // Дружественная функция перегрузки
... };
```

Число параметров при внешней перегрузке необходимо увеличить на единицу. Для одноместной операции - это один параметр, а для двуместной операции – это два параметра. Для глобальных функций перегрузки операций, после описания класса, необходимо описать сами функции так:

```
                // Внешняя перегруженная операция  "+" (двуместная)
Test1  operator+( Test1 &  X ,  Test1 &  Y )
{
        Test1 *pT = new Test1;
   pT->Num = Y.Num + X.Num ;
   return *pT;
};
// Внешняя перегруженная операция  "++" (одноместная)
Test1 & operator ++(Test1 & T1) {
int h = T1.Num;
h++;
T1.Num = h;
return T1;  };
```

Обратите внимание на то, что в дружественных функциях нет спецификации класса, они не являются методами класса. После описания, в основной программе мы можем проверить использование перегруженной двуместной операции:

```
Test1 a(5), b(3), c;
c = a + b;
 cout << "c = "  << c   <<endl;
c++;
 cout << "c = "  << c   <<endl;
```

Получим результат:

```
c = 8
c = 9
```

Для корректного использования перегруженных операций в сложных объектах и выражениях необходимо, чтобы в классах были корректно перегружены операция присваивания и конструктор копирования. Эти вопросы нужно самостоятельно учить в литературе [1].

3.25. Перегрузка операций потоковых операций ввода/вывода

Для примера покажем, как могут быть перегружены операции потокового вывода для пользовательских объектов. Перегрузка потоковых операций выполняется внешней дружественной функцией. Пусть у нас есть класс **Test4**. Тогда пример класса и перегруженной операции вывод может выглядеть так:

```
class Test4 {
public:
int Num;
Test4(int i){ Num = i; };
friend ostream & operator << ( ostream &  out  ,  Test4 &  obj );
friend istream & operator >> ( istream &  in  ,  Test4 &  obj );
};
```

Реализация перегруженного метода вывода имеет вид:

```
ostream & operator <<( ostream &  out  ,  Test4 &  obj )
{
out << obj.Num ;
return out;
};
```

Пример использования показан ниже:

```
Test4 t(5);
cout << "t = " << t << endl;
```

Получаемый при выполнении этого фрагмента результат приведен ниже:

```
t = 5
```

Перегрузка операции ввода выполняется аналогично с помощью внешней дружественной функции:

```
istream & operator >> ( istream &  in  ,  Test4 &  obj )
{
 cout<< "Введите Num: " ;
in >> obj.Num ;
return in;
};
```

Пример использования показан ниже:

```
Test4 t(5);
cin>>t; // Введем -  33
cout << "t = " << t << endl;
```

Получаемый при выполнении этого фрагмента результат приведен ниже:

```
Введите Num:  33
t = 33
```

3.26. Перегрузка операций индексирования

Для новых пользовательских контейнерных классов можно перегрузить операцию индексирования (“[“, “]”). Пусть мы имеем простейший класс типа массив со следующим описанием:

```
class Mas {
 public:
 int Var[20];
 Mas( int * pMas , int Razm)
   {    for (int i = 0 ; i < Razm ; i++) Var[i] = pMas[i];    };
...
 int operator [](int k)       { return Var[k]; };  };
```

В классе предусмотрен конструктор для создания объекта типа целого массива и перегруженный оператор индексирования(“[“, “]”). Теперь, если создать новый объект на основе массива, операцию можно использовать в программах:

```
// Перегрузка []
int iMas[] = {0,1,2,3,4,5};
Mas NewMas( iMas , sizeof(iMas)/sizeof(int));
cout << "NewMas[4] = " << NewMas[4] << endl;
```

После выполнения фрагмента получим следующий вывод на консоль:

```
Введите NewMas[4] =  4
```

3.27. Перегрузка операций вызова функции

Для новых пользовательских контейнерных классов можно перегрузить операцию индексирования (“(“, “0”). Пусть мы имеем простейший класс типа массив со следующим описанием:

```
class Mas {
 public:
 int Var[20];
 Mas( int * pMas , int Razm)
   {    for (int i = 0 ; i < Razm ; i++) Var[i] = pMas[i];    };
...
int operator ()(int k) { return Var[k]; };  };
```

В классе предусмотрен конструктор для создания объекта типа целого массива и перегруженный оператор индексирования(“(“, “)”). Теперь, если создать новый объект на основе массива, операцию можно использовать в программах:

```
// Перегрузка []
int iMas[] = {0,1,2,3,4,5};
Mas NewMas( iMas , sizeof(iMas)/sizeof(int));
cout << "NewMas(2) = " << NewMas(2) << endl;
```

После выполнения фрагмента получим следующий вывод на консоль:

```
Введите NewMas(2) =  2
```

В данном примере мы повторили содержание операции индексирования, но, естественно, алгоритм функции для перегрузки может быть любой.

3.28. КОНТЕЙНЕРЫ: списки и массивы

Практически в любой программе, программной системе выполняется обработка множества объектов (переменных). Для совместного хранения этих объектов используется специальный тип объектов – контейнеры. Контейнеры предназначены для динамического включения, доступа, распечатки, сортировки и удаления объектов. Простейшим видом контейнеров является массив, в котором основной вид доступа к объектам выполняется по

номеру (индексу). Данная ЛР посвящена изучению возможностей контейнеров и методов работы с объектами типа массив. Первоначально познакомимся с общими свойствами и разновидностями контейнеров.

3.29. Контейнеры

Контейнером называется специальная разновидность объектов, которые обеспечивают хранение объектов и доступ к этим объектам, а также их совместную обработку (например, печать, изменение и т.д.). Контейнеры-объекты создаются на основе контейнерных классов, которые имеют специальную структуру, специальные свойства и специальные операции. Примерами контейнеров в практике программирования являются, например:

– окно интерфейса, в которое включены управляющие элементы интерфейса (поля ввода, кнопки и т.д.);
– таблица базы данных (БД), в которой хранятся описания полей БД;
– таблица записей БД, в которую включены отдельные записи.
– объекты улицы с домами, расположенными на них
– И многие другие.

Расширенный перечень примеров контейнерных классов вы можете найти в списке вариантов домашнего задания (ДЗ/КЛР) по дисциплине. Этот список размещен на сайте.

3.30. Контейнерные и элементные классы ДЗ

При создании контейнерного класса предполагается, что в программе будут использоваться объекты, которые будут включаться в контейнерные объекты. Такие объекты называются элементными, а классы, которые создаются для их описания – элементными классами. В зависимости от разновидностей контейнеров, элементных классов может быть много или один. Никаких ограничений на структуру элементных классов не накладывается, в частности одни контейнерные классы могут использоваться для накопления и контейнерных объектов. Для хранения разнородных объектов контейнеров они должны иметь специальную организацию.

В каждом варианте домашнего задания по дисциплине, студенты создают собственные контейнерные классы и собственные элементные классы. Для создания контейнерных классов можно использовать механизмы: наследования от типовых классов системы программирования, полная разработка контейнерных классов и примы шаблонов классов.

3.31. Разновидности контейнеров

Контейнеры могут иметь разные свойства, особенности и разную реализацию. Контейнерные объекты подразделяются: по размерности, по доступу, по упорядоченности, по универсальности и по типу создания.

По размерности контейнеры подразделяются на следующие типы:

– С заданной размерностью (фиксированной) – например, стандартные массивы;
– С динамической размерностью – например, списки;
– С изменяемой размерностью – например, массивы с наращиваемой размерностью.

По доступу контейнеры подразделяются на следующие типы:

– С прямым доступом по индексу (номеру) – например, массивы;
– С прямым доступом по параметру (строка) – например, ассоциативные массивы;

– С последовательным доступом – например, списки;
– Со случайным доступом – например, множества.

По упорядоченности контейнеры подразделяются на следующие типы:
– Упорядоченные контейнеры – например, массивы и списки;
– Неупорядоченные контейнеры – например, множества и мультимножества;

По универсальности контейнеры подразделяются на следующие типы:
– Контейнеры, содержащие однородные объекты (одного типа) – например, шаблонные контейнеры;
– Контейнеры, содержащие разнородные объекты (разных типов) – например, объектные контейнеры;

По типу создания контейнеры подразделяются на следующие типы:
– Шаблонные контейнеры, для создания которых типизируется элементные классы;
– Объектные контейнеры, для создания которых используется базовый объектный класс, от которого наследуются рабочие элементные классы.

В некоторых контейнерных классах одновременно реализуются разные характеристики: есть контейнеры, обеспечивающие одновременно прямой доступ и последовательный доступ; контейнеры с фиксированной и наращиваемой размерностью.

Правильный выбор типа и свойств контейнеров необходим для грамотного решения поставленной задачи построения программы с использованием контейнеров.

3.32. Контейнеры - МАССИВЫ

Простейшим видом контейнерного класса является массив. В базовый язык C/C++ включен стандартный вид массивов. Для этого в скобках описания указывается его размерность (допустимое число для хранения в контейнере), например, примеры описания стандартных массивов, рассмотрены ниже:

```
int iMas[10]; // Элементы массива одного типа int  с номерами от 0 до 9-ти.
#define MAX 15
...
char cMas[MAX]; // Элементы массива одного типа char  с номерами от 0 до 14-ти.
...
const int Razm = 5;
My_Obj oMas[Razm]; //Элементы массива одного типа My_Obj с номерами от 0 до 4-ти.
My_Obj * poMas[Razm]; //Массив указателей на объекты типа My_Obj   0-4
...
int iRaz = 10;
float * pfMas = new float[iRaz];//Динамический массив типа float  с номерами от 0 до 9-ти.
// Доступ к элементом массивов выполняется с помощью индексных выражений
iMas[5] = 5;
cMas[k + 2] = ' A';
oMas[0] = new My_Obj;
pfMas [2] = 5.5f;
```

Первые четыре массива являются статическими (**iMas**, **cMas**, **oMas, poMas**). Размерность этих массивов определяется до начала компиляции. Изменить их размерность при выполнении программы невозможно. Четвертый массив, объявленный через указатель (**pfMas**), является динамическим, но с фиксированной размерностью: при выполнении программы увеличить или изменить размерность достаточно сложно. Недостатками таких массивов является также следующее: заранее определенный тип однородных элементных объектов; сложность добавления элементов массивов в середину массива; невозможность доступа к к такому массиву по имени (см. - ассоциативные массивы).

Для устранения перечисленных недостатков в современные системы программирования включены системы классов для работы с массивами, как с контейнерными объектами. Они определены в STL (**vector**), ATL (**CAtlArray**) и MFC(**CArray**, **CObArray**). Эти классы имеют различные методы и обеспечивают много возможностей для описания и работы с объектами типа массив.

3.33. Операции с контейнерами

При работе с контейнерными объектами можно выделить следующие основные операции:

- Создание (описание) контейнеров, с указанием размерности контейнера и его заполнения;
- Добавление объектов в контейнер;
- Удаление объектов из контейнера;
- Доступ к объектам контейнера;
- Очистка контейнера;
- Распечатка содержимого контейнера;
- Расширение размеров контейнера;
- Получение числа элементных объектов в контейнере;
- Обмен местоположения объектов в контейнере.
- Сравнение контейнеров (операции отношения контейнеров).

3.34. Итераторы и их применение

Для удобства и универсализации работы с контейнерами используются специальные разновидности объектов – итераторы. Итераторы создаются для объектов контейнеров и позволяют обеспечить последовательный доступ к объектам контейнера. Доступ может выполняться как в прямом (**iterator**), так и в обратном порядке (**reverse_iterator**). Итераторы обеспечивают выполнения следующих операций: последовательного движения по элементам контейнера (++, --), проверки достижения конца контейнера и доступа к конкретному элементу контейнера. Использование итераторов характерно для контейнеров библиотеки STL.

3.35. Операции с контейнерами типа массив

При работе с контейнерными объектами типа массив можно выделить следующие основные операции (в скобках показанные названия методов):

- Создание (описание) контейнеров, с указанием размерности контейнера и его заполнения (**vecror**, **CAtlArray**, **CArray**, **CObArray**);
- Добавление объектов в массив (например: **push_back , add, InsertAt**);
- Удаление объектов из массива (например: **pop_back , erase,RemoveAt**);
- Доступ к объектам массива (например**: [], at, GetAt**);
- Очистка контейнера-массива (например: **clear, Removeall**);
- Расширение размеров контейнера (например: **resize, SetAtGrow**);
- Получение числа элементных объектов в контейнере (например: **size, GetSize, GetCount**);
- Обмен местоположения объектов в контейнере (например: **swap**);
- Операции над массивами (например: **copy**, **append**, **assign**);
- Сравнение контейнеров (операции отношения контейнеров).

3.36. Контейнерные классы в VS

В системе программирования VS предусмотрено значительное разнообразие контейнерных классов для разных применений. Они содержатся в разных библиотеках:

- Библиотеке STL (vector, list, map, stack, set, multiset, map, multimap и queue);
- Библиотеке MFC (CArray, CList, CMap, CObArray, CObList, CMapPtrToPtr и др);
- Библиотеке ATL (CAtlArray, CAtlList, CAtlMap и др.);
- Библиотеке платформы .NET (Array, List и др);

3.37. Контейнерные классы массивов в STL (vector)

В библиотеке STL описывается шаблонный класс **vector**, который позволяет описывать массивы переменных любого типа. Формальное определение шаблона дано ниже:

```
template <
  class Type,
  class Allocator = allocator<Type>
> class vector {…};
```

Для работы объектами этого класса необходимо подключить заголовочный файл:

```
#include <vector>
```

Для переменных вектора нужно указать тип:

```
vector <int> v1; // Вектор/массив - переменных целого типа
v1.push_back( 1 ); // Добавить в конец
v1.push_back( 2 ); // Добавить в конец
v1.insert( v1.begin( ), 55 );// Добавить в начало
```

Для описания вектора (массива) целых. Для распечатки массива можно использовать следующий фрагмент программы (для вектора перегружены потоковые операции вывода целого типа):

```
for(int i = 0; (unsigned) i < v1.size( ) ; i++)
{ cout << "v1["<< i <<"] = " << v1[i] << endl;};
```

В результате работы данного фрагмента, совместно с предыдущим, получим:

```
v1[0] = 55
v1[1] = 1
v1[2] = 2
```

Перечень всех методов класса вектор приведен в разделе Справочные материалы (данных МУ), подробное описание методов вы можете найти в документации, литературе и справочной системе MSDN[5]. Рассмотрим еще несколько примеров использования объектов вектор. Для очистки массива может быть использован метод удаления (**erase**):

```
  v1.erase( v1.begin( ), v1.end( ) ); // Очистить весь массив
  v1.erase( v1.begin( ), v1. begin ( ) + 2 ); // Очистить 2 первых элемента
```

Получить текущую размерность массива можно так:

```
cout << "Размер массива v1 = "<<  v1.capacity( ) << endl;
```

Получить предельную максимально возможную размерность массива:

```
cout << "Максимальный размер массива v1 = "<<  v1.max_size( ) << endl;
```

Проверить является ли массив пустым можно специальным методом:

```
if ( v1.empty( ) )
     cout << "Массив пуст." << endl;
 else
```

```
    cout << "Массив не пуст." << endl;
```

Поменять содержимое массивов местами:

```
v1.swap( v2 );
```

Рассмотрим использование итераторов для массива типа вектор. Итератор нужно предварительно описать, а затем использовать в цикле распечатки массива. Для навигации по массиву можно применить перегруженную для итератора операцию - “++”. Например:

```
vector<int>::iterator iter;
...
for( iter = v1.begin(); iter != v1.end()  ; iter++)
{ cout << "v1[] = " << *iter << endl;};
```

Методы **begin** и **end** используются для управления циклом на основе итератора. Первый метод (**begin**) задает первый элемент вектора, а второй (**end**) указывает на последний элемент массива. Можно объявить реверсивный итератор и просматривать массив с конца до начала, тогда фрагмент программы может выглядеть так:

```
int i ;
vector <int>::reverse_iterator riter;
...
for( i= 0, riter = v1.rbegin(); riter != v1.rend()  ; riter++, i++)
{
cout << "v1["<<i <<"] = " << *riter << endl;};};
```

В рамках ЛР необходимо продемонстрировать использование методов класса вектор для объектов целого типа и собственного класса по варианте ЛР. Использование собственного класса в качестве содержимого векторного массива имеет особенности. На примере простого класса Point покажем их. Пусть создан простой класс:

```
class Point {
public:
int x; int y;
Point(){ x =0 ; y = 0;};
Point(int a , int b ){ x = a ; y = b;};
friend ostream & operator <<( ostream &  out  ,  Point &  obj );};
// Перегруженная дружественная операция вывода в поток
ostream & operator <<( ostream &  out  ,  Point &  obj )
{
out  <<"{ x = " << obj.x <<" y = " << obj.y << " } " <<endl;
return out;
};
```

Тогда мы можем описать вектор массив для этих объектов и итератор для него:

```
vector<Point> vP1;
vector<Point>::iterator PIter;
...
```

Опишем объекты типа Point и добавим их в конец массива:

```
Point P1(1,2);
Point P2(51, 52);
vP1.push_back( P1 );
vP1.push_back( P2 );
vP1.push_back( *new Point(31 ,32) ); // Динамические создание
```

Цикл печати построим с помощью итератора и вспомогательной переменной, позволяющей выводить индекс массива (i):

```
for( i = 0, PIter = vP1.begin() ; PIter != vP1.end( ) ; PIter++, i++)
{ cout << "vP["<<i<<"] = " << *PIter << endl;};
```

В результате получим в консольном окне:

```
vP[0] = { x = 1 y = 2 }
vP[1] = { x = 51 y = 52 }
```

vP[2] = { x = 31 y = 32 }

Особенностью использования собственных объектов является необходимость перегрузки операций ввода/вывода в поток. Другой особенностью класса vector является автоматическое корректное удаление динамических объектов, созданных в программе и включенных в массив. В ЛР необходимо описать собственный класс объектов и продемонстрировать все возможности массива типа vector.

3.38. Контейнерные классы массивов в MFC

В библиотеке MFC (Microsoft Foundation Class) предусмотрено два основных класса для работы с массивами: **CArray** и **CObArray**. Кроме этих классов предусмотрены классы для определенного типа включаемых объектов: CWordArray, CByteArray, CPtrArray, CDWordArray, CStringArray, CUintArray. Класс **CArray** является шаблонным, для создания объектов необходимо указать тип, а класс **CObArray** не требует типизации, в него могут включаться любые объекты, наследованные от класса **CObject**. Методы первого и второго классов практически совпадают по названию и назначению. Рассмотрим первоначально класс **CArray**. Для добавления в массив объектов используется метод **Add**. Элементы всегда добавляются в конец массива. Формально массивы типа CArray являются шаблонами:

```
template < class TYPE, class ARG_TYPE = const TYPE& >
class CArray :
  public CObject { …};
```

Пример описания и добавления в массив переменных типа **int**.

```
CArray<int , int> IntMas;
for (i = 0 ; i < 5 ; i++ )
      IntMas.Add(i);
for (i = 0 ; i < IntMas.GetCount() ; i++ )
      cout << IntMas[i] << " "  ;
      cout << endl;
```

Метод **GetCount** позволяет получить текущий размер массива, поэтому он может быть использован для проверки завершения цикла. После выполнения получим результат:

```
0 1 2 3 4
```

Метод **GetUpperBound** позволяет получить последний занятый индекс в массиве:

```
cout << "GetUpperBound = " << IntMas.GetUpperBound() << endl;
```

Получим:

```
4
```

Метод **GetAt** позволяет получить значение элемента списка по индексу (номеру):

```
i = IntMas.GetAt(2);
cout << "i = "<< i << "  IntMas[2] = " << IntMas[2] << endl;
```

Получим результат:

```
i = 2  IntMas[2] = 2
```

Метод **SetAt** позволяет заменить элемент в массиве на другой:

```
IntMas.SetAt(3, 10); // Значение элемента номер 3 заменяется на 10
for (i = 0 ; i < IntMas.GetCount() ; i++ )
      cout << IntMas[i] << " "  ;       cout << endl;
```

Получим результат:

```
0 1 2 10 4
```

С помощью метода **InsertAt** можно добавить в середину:

```
IntMas.InsertAt(1, 100);
for (i = 0 ; i < IntMas.GetCount() ; i++ )
      cout << IntMas[i] << " "  ;     cout << endl;
```

Получим результат:

```
0 100 1 2 10 4
```

С помощью метода **RemoveAt** можно удалить любой элемент по номеру:

```
IntMas.RemoveAt(2);
for (i = 0 ; i < IntMas.GetCount() ; i++ )
      cout << IntMas[i] << " "  ;     cout << endl;
```

Получим результат:

```
0 100 2 10 4
```

С помощью метода **IsEmpty** можно проверить наличие в массиве элементов, а с помощью метода **RemoveAll**, удалить все элементы:

```
if ( IntMas.IsEmpty() )
{ cout << "Массив пуст!" <<endl; }
else
{ cout << "Массив не пуст!" <<endl; };
IntMas.RemoveAll(); // Удаление всех элементов
if ( IntMas.IsEmpty() )
{ cout << "Массив пуст!" <<endl; }
else
{ cout << "Массив не пуст!" <<endl; };
```

Получим результат:

```
Массив не пуст!
Массив пуст!
```

Методы **SetSize** и **FreeExtra** позволяют установить новую размерность массива и удалить свободную память:

```
IntMas.SetSize(32, 128); // Размер 32
cout << "GetSize = " << IntMas.GetSize() << endl;
cout << "GetUpperBound = " << IntMas.GetUpperBound() << endl;
IntMas.SetSize(10, 128); // Новый Размер 10
IntMas.FreeExtra();
cout << "GetSize = " << IntMas.GetSize() << endl;
cout << "GetUpperBound = " << IntMas.GetUpperBound() << endl;
```

Получим результат:

```
GetSize = 32
GetUpperBound = 31
GetSize = 10
GetUpperBound = 9
```

Специальными методами можно копировать (**Copy**) или добавлять к существующему массиву (**Append**) однотипные массивы:

```
CArray<int , int> IntMas1;
IntMas1.Add(55); // Добавим 1 элемент в массив IntMas1
IntMas1.Append( IntMas );                 // Добавление массива IntMas с массиву IntMas1
for (i = 0 ; i < IntMas1.GetCount() ; i++ )
      cout << IntMas1[i] << " "  ;     cout << endl;
IntMas.Copy( IntMas1 );                   // Копирование массива IntMas1 в массив IntMas
for (i = 0 ; i < IntMas.GetCount() ; i++ )
      cout << IntMas[i] << " "  ;     cout << endl;
```

Получим результат:

```
55 0 100 2 10 4
55 0 100 2 10 4
```

Для класса массивов **CObArray** все методы аналогичны. Существенное отличие заключается в том, что такой массив поддерживает массив указателей на объекты типа **CObject**, или точнее, объекты классов наследованных от **CObject**. В этом случае их описание эквивалентно такому:

```
CArray< CObject *, CObject*> mArr; // Нужна настройка шаблона на CObject
CObArray mArrObj;  // Настройки шаблона не нужно
```

Если класс **Point** является наследником класса **CObject**, то все предыдущие фрагменты проверки работы методов для нашего класса объектов должны нормально функционировать, например:

```
class Point : public CObject { … };
…
for (i = 0 ; i < 5 ; i++ )
        mArrObj.Add(new Point(i,i));
cout << " ";
for (i = 0 ; i < mArrObj.GetCount() ; i++ )
        cout << *((Point *)(mArrObj[i])) << " "  ; //явное  приведение типа нужно для
// использования перегруженного метод вывода в поток
        cout << endl;
```

Получим результат:

```
{ x = 0 y = 0 }
{ x = 1 y = 1 }
{ x = 2 y = 2 }
{ x = 3 y = 3 }
{ x = 4 y = 4 }
```

Единственное отличие состоит в необходимости преобразования указателя к типу Point для автоматического вызова перегруженной операции вывода. Кроме этого, необходимо учесть что добавление, выборка объектов должны выполняться с указанием объектов соответствующих типов, наследованных от класса **CObject**.

3.39. Контейнерные классы массивов в ATL

Контейнерный класс в библиотеке ATL называется **CAtlArray**. Он является шаблонным классом и имеет следующее формальное описание:

template< typename E, class ETraits = CElementTraits< E >
> class CAtlArray { … };

В этом классе все очень похоже на класс **CArray** из MFC, но некоторые методы отсутствуют. В частности недоступны методы: **GetUpperBound**, **GetSize** и **SetSize**. Все остальные методы и свойства совпадают с массивами библиотек MFC.

При подключении классов библиотеки ATL в главный модуль нужно добавить следующий заголовочный файл:

```
#include <atlcoll.h>
```

Для примера использования класса **CAtlArray** рассмотрим фрагмент программы с массивом целых чисел:

```
CAtlArray<int> atlMas;
…
for (i = 0 ; i < 5 ; i++ )
        atlMas.Add(i);
for (i = 0 ; i < atlMas.GetCount() ; i++ )
        cout << atlMas[i] << " "  ;
        cout << endl;
```

После выполнения этого фрагмента программы получим результат:

```
0 1 2 3 4
```

Если в предыдущих фрагментах программ удалить строки с недоступными функциями и заменить название массивов, они будут, несомненно, работать.

3.40. Контейнеры - Списки

Контейнеры типа список обеспечивают работу с множеством объектов, число которых заранее неизвестно и может изменяться во время работы программы. Кроме того,

порядок этих объектов может также изменяться, например, с добавлением новых объектов в произвольное место упорядоченного множества. Работа с массивами, особенно если их размерность велика, приводит к значительным временным затратам процессорного времени. Хотя прямого доступа к элементам списка не определяется (операция индексирования недоступна - просмотр списка возможет только последовательно), динамические возможности перемещения по списку, его изменения, поиска и сортировки выполняются значительно эффективнее, чем в массивах. Контейнерные объекты типа список применяются практически во всех программах средней и большой сложности. На основе списков строятся более сложные структуры данных: деревья, сетевые структуры данных и многие другие.

3.41. Навигация, позиции и ссылки на объекты

При доступе к элементам контейнера типа список отсутствует возможность указания индекса элемента, по сравнению с массивами. Для доступа к конкретному элементу используется специальный тип данных, задающий позицию, – **POSITION**. Переменная этого типа фактически является адресом элемента списка, который содержит объект. Специальные методы (**GetHeadPosition GetTailPosition ,GetNext** и **GetPrev**) позволяют перемещаться по списку в прямом и обратном направлении (такое перемещение с определением новой позиции похоже на работу итераторов). С помощью специальных методов на основе переменной типа позиция может быть выбран объект из списка (**GetAt**). Такая навигация характерна для классов: **CList**, **CObList** и **CAtlList**. В отличие от использования итераторов **STL**, которые по сути играют такую же роль, операции инкремента для позиций недоступны. Пример:

```
POSITION pos;
...
pos = IntList.GetHeadPosition(); // Установка позиции первого элемента списка IntList
  cout << "IntList =  " << IntList.GetAt(pos) << endl;
    IntList.GetNext( pos); // Получение позиции следующего элемента
...
```

Если после выполнения операции значение pos становиться равным нулю (NULL), то это означает, что достигнут конец списка.

Методы доступа к элементам контейнера чаще возвращают ссылку на объект или сам объект, а не указатель на него. В этом случае необходимо предусматривать явное преобразование типов, например:

```
CObList PoList; // Список ссылок/ указателей на базовый класс CObject
...
  cout << "PList =  " << *((Point *)(PoList.GetAt(pos))) << endl; //Приведение типа
```

Первоначально полученный тип приводиться явно к указателю на класс Point, и только затем выполняется разыменование и получается объект.

3.42. Операции с контейнерами типа список

При работе с контейнерными объектами типа список можно выделить следующие основные операции (в скобках показанные названия методов):

- Создание (описание) контейнеров, с указанием размерности контейнера и его заполнения (**list**, **CAtlList**, **CList**, **CObList**);
- Добавление объектов в список (например: **push_back , addHead, InsertAfter**);
- Удаление объектов из списка (например: **pop_back , erase, RemoveAt**);
- Доступ к объектам списка (например**: GetAt, GetHead, GetTail**);
- Очистка контейнера - списка (например: **clear, erase, RemoveAll**);

– Получение числа элементных объектов в контейнере (например: **size, GetSize, GetCount**);
– Поиск в списке (например: **Find** , **FindIndex**);
– Проверка пустого (например: **empty**, **IsEmpty**)
– Сортировка (например: **sort**, **reverse**)
– Обмен местоположения объектов в контейнере (например: **swap**);
– Операции над массивами (например: **copy**, **append**, **assign**);
– Навигация по спискам (например: **GetNex**, **GetPrev**, **GetHeadPosition, GetTailPosition**);
– Сравнение контейнеров (операции отношения контейнеров).

3.43. Контейнерные классы списков в STL (list)

В библиотеке STL описывается шаблонный класс **list**, который позволяет описывать списки переменных любого типа. Формальное определение шаблона дано ниже:

```
template <
  class Type,
  class Allocator=allocator<Type>
> class list{ … };
```

Для работы объектами этого класса необходимо подключить заголовочный файл:

```
#include < list >
```

Для описания списков предусмотрены разные конструкторы. Примеры описания списков даны ниже:

```
  list <int> l0; // Пустой список l0
  list <int> l1( 3 ); // Список с тремя элементами равными 0
  list <int> l2( 5, 2 ); // Список из пяти элементов равными 2
  list <int> l3(l2); // Список l3 на основе списка l2
  list <int>::iterator l3_Iter; // Описания итератора
  l3_Iter = l3.begin( );  //
  l3_Iter++; l3_Iter++;
  list <int> l4( l3.begin( ), l3_Iter ); // Новый список l4 на основе первых двух элементов L3
```

Для описания пустого списка l (типа list) нужно указать тип int:

```
list <int> l;
list<int>::iterator iter;
…
cout<< "Добавление:" << endl;
  l.push_back( 1 ); // Добавление в конец списка
  l.push_back( 2 ); // Добавление в конец списка
  l.push_front( 5 ); // Добавление в начало списка
lPrint(l); // Собственная функция печати целого списка
```

Функцию печати опишем в заголовочном файле так, чтобы в ней в данной функции использован прямой итератор для класса list, а для индексации элементов списка вспомогательная целая переменная:

```
void lPrint(list<int>& l)
{
list<int>::iterator iter;
int i = 0;
if ( !l.empty() ) {  // Проверка пустого списка
for( iter = l.begin(); iter != l.end()  ; iter++ , i++)
{       cout << "l["<< i << "] = " << *iter << endl;} }
else
cout << "Список пуст!"<< endl;};
```

В данной функции используется итератор **iter**, который позволяет продвигаться по списку (**iter++**). Для установки итератора на первый элемент используется метод **begin**. Остановка просмотра проверяется методом **end**. Для проверки пустого списка используется метод **empty**. Для дальнейшей демонстрации возможностей списков будем использовать эту функцию. В первом фрагменте получим результат:

```
Добавление:
l[0] = 5
l[1] = 1
l[2] = 2
```

Перечень методов класса список приведен в разделе Справочные материалы, подробное описание методов вы можете найти в документации, литературе и справочной системе MSDN[5]. Рассмотрим еще несколько примеров использования объектов класса список и его методов. Для копирования списков может быть использован метод слияния (**assign**), а для очистки списка может быть использован метод удаления (**clear**):

```
// Копирование и очистка
list <int> l20;
lPrint(l);
l20.assign( l.begin( ), l.end( ) );
cout<< "После копирования l в l20:" << endl;
lPrint(l20);
l20.clear( );
cout<< "После очистки l20:" << endl;
```

В первом фрагменте получим результат:

```
После копирования l в l20:
l[0] = 5
l[1] = 1
l[2] = 2
После очисткиl20:
Список пуст!
```

Получить текущую размерность списка можно методом **size**:

```
cout << "Число элементов в списке =  " << l.size( )<< endl;
```

Получим результат:

```
Число элементов в списке =  3
```

Манипулировать с содержимым списка (вставка, удаление и т.д.) можно с помощью методов (**insert**, **remove**, **erase**, **remove_if** и др.). Здесь приводиться работоспособный фрагмент программы, а комментарии даны в тексте программы:

```
// Вставка
   iter = l.begin( );
   iter++; iter++;
   cout<< "Вставка:" << endl;
   l.insert( iter, 100 );
lPrint(l);
// Удаление по числу
  l.remove( 100 );
     cout<< "Удаление по числу 100:" << endl;
lPrint(l);
// Удаление по номеру
  l.erase( l.begin( )  ); // Из головы списка
     cout<< "Удаление по номеру 0:" << endl;
lPrint(l);
// Удаление по итератору
   l.push_back( 20 );
   l.push_back( 31 );
   cout<< "Перед удалением:" << endl;
lPrint(l);
   iter = l.begin( );
```

```
  iter++; iter++; // Сдвиг на два элемента
  l.erase( iter  );
    cout<< "Удаление по итератору 2 (3-й элемент):" << endl;
// Удаление по условию (предикату)
lPrint(l);
  list <int> l2 = l;
  cout<< "Удаление четных чисел:" << endl;
  cout<< " До удаления:" << endl;
  l2.remove_if( is_odd<int>( ) );
  cout<< " После удаления:" << endl;
lPrint(l2);
```

В последнем случае нужно создать специальный класс с булевским оператором шаблонного типа (унарный одноместный предикат - функция):

```
template <class T> class is_odd : public std::unary_function<T, bool>
{
public:
  bool operator( ) ( T& val )
  {
  return ( val % 2 ) != 1;    } }; // Условие предиката – четное = 0
```

Это позволяет установить для каждого элемента списка значение условия для его удаления. В нашем случае выполняется проверка на честность параметра элемента, передаваемого в функцию. В задании на ЛР необходимо для своего целого списка объявить шаблонный класс для условия удаления элементов с заданным значением переменной.

Для предыдущего фрагмента удаления четных значений из l2, созданного на основе l, получим:

```
Удаление четных чисел:
До удаления
l[0] = 1
l[1] = 31
l[2] = 20
После удаления четных
l[0] = 1
l[1] = 31
```

Сортировка списка выполняется методом **sort**, при этом может быть указан параметр или не указан. Если параметр не задан, то сортировка выполняется в порядке возрастания ключевой переменной (**less**). При задании параметра сортировки **greater** сортировка выполняется в порядке убывания основной переменной списка. Примеры и результат приведены ниже.

```
// Сортировка
  l.sort( ); // Сортировка по возрастанию по-умолчанию less
  cout<< "После сортировки списка!" << endl;
lPrint(l);
  l.sort( greater<int>( ) ); // Сортировка по убыванию
  cout<< "После greater сортировки списка!" << endl;
lPrint(l);
  l.sort( less<int>( ) ); // Сортировка по возрастанию
  cout<< "После less сортировки списка!" << endl;
lPrint(l);
```

Переупорядочение списка, изменение порядка на обратный, выполняется методом **reverse**. Пример:

```
cout<< "После reverse сортировки списка!" << endl;
l.reverse( );
lPrint(l);
```

Получим результат:

```
После сортировки списка!
l[0] = 1
l[1] = 2
```

```
l[2] = 20
l[3] = 31
После greater сортировки списка!
l[0] = 31
l[1] = 20
l[2] = 2
l[3] = 1
После less сортировки списка!
l[0] = 1
l[1] = 2
l[2] = 20
l[3] = 31
После reverse сортировки списка!
l[0] = 31
l[1] = 20
l[2] = 2
l[3] = 1
```

Очистка списка (**clear**):

```
l.clear( );
cout<< "После очистки списка!" << endl;
lPrint(l);
```

Получим результат:

```
Пустой список!
```

3.44. Использование класса list для включения объектов своего класса

В рамках ЛР необходимо продемонстрировать использование методов класса список для объектов собственного класса, который определяется вариантом домашнего задания. Использование собственного класса в качестве содержимого списка ряд особенностей. На примере простого класса **Point** покажем примеры. Пусть создан простой класс (наследование от класса **CObject**, здесь не используется, а будет необходимо в последующих примерах):

```
class Point : CObject {
public:
int x; int y;
// Конструкторы
Point(){ x =0 ; y = 0;};
Point(int a , int b ){ x = a ; y = b;};
// Перегружественная дружественная операция вывода в поток
friend ostream & operator <<( ostream &  out  ,  Point &  obj );
};

ostream & operator <<( ostream &  out  ,  Point &  obj )
{
out  <<"{ x = " << obj.x <<" y = " << obj.y << " } " <<endl;
return out;
};
```

Тогда мы можем описать разнообразные списки этих объектов. Ниже приведены примеры вариантов использования конструкторов:

```
Point P2(51,52); // Описание точки для заполнения конструктором
  list <Point> lp; // Пустой список lp0
  list <Point> lp1( 3 ); // Список с тремя элементами равными 0
  list <Point> lp2( 5, P2 ); // Список из пяти элементов равными P2
  list <Point> lp3(lp2); // Список l3 на основе списка l2
  list <Point>::iterator lp3_Iter;
  lp3_Iter = lp3.begin( );
```

```
  lp3_Iter++; lp3_Iter++;
  list <Point> lp4( lp3.begin( ), lp3_Iter ); // Новый список lp4 на основе первых двух элементов lp3
  cout<< "КОНСТРУКТОРЫ: СОБСТВЕННЫЙ КЛАСС!!!" << endl;
lPPrint (lp2);
lPPrint (lp4);
```

Тогда мы можем описать списки этих объектов и итератор для него:

```
list<Point> lP; // Пустой список
list<Point>::iterator PIter; // Итератор для списка
Point P1(1,2);
Point P2(51,52);
// Заполнение списка
lP.push_back( P1 );
lP.push_back( P2 );
lP.push_front( *new Point(31 ,32) );
Print_list(lP);
```

Метод печати списка (**Print_list**) можно объявить сразу для всех типов объектов в виде шаблона следующего приведенного ниже. Он будет работать для всех списков для объектов, в которых перегружена операция вывода:

```
// Шаблон функции печати списка list
template <class T> void Print_list(T & l)
{ list<Point>::iterator iter;
int i = 0;
if ( !l.empty() )
{
for( iter = l.begin(); iter != l.end()  ; iter++ , i++)
{       cout << "l["<< i << "] = " << *iter ;} }
else
cout << "Список пуст!"<< endl;
cout << endl; };
```

Все другие методы класса list из STL можно использовать для класса Point. Рассмотрим другие примеры. Метод позволяет **assign** добавить в новый список элементы из первого списка

```
list <Point> lp20;
cout<< "Исходный список lP:" << endl;
Print_list(lP);
Print_list(lp20);
lp20.assign( lP.begin( ), --lP.end( ) );
cout<< "После копирования lP в lp20:" << endl;
Print_list(lp20);
lp20.clear( );
cout<< "После очистки lp20:" << endl;
Print_list(lp20);
```

Получим результат, при копировании всего списка без последнего элемента (**--lP.end( )**):

```
Исходный список lP:
l[0] = { x = 31 y = 32 }
l[1] = { x = 1 y = 2 }
l[2] = { x = 51 y = 52 }
Список пуст!
После копирования lP в lp20:
l[0] = { x = 31 y = 32 }
l[1] = { x = 1 y = 2 }
После очистки lp20:
Список пуст!
```

Манипулировать с содержимым списка (вставка, удаление и т.д.) можно с помощью методов (**insert, erase, remove_if** и др.):

```
// Вставка по итератору
```

```
   PIter = lP.begin( );
   PIter++; PIter++;
   cout<< "Вставка до:" << endl;
   Print_list(lP);
   cout<< "Вставка после:" << endl;
   lP.insert( PIter, P1 );
Print_list(lP);
// Удаление по итератору
  lP.erase( lP.begin()  );
  cout<< "Удаление по итератору (0, конец):" << endl;
Print_list(lP);
// Простой предикат для удаления x = 20
   lP.remove_if( is_X20<Point>( ) ); // Работает!!!
   cout<< "После удаления x = 20 списка!" << endl;
Print_list(lP);
```

Простой унарный предикат (задается в виде булевской унарной функции шаблонного класса) для удаления точек с условием x = 20 выглядит так:

```
template <class T> class is_X20 : public std::unary_function<T, bool>
{ public:
   bool operator( ) ( T& val )
   {    return ( val.x = = 20 ); // Возвращается истина, если объект в списке имеет x = 20
   } };
```

В результате выполнения данного фрагмента программы получим:

```
Вставка до:
l[0] = { x = 31 y = 32 }
l[1] = { x = 1 y = 2 }
l[2] = { x = 51 y = 52 }

Вставка после:
l[0] = { x = 31 y = 32 }
l[1] = { x = 1 y = 2 }
l[2] = { x = 1 y = 2 }
l[3] = { x = 51 y = 52 }

Удаление по итератору (0, конец):
l[0] = { x = 1 y = 2 }
l[1] = { x = 1 y = 2 }
l[2] = { x = 51 y = 52 }

l[0] = { x = 20 y = 32 }
l[1] = { x = 1 y = 2 }
l[2] = { x = 1 y = 2 }
l[3] = { x = 51 y = 52 }

После удаления x = 20 списка!
l[0] = { x = 1 y = 2 }
l[1] = { x = 1 y = 2 }
l[2] = { x = 51 y = 52 }

l[0] = { x = 51 y = 52 }
l[1] = { x = 1 y = 2 }
l[2] = { x = 1 y = 2 }
```

Функция **remove** (удалить по объекту) требует описания перегруженной операции равенства, присваивания и явных конструкторов копирования. Например:

```
Point(const Point & p){ x = p.x ; y = p.y ;};
Point & operator=(const Point & p){ x = p.x ; y = p.y ; return *this;};
friend bool operator==(const Point & p1 , const Point & p2) ;
```

```
...
// Перегурузка операции сравнения – равенства  для удаления по объекту
bool operator==(const Point & p1 , const Point & p2)
{
return ( (p1.x==p2.x) && (p1.y==p2.y) );
};
```

Тогда можно выполнить следующий фрагмент текста с функцией **remove**:

```
 lP.insert( PIter, P2 );
 lP.insert( PIter, *new Point(51, 61));
 Print_list(lP);
 lP.remove( P2 );
 cout<< "Удаление по объекту P2 (после):" << endl;
 Print_list(lP);
```

В результате выполнения данного фрагмента программы получим (удаляются точки с координатами <51,52>):

```
Удаление по объекту P2 (до):
l[0] = { x = 31 y = 32 }
l[1] = { x = 1 y = 2 }
l[2] = { x = 1 y = 2 }
l[3] = { x = 51 y = 52 }
l[4] = { x = 51 y = 61 }
l[5] = { x = 51 y = 52 }
l[6] = { x = 51 y = 61 }
l[7] = { x = 51 y = 52 }
Удаление по объекту P2 (после):
l[0] = { x = 31 y = 32 }
l[1] = { x = 1 y = 2 }
l[2] = { x = 1 y = 2 }
l[3] = { x = 51 y = 61 }
l[4] = { x = 51 y = 61 }
```

Функция **remove_if** тоже требует описания унарных предикатов. Пусть мы имеем доступную глобальную переменную:

```
Point GP(51,52);
```

Тогда можно описать унарный предикат вида:

```
// Унарная функция для предикта условного удаления по объекту
template <class T> class is_XA : public std::unary_function<T, bool>
{
public:
  bool operator( ) (  T& val   )
  {
  return ( val = = GP );    } // Использование перегруженной операции "= ="
};
```

После этого можно выполнить следующий фрагмент программы:

```
 cout<< "Удаление по условию = Point(51,52) P2 (до):" << endl;
 lP.insert( PIter, P2 );
 Print_list(lP);
 GP.x = 51; GP.y = 52;
 lP.remove_if( is_XA<Point>( )  );
 cout<< "Удаление по условию = Point(51,52) P2 (после):" << endl;
 Print_list(lP);
```

Получим в результате

```
Удаление по условию = Point(51,52) P2 (до):
l[0] = { x = 51 y = 52 }
l[1] = { x = 51 y = 52 }
l[2] = { x = 31 y = 32 }
```

```
l[3] = { x = 1 y = 2 }
l[4] = { x = 1 y = 2 }
l[5] = { x = 51 y = 61 }
l[6] = { x = 51 y = 61 }

Удаление по условию = Point(51,52) P2 (после):
l[0] = { x = 31 y = 32 }
l[1] = { x = 1 y = 2 }
l[2] = { x = 1 y = 2 }
l[3] = { x = 51 y = 61 }
l[4] = { x = 51 y = 61 }
```

Функции **sort** требуют описания перегруженных операций сравнения (“>”,”<”) для объекта **Point**. Для упрощения предиката зададим сортировку только по параметру x. Например, дружественной функцией:

```
...
friend bool operator>(const Point & p1 , const Point & p2) ; // Описание в классе Point
friend bool operator<(const Point & p1 , const Point & p2) ; // Описание в классе Point
...
// Перегурузка операции “<” для сортировки (описание вне класса)
bool operator<(const Point & p1 , const Point & p2)
{
return ( p1.x < p2.x );
};
//
// Перегрузка операции “>”для сортировки
bool operator>(const Point & p1 , const Point & p2)
{
return ( p1.x > p2.x );
};
```

Пример использования метода sort показан ниже, всеемте со следующим. Используемый метод **reverse** позволяет изменить порядок расположения объектов на обратный порядок. После этого можно выполнить следующий фрагмент программы:

```
  lP.push_front( *new Point(20 ,32) );
   cout<< "Обратный порядок (до):" << endl;
 Print_list(lP);
  lP.reverse( ); // Изменение порядка на обратный
   cout<< "Обратный порядок (после):" << endl;
  Print_list(lP);
   cout<< "Сортировка по возрастанию x:" << endl;
  lP.sort( ); // По умолчанию less (меньше) – сортировка по возрастанию
  Print_list(lP);
 cout<< "Сортировка по убыванию x:" << endl;
  lP.sort( greater<Point>( ) ); // здесь используются операции отношения и сортировка по
// убыванию greater
  Print_list(lP);
```

Получим в результате:

```
Обратный порядок (до):
l[0] = { x = 20 y = 32 }
l[1] = { x = 31 y = 32 }
l[2] = { x = 1 y = 2 }
l[3] = { x = 51 y = 61 }
l[4] = { x = 51 y = 61 }
Обратный порядок (после):
l[0] = { x = 51 y = 61 }
l[1] = { x = 51 y = 61 }
l[2] = { x = 1 y = 2 }
```

```
l[3] = { x = 31 y = 32 }
l[4] = { x = 20 y = 32 }
Сортировка по возрастанию x:
l[0] = { x = 1 y = 2 }
l[1] = { x = 20 y = 32 }
l[2] = { x = 31 y = 32 }
l[3] = { x = 51 y = 61 }
l[4] = { x = 51 y = 61 }
Сортировка по убыванию x:
l[0] = { x = 51 y = 61 }
l[1] = { x = 51 y = 61 }
l[2] = { x = 31 y = 32 }
l[3] = { x = 20 y = 32 }
l[4] = { x = 1 y = 2 }
```

Предлагаю сильным студентам разобраться с использованием других различных предикатов самостоятельно. Особенностью использования собственных объектов является необходимость перегрузки операций вывода. Другой особенностью класса **list** является автоматическое корректное удаление динамических объектов, созданных в программе и включенных в массив. В ЛР необходимо описать собственный класс объектов и продемонстрировать все возможности списков типа **list**.

3.45. Контейнерные классы списков в MFC

В библиотеке MFC (Microsoft Foundation Class) предусмотрено два основных класса для работы со списками: **CList** и **CObList**. Кроме этих классов предусмотрены классы для заданного типа включаемых объектов: **CStringList**, **CPtrList**. Класс **CList** является шаблонным, для создания объектов необходимо указать тип, а класс **CObList** не требует типизации, в него могут включаться любые объекты, наследованные от класса **CObject**. Методы первого и второго классов практически совпадают по названию и назначению. Рассмотрим первоначально класс **CList**. Для добавления в список используются различные методы: **AddHead, AddTail, InsertAfter, InsertBefore** и другие. Формально списки типа **CList** являются шаблонами вида:

```
template< class TYPE, class ARG_TYPE = const TYPE& >
class CList :    public CObject { …};
```

Настройка конкретного объекта списка требует указания типа (например, int). Пример описания и добавления в список переменных типа **int**.

```
CList<int , int> IntList; // Список объектов целого типа
for (i = 0 ; i < 3 ; i++ )
      IntList.AddTail(i);
 POSITION pos;
 cout << "Цикл вывода из списка !" << endl;
   for( pos = IntList.GetHeadPosition(); pos != NULL ; )
   {
            cout << "IntList =  " << IntList.GetAt(pos) << endl;
     IntList.GetNext( pos);    };
```

В результате получим:

```
Цикл вывода из списка !
IntList =  0
IntList =  1
IntList =  2
```

Методы **GetHeadPosition** и **GetNext** позволяют получить позицию начала списка и следующего элемента соответственно. Доступ к значению элемента списка производится с помощью метода **GetAt**. Метод **GetAt** позволяет получить значение элемента списка по позиции, позицию можно определить методом **Find**:

```
pos = IntList.Find(2, IntList.GetHeadPosition() );
          cout << "Для поиска элемента равного 2 =  " << IntList.GetAt(pos) << endl;
```

Получим в результате (начиная с 0):

```
Для поиска элемента равного 2 =  2
```

Метод **SetAt** позволяет заменить один элемент в списке на другой:

```
IntList.SetAt(pos, 33);
for( pos = IntList.GetHeadPosition(); pos != NULL ; )
{
        cout << "Новый IntList =  " << IntList.GetAt(pos) << endl;
   IntList.GetNext( pos);    };
```

Получим результат:

```
Новый IntList =  0
Новый IntList =  1
Новый IntList =  33
```

С помощью метода **InsertAfter и InsertBefore** можно добавить в середину после вычисленной ранее позиции и до нее:

```
Print_Clist( IntList );
IntList.InsertAfter(pos, 55);
IntList.InsertBefore(pos, 44);
Print_Clist( IntList );
```

Получим результат:

```
Список [0] = 0
Список [1] = 1
Список [2] = 44
Список [3] = 33
Список [4] = 55
```

Метод печати **Print_Clist** опишем в заголовочном файле следующим образом:

```
void Print_Clist(CList<int, int> & Cl)
{
        POSITION pos;
        int i = 0;
       if ( !Cl.IsEmpty())
  for( i = 0, pos = Cl.GetHeadPosition(); pos != NULL ; i++)
  {
     cout << "Список ["<< i << "] = " << Cl.GetAt(pos) << endl;
     Cl.GetNext( pos);
  }
       else
  cout << "Список пуст!"<< endl; };
```

Метод **IsEmpty** позволяет проверить пустой список. Метод **GetNext** вычисляет следующую позицию в списке. Вычисление позиции начала (**GetHeadPosition**) и конца (**GetTailPosition**) списка может быть выполнено так:

```
pos = IntList.GetHeadPosition();
        cout << "Для головы списка элемент =  " << IntList.GetAt(pos) << endl;
pos = IntList.GetTailPosition();
        cout << "Для хвоста списка элемент =  " << IntList.GetAt(pos) << endl;
```

Получим результат:

```
Для головы списка элемент =  0
Для хвоста списка элемент =  55
```

С помощью методов (**RemoveHead, RemoveTail, GetHead, GetTail, AddHead, AddTail**) можно удалять, добавлять элементы списка, получать значения из головы и хвоста списка:

```
IntList.RemoveHead();
IntList.RemoveTail();
//
```

```
   cout << "Для головы списка элемент (GetHead) =  " << IntList.GetHead() << endl;;
   cout << "Для головы списка элемент (GetTail) =  " << IntList.GetTail() << endl;;
//
IntList.AddHead( 10 );
IntList.AddTail( 50 );
Print_Clist( IntList );
```

Получим результат:

```
Для головы списка элемент (GetHead) =  1
Для головы списка элемент (GetTail) =  33
Список [0] = 10
Список [1] = 1
Список [2] = 44
Список [3] = 33
Список [4] = 50
```

Для подсчета числа элементов в списке используется метод **GetCount**:

```
cout << "Число элементов в списке (GetCount) =  " << IntList.GetCount() << endl;
```

Получим результат:

```
Число элементов в списке (GetCount) =  5
```

С помощью метода **FindIndex** можно найти элемент списка по номеру:

```
 pos = IntList.FindIndex(3 ); // Порядковый номер элемента в списке 3 (считаем от нуля)
cout << "Для поиска индекса 3 элемент =  " << IntList.GetAt(pos) << endl;
```

Получим результат:

```
Для поиска индекса 3 элемент =  33
```

С помощью метода **RemoveAll** можно удалить все элементы списка:

```
 IntList.RemoveAll();
 Print_Clist( IntList );
```

Получим результат:

```
Список пуст!
```

Для объектов собственного класса (в примере Point) почти все методы работают аналогично. Описание и заполнение списка из **Point**:

```
CList<Point , Point> PList;
for (i = 0 ; i < 3 ; i++ )
      PList.AddTail( *new Point(i,i));
cout << "Цикл вывода из списка Point !" << endl;
  for( pos = PList.GetHeadPosition(); pos != NULL ; )
  {
           cout << " PList =  " << PList.GetAt(pos) << endl;
     PList.GetNext( pos);    };
```

Получим результат:

```
Цикл вывода из списка Point !
PList =  { x = 0 y = 0 }
PList =  { x = 1 y = 1 }
PList =  { x = 2 y = 2 }
```

Метод **Find** (поиск по объекту) требует перегрузки операции сравнения (“= =”), конструктора копирования для класса **Point**. Например:

```
Point(const Point & p){ x = p.x ; y = p.y ;}; // Конструктор копирования
Point & operator=(const Point & p){ x = p.x ; y = p.y ; return *this;}; // Перегрузка присваивания
friend bool operator= =(const Point & p1 , const Point & p2) ;
...
// Перегрузка операции сравнения – равенства  для удаления по объекту
bool operator= =(const Point & p1 , const Point & p2)
```

```
{
return ( (p1.x= =p2.x) && (p1.y= =p2.y) );
};
```

Тогда для следующего фрагмента программы:

```
pos = PList.Find( Point(2,2));
        cout << "Find Point(2, 2) для PList =  " << PList.GetAt(pos) ;
```

Получим результат:

```
Find Point(2, 2) для PList =  { x = 2 y = 2 }
```

Для методов **FindIndex** и **RemoveAll** следующего фрагмента программы:

```
Print_PL(PList);
pos = PList.FindIndex( 1);
        cout << "FindIndex( 1) для PList =  " << PList.GetAt(pos) ;
PList.RemoveAll();
Print_PL(PList);
```

Получим результат:

```
PoList =  { x = 0 y = 0 }
PoList =  { x = 1 y = 1 }
PoList =  { x = 2 y = 2 }
FindIndex( 1) для PList =  { x = 1 y = 1 }
Список пуст!
```

Для класса списков **CObList** также все методы аналогичны. Существенное отличие заключается в том, что такой список поддерживает список указателей на объекты типа **CObject**, или точнее, объекты классов наследованных от **CObject**. В этом случае их описание эквивалентно такому:

```
CObList PoList;
for (i = 0 ; i < 3 ; i++ )
      PoList.AddTail( (CObject *)new Point(i,i));
cout << "Цикл вывода из списка CObList Point !" << endl;
  for( pos = PoList.GetHeadPosition(); pos != NULL ; )
  {        cout << "PList =  " << *((Point *)(PoList.GetAt(pos))) << endl;
     PoList.GetNext( pos);     };
```

Получим результат:

```
Цикл вывода из списка CObList Point !
PoList =  { x = 0 y = 0 }
PoList =  { x = 1 y = 1 }
PoList =  { x = 2 y = 2 }
```

Класс **Point** в этом случае должен быть наследником класса **CObject**. В этом случае все предыдущие фрагменты проверки работы методов для списков должны правильно работать:

```
class Point : public CObject { ... }; // Все остальное аналогично см. выше
```

Единственное отличие состоит в необходимости преобразования указателя к типу **Point** для автоматического вызова перегруженной операции вывода.

Если использовать вспомогательный метод печати, описанный так, как показано ниже, то при передаче ссылки, преобразования требуется преобразование типов от CObject к типу Point. Так как в список CObList включаются указатели, то необходимо также разыменование:

```
void Print_OL(CObList & Cl)
{ int i = 0;
 POSITION pos;
  if ( !Cl.IsEmpty())
  for( pos = Cl.GetHeadPosition(), i = 0 ; pos != NULL ; i++ )
```

```
  {
    cout << "Список CObList ["<< i <<"] =  " << *((Point * )(Cl.GetAt(pos))) ;
     Cl.GetNext( pos);    }
 else
  cout << "Список пуст!"<< endl; };
```

Тогда при вызове этой функции печати списка **PoList**:

```
Print_OL( PoList);
```

Получим результат:

```
Список CObList [0] =  { x = 10 y = 10 }
Список CObList [1] =  { x = 11 y = 11 }
Список CObList [2] =  { x = 12 y = 12 }
```

Кроме этого, необходимо учесть что добавление, выборка объектов должны выполняться с указанием объектов соответствующих типов.

3.46. Контейнерные классы списков в ATL

Контейнерный класс в библиотеке ATL называется **CAtlArray**. Он является шаблонным классом и имеет следующее формальное описание:

```
template<  typename E,  class ETraits = CElementTraits< E >
> class CAtlArray { … };
```

В этом классе все очень похоже на класс **CList** из MFC, но некоторые методы добавлены, например: **SwapElements** , **MoveToHead**, **MoveToTail** , **AddHeadList** и **AddTailList**. Все остальные методы и свойства совпадают с массивами библиотек MFC.

При подключении классов библиотеки **ATL** в главный модуль нужно добавить следующий заголовочный файл:

```
#include <atlcoll.h>
```

Для примера использования класса **CAtlArray** рассмотрим фрагмент программы с массивом целых чисел (**int**):

```
CAtlArray<int> atlMas;
...
for (i = 0 ; i < 5 ; i++ )
      atlMas.Add(i);
for (i = 0 ; i < atlMas.GetCount() ; i++ )
      cout << atlMas[i] << " "  ;
      cout << endl;
```

После выполнения этого фрагмента программы получим результат:

```
0  1  2  3  4
```

В случае включения в список объектов **Point**, для следующей программы:

```
CAtlList<Point> PAList; // Список точек
for (i = 0 ; i < 3 ; i++ )
      PAList.AddTail( *new Point(i+ 100,i+ 100));
cout << "Цикл вывода из списка CAtlList Point !" << endl;

 for( pos = PAList.GetHeadPosition(); pos != NULL ; )
 {
          cout << "PAList =  " << PAList.GetAt(pos) << endl;
     PAList.GetNext( pos);     };
```

Получим результат:

```
Цикл вывода из списка CAtlList Point !
PAList =  { x = 100 y = 100 }
PAList = { x = 101 y = 101 }
PAList =  { x = 102 y = 102 }
```

Если в предыдущих фрагментах программ заменить название списков, они будут работать. Ниже показано использование методов: **SwapElements** , **MoveToHead**, **MoveToTail** , **AddHeadList** и **AddTailList**. Первые три метода позволяют перемещать объекты внетри списка (функция печати **Print_AL** построена на основе предыдущего цикла):

```
Print_AL(PAList);
   cout << " SwapElements " << endl;
PAList.SwapElements( PAList.GetHeadPosition() , PAList.GetTailPosition());
  Print_AL(PAList);
    cout << " MoveToHead " << endl;
  PAList.MoveToHead(PAList.GetTailPosition());
  Print_AL(PAList);
    cout << " MoveToTail " << endl;
  PAList.MoveToTail(PAList.GetHeadPosition());
  Print_AL(PAList);
```

Получим результат:

```
SwapElements
Список CAtlList[0] =  { x = 102 y = 102 }
Список CAtlList[1] =  { x = 101 y = 101 }
Список CAtlList[2] =  { x = 100 y = 100 }
 MoveToHead
Список CAtlList[0] =  { x = 100 y = 100 }
Список CAtlList[1] =  { x = 102 y = 102 }
Список CAtlList[2] =  { x = 101 y = 101 }
 MoveToTail
Список CAtlList[0] =  { x = 102 y = 102 }
Список CAtlList[1] =  { x = 101 y = 101 }
Список CAtlList[2] =  { x = 100 y = 100 }
```

При слиянии целых списков в голову(**AddHeadList**) и хвост (**AddTailList**):

```
typedef CAtlList<Point> PAltList; // Объявление нового типа списка точек
PAltList PAList2; // пустой  список
     cout << " AddHeadList " << endl;
          PAList2.AddTail( *new Point(33,33)); // Добавим в PAList2 точку 33-33
   PAList.AddHeadList( &PAList2 ); // Слияние в голове

  Print_AL(PAList);
     cout << " AddTailList " << endl;
  PAList2.AddTail( *new Point(55,55)); // Добавим в PAList2 точку 55-55
          PAList.AddTailList( &PAList2 ); // Слияние в хвосте
  Print_AL(PAList);
```

Получим результат:

```
AddHeadList
Список CAtlList[0] =  { x = 33 y = 33 }
Список CAtlList[1] =  { x = 102 y = 102 }
Список CAtlList[2] =  { x = 101 y = 101 }
Список CAtlList[3] =  { x = 100 y = 100 }
 AddTailList
Список CAtlList[0] =  { x = 33 y = 33 }
Список CAtlList[1] =  { x = 102 y = 102 }
Список CAtlList[2] =  { x = 101 y = 101 }
Список CAtlList[3] =  { x = 100 y = 100 }
Список CAtlList[5] =  { x = 33 y = 33 }
Список CAtlList[4] =  { x = 55 y = 55 }
```

3.47. Наследование для списков контейнерных классов ДЗ

Изученные классы библиотек необходимо использовать в домашнем задании (ДЗ) по дисциплине и комплексной ЛР (ЛР№12-15). Ниже приводиться описание класса Home

для примера из методических указаний по ДЗ (дома **Home** и улицы - **Street**). Отмечу, что наследования от **Point**, в этом случае, приводиться для простоты восприятия примера, в ДЗ/КЛР необходимо предусмотреть прямое наследование от класса **CObject** с правильным его описанием. Ниже дано описание элементного класса ДЗ. Данный текст нужно разместить в заголовочном файле проекта.

```
class Home : public Point{
public:
string Name;
// … другие свойства //
Home(string S = ""): Point(){ Name = S;};
// … другие конструкторы  //
Home(int a , int b , string S = ""): Point(a , b){
        x =a; y = b;
        Name = S;};
// … //
friend ostream & operator <<( ostream &  out  ,  Home &  obj );
// …другие методы класса … //
};
// Перегурузка операции для вывода объекта Home
ostream & operator <<( ostream &  out  ,  Home &  obj )
{
out  <<"{ x = " << obj.x <<" y = " << obj.y << " Name = " << obj.Name <<" } " <<endl;
return out; };
```

Контейнерный класс ДЗ должен быть наследован от одного из классов библиотек (списки и массивы – см. варианты задания). Описание необходимо подключить в заголовочный файл проекта. Не забудьте подключить библиотеку (**atlcoll.h** – для этого класса):

```
class Street: public CAtlList<Home>
{ public:
string Name;
// …другие свойства класса улиц… //
Street(string S = "" ){ Name = S; };
// …другие методы класса улиц… // };
```

На основе описаний можно проверить простой текст программы для включения в список (улица) домов и распечатки содеожимого списка:

```
Street S ("Моя улица!");
Home H1("Первая!");
Home H2(1 , 2 , "Вторая!");
S.AddHead (H1);
S.AddHead (H2);
cout << endl << "Название улицы  = " << S.Name << endl;
for( pos = S.GetHeadPosition(), i = 0 ; pos != NULL ; i++)
  {
          cout << "Улица - " << S.Name <<" ["<< i << "] = " << S.GetAt(pos) << endl;
    S.GetNext( pos);     };
```

После выполнения программы должны получить:

```
Название улицы  = Моя улица!
Улица - Моя улица! [0] = { x = 1 y = 2 Name = Вторая! }
Улица - Моя улица! [1] = { x = 0 y = 0 Name = Первая! }
```

3.48. Сравнение списков и массивов

В 7-й и 8-й ЛР вы изучили контейнерные классы списков и массивов. В ДЗ вы должны научиться их использовать для построения собственной системы классов по вариантам задания. В вариантах предписано использование классов: **CArray**, **CList**, **CObAr-**

ray или **CObList**. Несомненно, использование типа зависит от содержания задания. Варианты используемых классов назначены для групп студентов. Считаю, что возможности классов данных библиотек должны подойти для ваших заданий. В действительности для каждой задачи использование массивов или имеет следующие существенные особенности:

– Массивы целесообразно использовать, если необходим прямой доступ к элементу контейнера по индексу.

– Списки предпочтительнее использовать, если содержимое, в первую очередь, последовательность элементов часто изменяется (сортировка).

– Массивы целесообразно использовать для больших объемах данных, заносимых в контейнер.

– Списки предпочтительнее использовать, если предполагается сливать или разделять контейнеры.

– Массивы целесообразно использовать для множеств упорядоченных по номеру.

В каждом конкретном случае необходимо оценить какой тип контейнеров целесообразно применить. Нельзя также забывать, что в библиотеках есть и другие типы контейнерных классов: очереди, стеки, множества, ассоциативные массивы, мультимножества и т.д. Эти классы предлагаются студентам для самостоятельной проработки.

4. Методические пояснения к темам ДЗ

Проработка предметной области задания

В данном разделе предоставлен материал, который должен помочь студенту приступить к проработке домашнего задания. Не думайте, что его можно безболезненно пропустить. Его освоение, несомненно, должно облегчить вам работу над проектом ДЗ, посвященному разработке системы классов по вашему варианту. Рассмотрим подготовку по отдельным шагам.

Студент получает индивидуально тему работы, которая кратко формулируется так: “Разработать систему классов для …”. Могут быть даны незначительные пояснения и уточнения к теме, хотя студент должен выполнить проработку (фактически проектирование), в основном, самостоятельно. Для этого нужно представить задачу или задачи, для которых система класса может быть использована. Это позволит сформулировать требования к классам: их свойства (характеристики) и поведение (методы класса). Далее выполняется словесное описание этих требований и программное описание классов по правилам языка программирования.

В каждом задании разрабатывается минимум один контейнерный класс и один элементный класс (см. ниже).

Первым шагом в выполнении КЛР (ДЗ) должно быть **осмысление** поставленной задачи, и, в частности, выделение новых типов объектов, для которых нужно разработать обобщенные описания объектов в виде классов на языке СИ++. Нужно определить свойства объектов каждого класса и его поведение. Другими словами, необходимо первоначально выделить основные данные для объектов класса и перечень методов/функций, которые эти данные изменяют. На первом этапе это делается на абстрактном уровне.

Для начала проектирования является важным определение понятия **предметной области**. Под предметной областью понимается совокупность понятий, объектов реального мира, их свойств, а также что с этими объектами можно делать в программе и делается в жизни. Для одной и той же системы классов может быть выделено много различных предметных областей, которые, в конечном счете, зависят от решаемой (поставленной) задачи автоматизации. Поэтому первым шагом нужно задать одну или несколько предметных областей, для которых будет разрабатываться система классов. Важно отметить, для уточнения понятия предметной области, что несправедливо другое утверждение: для одной предметной области может быть разработано несколько систем классов. Это объясняется тем, что систему классов, как раз и определяет содержание предметной области(!).

Пример. Будем рассматривать для пояснения пример разработки системы классов улиц и классов домов, которые могут располагаться на конкретных улицах. Пример реализации этого проекта приведен в общих методических указаниях по курсу. Возможны следующие задачи, для которых разрабатывается такая система классов (улицы и дома): для контроля оплаты жителями коммунальных услуг; для проведения выборов; для учета ремонта строений на улице и самой улицы; для ведения паспортного учета жителей (или прописки); для подсчета числа жителей на улице при планировании социальных услуг; для построения электронных карт города с улицами (детализация по домам), для оценки движения автомобилей по улице (пробок) и т.д. Из этого перечня задач видно, что от выбора конкретной задачи существенно зависят формализуемые свойства будущих объектов и набор операций (методов), которые над этими объектами предусматриваются. Определим, для примера, предметную область так: учет числа жителей на улице и оценка необходимости ремонта строений улицы.

Далее для каждого объекта предметной области, на содержательном уровне, нужно более детально задать перечень свойств (данных) и возможных операций над этими свойствами.

Пример. Например, выделим свойства для дома: номер дома, этажность дома, число жителей в доме, признак ремонта, тип строения и т.д. Операции для дома: установка числа жителей, изменение типа дома, установка или сброс признака необходимости ремонта дома. Для улицы можно выделить такие свойства: название улицы, тип улицы, число домов на улице, число жителей, признаки ремонта улицы и т.д.

Далее целесообразно проанализировать предметную область и поставленную задачу и определить необходимость дополнительных классов для ее реализации. Эти классы могут быть технологическими (вспомогательными) и содержательными. Технологические классы могут быть выделены, например, для реализации контейнеров элементных объектов (списки, массивы, множества и т.д.) и для выполнения других функций решения задачи. Эти классы, возможно, не имеют явных аналогов в предметной области и выделяются на основе опыта программиста. Дополнительные содержательные классы могут потребоваться для комплексного решения задачи. Например, Для нашего случая это могут быть: класс городов (при построении карт), класс планов ремонта улиц (в ремонтной задаче), классы бюджетов оплаты жилья (для контроля оплаты квартир) и т.д.

Более важным является определение перечня возможных операций над объектами. Не всегда удается сразу определить его полностью: он наращивается и видоизменяется в процессе проектирования и реализации программной системы.

Пример. Предположим тема задания следующая - разработать систему классов для описания объектов строений/домов улиц (элементные объекты) и объектов типа улица, содержащих в себе элементные объекты (дома). Улица - это контейнерный объект. Задачи, для которых разрабатывается система классов, заданы выше. Для домов целесообразно выделить свойства создания дома (построение), удаления дома (снос дома), изменения свойств дома (установка признаков необходимости ремонта, числа жителей и т.д.). Для улиц возможные операции: добавление и удаление домов, переименование улицы, поверка признака ремонта улицы и домов на улице. Предположим, что нужно предусмотреть операции для контейнерного объекта: добавления строений, переименования улиц, сноса строений, печати списка домов, добавления строений и т.д. Более детально это расшифровано ниже, включая и раскрытие понятий предметной области:

1. В нашем случае, на первом этапе мы должны выделить следующие понятия: дома/строения и улицы, как упорядоченной совокупности домов.
2. Понятие дома - объекта, возможно, будет обладать следующими свойствами: номер дома, этажность дома, число жителей дома, расположение на левой или правой стороне улицы, название дома, характеристика строения и т.д.
3. Понятие улицы – объекта, возможно, будет содержать следующие свойства: название улицы, перечень домов улицы, число проживающих на улице, число домов, район расположения, город улицы и т.д.
4. Поведение дома – набор методов, возможно, будет содержать следующие действия: создание дома, разрушение дома, чтение параметров, изменение номера дома и название, установка число проживающих, изменение этажности – перестройка дома, распечатка дома и т.д.
5. Поведение улицы – набор методов, возможно, будет содержать следующие действия: создание улицы, удаление улицы, добавление дома на улицу, удаление дома с улицы, новая нумерация домов на улице, распечатка домов улицы, изменение названия улицы, чтение параметров улицы, объединение двух улиц, деление улиц на две и т.д.
6. Уже сейчас, можно придумать название для классов этих объектов: дом **Home**, а улица – **Street** (Англоязычные названия используются для требований языков программирования. Они в переводе должны соответствовать содержанию объек-

та). Так как это описание классов на языке программирования, то мы должны давать названия латиницей. Лучше эти названия дать осмысленно.

7. В качестве технологических классов можно выделить классы: список (**List**) и элемент списка (**listElement**) и базовые абстрактные классы (по заданию ДЗ).

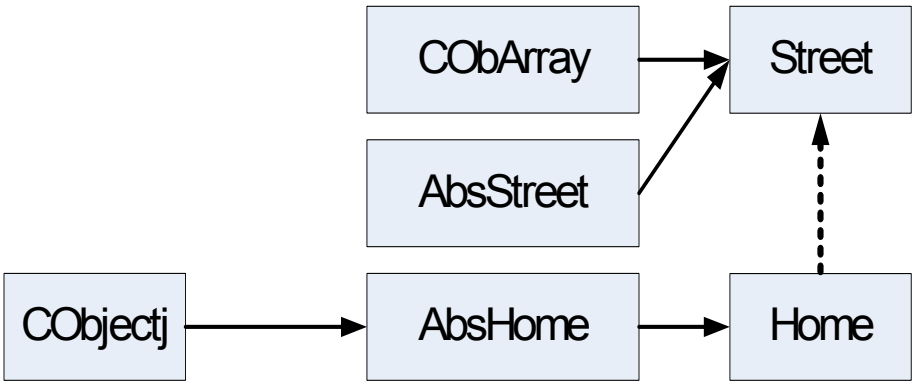
Еще раз уточню требования: в каждом варианте должно быть выделено, по крайней мере, **три содержательных класса**: один класс представляет собой описание конкретного материального или осмысленного объекта, а другой класс хранилище (список, массив, множество - контейнер) объектов первого класса с определенным порядком хранения, занесения и удаления объектов из него. Нужно придумать свойства и алгоритмы работы с этими объектами, продумать и вложить конкретный смысл в операции над ними. Над основными объектами и объектами контейнерных классов должны быть определены операции (объединения, сложения, удаления и т.д.).

Пример. В нашем случае материальный объект – дом/строение. Контейнерный объект улица, как совокупность домов. На первом этапе необходимо очень внимательно на понятийном уровне продумать смысл каждого свойства и каждого действия над выделенными типами объектов. Кроме того, как опытные разработчика, уже сейчас Вы должны "догадываться" во что в программе будут выливаться выделенные свойства и методы. К примеру, ниже приведены некоторые свойства классов с описание содержания:

1. Номер дома порядковый – целочисленная переменная (int **HomeNumber**), а операции по изменению номера дома и чтению этих параметров (**SetHomeNumber** и **GetHomeNumber**) и т.д. Придуманные обозначения желательно поместить в таблицу, в которой будут перечислены все понятия их свойства и методы работы с данными объекта, их названия и типы.
2. Номер дома символьный – символьная переменная (char ***Home_Number**), потребуется, так как дома часто имеют различные индексы в номере (Например: “Дом 5а-стр.2”).
3. Название улицы – символьный массив (char **NameStreet**[30]) или указатель на имя (char * **pNameStreet**). Для добавления дома на улицу придумаем метод (**AddHome**), а для удаления дома с улицы (**DeleteHome**). И т.д. Придуманные обозначения желательно поместить в таблицу, в которой будут перечислены понятия/методы, их названия и типы.

Одновременно с разработкой классов должна быть разработана проектная документация в виде: диаграмм классов, диаграмм объектов тестового примера, техническое описание классов, уже на этом этапе нужно думать о сдаче проекта, поэтому продумывается и программа методика испытаний системы классов. Выделяется и разрабатывается также документация для эксплуатации программного продукта (см. ниже).

Диаграмма классов в нашем случае может иметь следующий вид:



В ней представлены следующие стандартные классы: **CObject** , **CObArray** (по варианту группы). Собственные классы ДЗ: **AbstrHome , Home**, **Street, AbstrStreet.** Между классами показаны связи: сплошная стрелка – наследование (от базового к производному) и накопления пунктирная стрелка.

Таблицы с описанием классов домов могут выглядеть примерно так:

Класс домов - Home

Смысловое описание класса: класс домов используется для создания объектов типа дом для хранения информации о конкретном доме: номер, число жителей, число квартир, тип дома, требования к ремонту, число этажей.

Базовые классы класса **Home: AbstrHome**

Свойства класса **Home**

№ п/п	Содержание свойства	Тип данных и название	Примечание
1.	Символьный номер дома	char ***Home_Number**;	
2.	Номер дома числовой	int **iHome**;	
3.	Число этажей в доме	int **EtagCount**;	
4.	Число жителей в доме	int **MenCount** ;	
5.	Тип дома	HomeType **TypeHome** ;	
6.	Требуется ли ремонт дома	BOOL **HomeRemont** ;	
7.	Число квартир в доме	int **NumbApartament**;	

Методы класса **Home**

№ п/п	Содержание метода	Тип метода	Прототип метода	Примечание
1.	Создание дома без параметров	конструктор (конст.)	**Home**();	Построение пустого дома
2.	Создание дома по типу другого	конст.	**Home**(Home & H) ;	Построение
3.	Создание дома по типу другого	конст.	**Home**(Home * pH);	Построение
4.	Создание дома с симв. номером	конст.	**Home**(const char *HomName, const char *Number);	Построение
5.	Создание дома с симв. И числовымномерами	конст.	**Home**(const char *HomName, const char *Number, int Numb) ;	Построение
6.	Создание дома со всеми параметрами	конст.	**Home**(const char *HomName, const char *Number, int Numb, int Etag, int Men=0,HomeType Type = fast,int Apart=0)	Построение
7.	Удаление дома	Деструктор	~**Home**()	Снос дома
8.	Изменить имя дома	Метод	void **setName**(const char *HomName , const char *Number=NULL);	
9.	Получить имя дома	Метод	const char ***getName**();	
10.	Получить номер дома	Метод	const char ***getNumb**();	

11.	Получить числовой номер дома	Метод	int **getNo**();	
12.	Получить параметры дома	Метод	void **getParam**(int & iH, int & Etag ,int & Men ,HomeType & Type, int & Apart);	
13.	Установить параметры дома	Метод	void **setParam**(int iH, int Etag ,int Men ,HomeType Type, int Apart);	
14.	Установить все параметры дома	Метод	void **setAllParam**(const char *HomName , const char *Number, int iH, int Etag , int Men ,HomeType Type, int Apart , BOOL rem = false);	
15.	Оператор присваивания	Оператор =	Home **operator** =(Home & H);	Оператор должен быть перегружен, так как в объекте есть данные из динамической памяти
16.	Сложение двух домов	Оператор +	**friend** Home & operator +(Home & H1 , Home & H2);	Дружественная функция
Виртуальные методы класса **Home**				
17.	Получить тип класса	Виртуальный метод	virtual int **classType**();	
18.	Получить имя класса	Виртуальный метод	virtual char ***className**();	
19.	Распечатать параметры дома	Виртуальный метод	Virtual void **printOn** (ostream & out);	

Класс абстрактных домов - AbstrHome			
Смысловое описание класса: Абстрактный класс, используется в учебно-методическом аспекте для получения навыков использования абстрактных классов и виртуальных функций, а также для возможностей развития системы классов улиц и домов, например для поисковых систем.			
Базовые классы класса **AbstrHome:** : public **CObject**			
Свойства класса **AbstrHome**			
№ п/п	**Содержание свойства**	**Тип данных и название**	**Примечание**
1.	Символьное поле для поиска объекта дома	char ***name**;	
2.	Номер-ключ для поиска объекта дома	int **no**;	
Методы класса **AbstrHome**			
№ п/п	**Содержание метода**	**Тип метода**	**Прототип метода**	**Примечание**

20.	Создание объекта	конструктор (конст.)	**AbstrHome**();	
21.	Удаление объекта	Деструктор	~ **AbstrHome** ()	
Виртуальные методы класса **AbstrHome**				
22.	Получить тип класса	Виртуальный метод	virtual int **classType**()=0;	Чистая ВФ
23.	Получить имя класса	Виртуальный метод	virtual char ***className**() =0;	Чистая ВФ
24.	Распечатать параметры дома	Виртуальный метод	virtual void **printOn** (ostream & out) =0;	Чистая ВФ

Для класса улиц мы можем получить следующую таблицу свойств и методов:

Класс улиц - Street			
Смысловое описание класса улиц: класс улиц используется для создания объектов типа улица, включающая в себя дома. Дома на улице упорядочены. Учитываются свойства: название улицы, признаки ремонта, тип улицы, число домов на улице, список соседних улиц.			
Базовые классы класса **Street: AbstrStreet, , CObArray**			
Свойства класса **Street**			
№ п/п	**Содержание свойства**	**Тип данных и название**	**Примечание**
1.	Символьное название улицы	char *Name_Street	
2.	Число домов на улице	int Homes_num	
3.	Номер улицы	int NumberStreet	
4.	Признак ремонта домов на улице	BOOL Remont	
5.	Признак ремонта улицы	BOOL RemontStreet	
6.	Тип улицы	StreetType StrType	
7.	Список соседних улиц	Street * ListOfNear	Зарезервировано для развития
Методы класса **Street**			
№ п/п	**Содержание метода**	**Тип метода**	**Прототип метода**	**Примечание**
1.	Создать улицу без параметров	конструктор (конст.)	**Street**()	Проложить улицу
2.	Создать улицу с названием	конст.	**Street**(const char *sName)	Проложить улицу
3.	Создать улицу с названием и ключом для поиска	конст.	**Street**(const char *sNumbSearch, const char *sName)	Проложить улицу
4.	Создать улицу с номером	конст.	**Street**(int Num)	Проложить улицу
5.	Создать улицу с названием и номером	конст.	**Street**(const char *sName , int Num)	Проложить улицу

6.	Создать улицу на основе другой	конст	**Street**(Street & S)	Проложить улицу
7.	Удалить улицу	Деструктор	~**Street**()	Снести улицу
8.	Добавить дом на улицу	Метод	void **add**(Home *pH, TypeAddDel = tail , int Numb = 1 , TypeAddDel = createObj)	
9.	Удалить дом с улицы	Метод	void **del**(Home *pH , TypeAddDel = tail , int Numb = 1 , TypeAddDel = nodeleteObj)	
10.	Получить число домов	Метод	int **GetNumberHome**()	
11.	Получить число жителей	Метод	int **GetNumberMens**()	
12.	Получить число квартир	Метод	int **GetNumberApart**()	
13.	Получить название улицы	Метод	char * **GetNameStreet**()	
14.	Получить имя улицы для поиска	Метод	char * **GetKeyNameStreet**()	
15.	Получить номер улицы	Метод	int **GetNumbStreet**()	
16.	Получить номер улицы для поиска	Метод	int **GetKeyNumbStreet**()	
17.	Установить название улицы	Метод	void **SetNameStreet**(const char * NameStr)	
18.	Установить название ключ для поиска улицы	Метод	void **SetKeyNameStreet**(const char * sName)	
19.	Установить номер улицы	Метод	void **SetNumbStreet**(int n)	
20.	Установить номер ключ для поиска улицы	Метод	void **SetKeyNumbStreet**(int k)	
21.	Получить признак ремонта домов	Метод	BOOL **GetRemont**()	
22.	Получить признак ремонта улицы	Метод	BOOL **GetRemontStr**()	
23.	Установить признак ремонта улицы	Метод	void **SetRemontStr**(BOOL rS)	
24.	Получить тип улицы	Метод	StreetType **GetStreetType**()	
25.	Установить тип улицы	Метод	void **SetStreetType**(StreetType t)	
26.	Оператор присваивания	Оператор =	Street **operator** =(Street & S);	Оператор должен быть перегружен, так как в объекте есть данные из динамической памяти

27.	Сложить две улицы	- оператор	friend Street & **operator** +(Street & X , Street & Y);	Дружественная внешняя функция
Виртуальные методы класса **Street**				
28.	Получить тип класса	Виртуальный метод	virtual int **classType**();	
29.	Получить имя класса	Виртуальный метод	virtual char ***className**();	
30.	Распечатать параметры дома	Виртуальный метод	Virtual void **printOn** (ostream & out);	

Класс абстрактных улиц - AbstrStreet				
Смысловое описание класса: Абстрактный класс, используется в учебно-методическом аспекте для получения навыков использования абстрактных классов и виртуальных функций, а также для возможностей развития системы классов улиц и домов, например для поисковых систем.				
Базовые классы класса **AbstrStreet: нет**				
Свойства класса **AbstrStreet**				
№ п/п	**Содержание свойства**	**Тип данных и название**	**Примечание**	
3.	Символьное поле для поиска объекта дома	char ***name**;		
4.	Номер-ключ для поиска объекта дома	int **no**;		
Методы класса **AbstrStreet**				
№ п/п	**Содержание метода**	**Тип метода**	**Прототип метода**	**Примечание**
25.	Создание объекта	конструктор (конст.)	**AbstrStreet** ();	
26.	Удаление объекта	Деструктор	~ **AbstrStreet** ()	
Виртуальные методы класса **AbstrStreet**				
27.	Получить тип класса	Виртуальный метод	virtual int **classType**()=0;	Чистая ВФ
28.	Получить имя класса	Виртуальный метод	virtual char ***className**() =0;	Чистая ВФ
29.	Распечатать параметры дома	Виртуальный метод	virtual void **printOn** (ostream & out) =0;	Чистая ВФ

На втором этапе необходимо запрограммировать классы на языке С++. Отладить программы описания классов (обязательно использование отладчика), разработав для этого тестовый, демонстрационный пример, который иллюстрирует работу всех методов всех разработанных классов, а также использование всех свойств/атрибутов объектов всех разработанных классов.

Пример. Описание классов нашего примера может выглядеть так, мы поместим эти описания в модуль – DZ_Class.H:

```
// Абстрактный класс для дома
class AbstrHome: public CObject {
public:
        virtual int classType() = 0;
        virtual char *className() = 0;
        virtual void printOn(ostream &) = 0;
        AbstrHome(){ name = (char *)NULL;
             no = NULL;};
        ~AbstrHome(){
        if( name != (char *)NULL)
       delete [] name;
        };
        char *name;     // ? Резерв Поисковое имя
        int no;         // ? РезервНомер в списке
};
// Абстрактный класс для улицы
class AbstrStreet {
public:
        virtual int classType() = 0;
        virtual char *className() = 0;
        virtual void printOn(ostream &) = 0;
        AbstrStreet(){ name = (char *)NULL;
             no = NULL;};
        ~AbstrStreet(){
        if( name != (char *)NULL)
       delete [] name;
        };
        char *name;     // ? Резерв Поисковое имя
        int no;         // ? РезервНомер в списке
};
//////////////////////
//// Класс домов
//////////////////////
class Home: public AbstrHome  {
// Конструкторы
public:
        Home();
        Home( Home & H) ;
        Home( Home * pH);
        Home(const char *HomName, const char *Number);
        Home(const char *HomName, const char *Number, int  Numb) ;
        Home(const char *HomName, const char *Number, int  Numb,
                int Etag, int Men=0,HomeType Type = fast,int Apart=0);
// Деструктор
        ~Home();
 //Оператор присваивания для поддержки перегрузки "+"
        Home  operator =(Home &  H );
// Виртуальные методы
public:
        virtual int classType() {  return HomeClass; }
        virtual char *className() { return "Home"; }
        virtual void printOn(ostream & out);

// Методы класса Home
        void setName(const char *HomName , const char *Number=NULL);
        const char *getName() {         return (const char *)name; };
        const char *getNumb() {         return (const char *)Home_Number; };
        int  getNo() {    return no; };
        void getParam(int & iH, int & Etag ,int & Men ,HomeType & Type, int & Apart )
           {iH = iHome; Etag = EtagCount; Men = MenCount;
            Type = TypeHome; Apart = NumbApartament ; };
        void setParam(int  iH, int  Etag ,int  Men ,HomeType  Type, int  Apart )
```

```
            {  iHome= iH; EtagCount= Etag; MenCount= Men;
               TypeHome = Type; NumbApartament = Apart; };
          void setAllParam(const char *HomName , const char *Number, int  iH, int  Etag ,
        int  Men ,HomeType  Type, int  Apart , BOOL rem = false);
// Дружественная функция для перегрузки
    friend Home & operator +(Home & H1 , Home & H2);
// Свойства класса Home
public:
          char *Home_Number;   // Символьный номер дома
          int iHome;           // Номер дома числовой
          int EtagCount;        // Число этажей
          int MenCount ;        // Число жителей в доме
          HomeType TypeHome ;       // Тип дома
          BOOL HomeRemont ; // Требуется ли ремонт дома
          int NumbApartament;  // Число квартир// …
};
//////////////////////
// Класс улиц (Универсальное описание, включая закомментированный заголовок класса)
//////////////////////
// Разница для массивов и списков только здесь !!! (нужное для массивов перекомментировать)
 class Street: public AbstrStreet , public CObArray {
//class Street: public AbstrStreet , public CArray <CObject *, CObject * >{
///////// Для списков кроме перекомментирования нужно изменить библиотеку DZ_LIB.cpp
// class Street:  public AbstrStreet , public CObList {
// class Street:  public AbstrStreet , public CList<CObject *, CObject * > {
////////
public:
    Street();
          Street(const char *sName);
          Street(const char *sNumbSearch, const char *sName);
          Street(int Num);
          Street(const char *sName , int Num);
          Street(Street & S);
          ~Street() {    };
// Перегрузка присваивания
Street operator =(Street & S);
// Виртуальные методы
public:
          virtual int classType(){  return StreetClass;};
          virtual char *className() {return "Street";};
          virtual void printOn(ostream & out);
// Свойства класса Street
// Номер улицы – тип целый
public:
          char *Name_Street;
          int Homes_num;
          int NumberStreet;
          BOOL Remont;      // для всех домов
          BOOL RemontStreet;
          StreetType StrType;
          Street * ListOfNear;   // соседние улицы   - резервировано
// Методы класса Улиц - Street
          void add(Home *pH, TypeAddDel = tail , int Numb = 1 , TypeAddDel = createObj);
          void del( TypeAddDel = tail , int Numb = 1 , TypeAddDel = nodeleteObj);
 // Получить и установить параметры улицы name,, no, NumberStreet
          int GetNumberHome(){return  (int) GetCount()  ;};
          int GetNumberMens(); //
          char * GetNameStreet(){ return Name_Street;};
          char * GetKeyNameStreet(){ return name;};
          int GetNumbStreet(){ return NumberStreet;};
          int GetKeyNumbStreet(){ return no;};
          void SetNameStreet(const char * NameStr);
```

```
        void SetKeyNameStreet(const char * sName);
        void SetNumbStreet( int n ){ NumberStreet = n;  return;};
        void SetKeyNumbStreet( int k){ no = k; return;};
        int Street::GetNumberApart();
        BOOL GetRemont();
        BOOL GetRemontStr(){ return RemontStreet;};
        void SetRemontStr(BOOL rS){ RemontStreet = rS; return;};
        StreetType GetStreetType(){ return StrType;};
        void SetStreetType(StreetType t){ StrType= t; return;};
  // Дружественные функции
  friend Street & operator +( Street &  X ,  Street &  Y );
        };
```

В приложении к данным МУ приведены документы, в которых дано подробное техническое описание для нашего примера. Кроме этого, в приложении даны примеры листингов программ, которые могут быть настроены для наследования от списков и массивов.

5. Обязательные требования к ПО КЛР/ДЗ

Приведенные ниже требования являются обязательными для выполнения в рамках ДЗ/КЛР. Для сдачи заданий и получения хорошей отметки они все должны быть выполнены в полном объеме..

При выполнении задания должны быть обязательно учтены следующие требования:

1. Студент разрабатывает: (1) систему классов, (2) демонстрационный пример ее использования и (3) комплект документации на программный продукт индивидуально. Должна быть разработана диаграмма классов.

Пример. Система классов улиц и домов. Полный комплект документации, включая ТЗ и ПМИ (см. ниже). Демонстрационный пример для проверки работы системы классов на основе ПМИ (документ Программа и Методика Испытаний). При отладке программ классов может быть использован отдельный пример для отладки.

2. Студентом должны быть разработаны, по крайней мере, **три** новых класса, отличных от стандартных классов систем программирования и других библиотек классов. Имена классов должны быть понятными и легко запоминаться. Рекомендуется использовать венгерскую нотацию для именования классов, их свойств и методов.

Пример. Два класса дом и улица уже есть (**Home** и **Street** – см. выше). Класс **Home** должен быть наследован от **CObject,** а класс **Street** от классов список или массив (см. варианты групп). Третий класс может быть определен как абстрактный класс для строений или улиц, для того чтобы иметь возможность развития классов: строения могут быть не только жилыми домами, но и магазинами, учреждениями и т.д. Стандартные классы используются для контейнеров (массивы и списки), а также для формирования свойств виртуальности – базовый объект (Например, **CObject**).

3. Для классов объектов **явно** определяются и разрабатываются: **конструкторы** (не менее трех конструкторов для каждого класса) и **деструкторы** (для каждого класса). Кроме этого, перегружается оператор присваивания объектов, и создается конструктор копирования на основе объекта.

Пример. Конструкторы могут с первоначальным заданием параметров или нет, например, с заданием названия улицы, номера дома и т.д.(см. выше).

4. В каждом классе **помимо** конструкторов и деструкторов должно быть определено не менее **пяти** методов (функций членов класса), а также не менее **пяти** индивидуальных свойств разных типов (данных класса). Обязательно должны быть реализованы разные стандартные типы: int, double, char массив (строка). В дополнение к ним могут быть заданы типы стандартной библиотеки. Использование класса **string** (из STL, и MFC) не допускается.

Пример. Методы для улиц могут быть такими: добавление дома, удаление дома, распечатки домов улицы, нумерация домов, сложение улиц, деление улиц и т.д. Например, деление улицы на две может интерпретироваться так: первые N домов переносятся на "улицу Горького", а остальные на "Тверскую". Т.е., другими словами, из одного объекта мы получаем два других объекта с определенными свойствами. Разрабатывая такую операцию нужно продумать вопросы: где создаются новые объекты, как они заполняться, уничтожается ли исходный объект и т.д.

5. Должны быть продемонстрировано использование механизмов **наследования** классов для элементного и контейнерного класса. В каждом классе, как минимум, один класс должен быть базовым, а другой производным. Для класса контейнера (список или массив) наследование выполняется от стандартного контейнерного класса, заданного вариантом (список или массив) и собственного абстрактного класса. Таким образом должен быть продемонстрирован механизм множественного наследования.

Пример. Наследование, если это не определяются по-другому характером задачи желательно сделать также от абстрактного класса. Например, для дома или улицы.

6. Для элементного и контейнерного классов должны быть **перегружены** операции "+", в алгоритм функции перегрузки которой, закладывается, по возможности, оригинальный и конкретный смысл операции. Кроме этого должна быть перегружена операция из группового варианта.

Пример. Операция сложения двух домов, например, при выполнении новой нумерации улицы или дополнительных построек домов (дом 2 строение 3.).

7. В одном из своих классов, по выбору, должны быть реализованы скрытые методы и данные, не менее 3-х (доступ **private**). Доступ к этим данным и методам должен быть выполнен только с помощью открытого метода класса, что должно быть продемонстрировано на примерах в демонстрационном примере.

Пример. Доступ к вычисляемому признаку ремонта домов улицы (скрытому - **private**) и метода расчета признака ремонта может быть выполнен отдельным методом.

8. В одном классе (элементном или контейнерном) на выбор определяется, как минимум, одна **чистая** **виртуальная** функция (это предопределяет использование одного абстрактного класса) и демонстрируется ее переопределение в других классах, для этого может быть создан дополнительный класс (типа **Object**). Функция затем переопределяется и демонстрируется е применение в проекте ДЗ.

Пример. Чистые виртуальные функции задаются в абстрактном классе и переопределяются в производных классах. Например, функция печати объекта на **cout** (например - **printOn**).

9. Один из разрабатываемых классов должен быть **контейнерного** типа (массив, список, множество и т.д.), в нем должны быть определены операции добавления, удаления, распечатки из контейнера и др. Должны учитываются дополнительные требования к данному классу, которые выделены в групповых и индивидуальных вариантах задания. Контейнерные свойства класса наследуются от стандартных контейнерных классов по варианту группы (список, массив). Операции с собственным контейнерным классом реализуются на основе методов и свойств базового класса. Допускается реализация собственных дополнительных базовых контейнерных классов (списки, массивы и т.д.), при этом они реализуются без наследования от классов стандартных библиотек (**д.т.**).

Пример. В контейнерном классе улица предусматриваются операции добавления домов, их удаления, печати всех домов улицы и т.д. Например, для определения числа объектов в контейнере для класса **CArray** может быть использован метод **GetCount**. Возможна собственная реализация классов список или массив.

10. В контейнерном классе**,** в зависимости от варианта, задается как минимум **одна** перегружаемая операция (см. обязательная операция, обозначенная знаком по варианту группы). Операция должна иметь внятный смысл и быть продемонстрирована на примере.

Пример. В нашем случае лучше определить операцию сложения двух улиц или операцию добавления дома на улицу. В первом случае это может выглядеть так:

Оператор присваивания сложения улиц: **Street3 = Street1+ Street2;**
а во втором простое выражение присваивания: **Street1+Home1;**

11. Отладка системы классов может быть выполнена с помощью специального **отладочного** примера, созданного в произвольной форме, однако предпочтительней является вариант с переключателем и отдельными блоками для проверки методов и свойств классов (конструкторы, методы, операции). Для проведения испытаний ПО ДЗ студенты разрабатывают отдельный документ ПМИ (см. ниже). Возможно, для проведения испытаний потребуется разработать и отдельный (не отладочный проект) тестовый пример и описать в документе ОТП – описание тестового примера (см. ниже).

Пример. В приложении к данным МУ можно посмотреть такой пример (Закомментированная функция **mainDEBUG**()).

12. Для сдачи работы, должен быть разработан, описан в документе и представлен специальный **демонстрационный** (тестовый) пример, иллюстрирующий работу всех возможностей разработанной системы классов – всех пунктов ТЗ (свойств и методов/функций членов). Он является основой для демонстрации и сдачи домашнего задания (в соответствии с пунктами ТЗ и ПИМ – программой и методикой испытаний). Все пункты ТЗ (раздел 5.1 технического задания) должны проверяться в тестовом примере. Можно использовать переключатель для обеспечения выборочной проверки пунктов ТЗ.

Пример. В демонстрационном примере (**mainMETOD()**) должны быть показано использование всех классов, на основе определения объектов и работу всех методов этих классов. Выполнение всех пунктов ТЗ. Он строится на основе консольного меню и переключателя. Например, для классов улиц: создание улиц статическое и динамическое через операцию new, уничтожение улиц через операцию **delete**, добавление домов на улицу, их удаление, распечатку домов, слияние улиц и т.д.

13. Описания классов и их методы должны быть вынесены в отдельные файлы – модули проекта (*.cpp, *.h или *.hpp). При этом необходимо исключить повторное включение описаний в исходный текст. Проектирование модульной структуры и распределение описаний по модулям выполняется в самом начале работы над проектом домашнего задания.

Пример. Например, с помощью переменных этапа компиляции (define __STREET_H):

```
#if !defined( __STREET_H )
#include <Street.h>
#endif //
```

14. Должны быть предусмотрены специальные методы очистки (**Clear**) и распечатки контейнерного объекта (**PrintOn**).

Пример. Смотрите примеры в приложении.

15. Предусмотреть в классе методы для записи содержимого вашего контейнера в двоичный файл и восстановления содержимого контейнера из файла. Данный пункт должен быть в ТЗ для ДЗ, в разделе 5.1 и МПИ.

Пример. Запись в файл всех домов с улицы (**StreetToFile** и **StreetFtomFile**). Пока этих функций в примерах приложения нет.

16. После компиляции программ и создания исполнимого модуля (build) не должно наблюдаться любых предупреждений в окне фиксации ошибок. При необходимости преподаватель при демонстрации проекта может попросить скомпилировать модули и пересоздать проект.

Примечание ВНИМАНИЕ: Контейнерные классы в домашнем задании должны быть реализованы на основе стандартных классов типа списки и массивы из библиотек (по вариантам групп). Задания, реализованные на основе простых стандартных массивов объектов (например, для домов - **Home**[20]), приниматься к защите не будут.

Требования к реализации контейнерных классов

В каждом задании на КЛР/ДЗ необходимо создать минимум один собственный контейнерный класс. Контейнерные классы (см. выше) – это такие классы, на основе которых создаются объекты, позволяющие включать в себя другие объекты или, по другому, хранить в себе. Это отношение накопления между классами. Контейнерные классы могут быть упорядоченными (массивы и списки) и неупорядоченными (множества). Для реализации контейнерных классов студенты могут выбрать один из подходов:

- Использование стандартных (библиотечных) классов типа список или массив, а собственный контейнерный класс наследовать от них. Все основные методы базового класса для выполнения операций наследуются.
- Создание собственных списков (или массивов) в контейнерном классе задания (например, в классе улиц) и реализация в них всех операций для работы над списками (массивами). Это менее универсальный вариант, но более трудоемкий способ реализации проекта задания (вариант из дополнительных требований). Он может быть полезен в учебном плане для подготовки программиста.
- Создание собственных отдельных базовых классов типа список и массив, с дальнейшим наследованием ваших контейнерных классов задания от них. Более универсальный вариант, но еще более трудоемкий случай (вариант также из дополнительных требований). Он может быть аналогично полезен в учебном плане для тренировки программиста.

Если студенты выбирают первый способ, то они должны использовать классы MS VS, которые записаны для отдельных требований групп (см. ниже).

Дополнительные требования для сильных студентов

Для сильных студентов предлагается более сложный вариант задания. Во-первых, можно выбрать в таблице вариантов более сложный вариант задания (д.т.), согласовав выбранную тему с преподавателем.

Во-вторых, в стандартном задании можно выполнить следующие дополнительные требования:

1. Создать собственные классы для двунаправленных списков (или специальных динамических массивов) и использовать их для наследования в ваших тематических контейнерных классах.

2. Выполнить оригинальную перегрузку операций ввода/вывода в поток(**<< , >>**). Это нужно сделать и для элементного и контейнерного классов задания.

3. Использовать бинарные файлы для хранения информации из контейнерных классов. Для этого предусмотреть специальные конструкторы и деструкторы. Предусматривается загрузка и выгрузка из файлов из/в ОП. Возможна выборка объектов по номеру в контейнере и/или по условиям, наложенным на атрибуты элементных объектов.

4. Использовать динамически создаваемые объекты, включаемые в контейнер, с контролем создания и удаления объектов и их составляющих (например, для строковых

полей класса). Предусмотреть возможность удаления и сохранения объектов после выборки из контейнера.

Студент может выбрать одно или несколько дополнительных требований и указать их содержание на титульных листах документов ДЗ. Выполнение дополнительных требований может быть полезно в учебном плане для тренировки программиста, а также учитывается при подведении итогов семестра по данной дисциплине (автоматы или значение отметки).

Примечание: Для утверждения дополнительных требований и дополнительных тем нужно(31-36) их нужно выбрать и согласовать с преподавателем в начале семестра (до 5-й недели семестра). На титульных листах документов должен быть отмечен факт выполнения дополнительных требований и их содержания.

6. Требования к документации КЛР/ДЗ

Общие требования к документации по КЛР/ДЗ

Документация по КЛР выполняется студентом индивидуально и должна включать стандартный набор документации на программное изделие (состав и требования к документации представлены в данных методических указаниях и на сайте дисциплины **www.sergebolshakov.ru** – см. мой сайт раздел ДЗ и вариантов ДЗ). Для упрощения работы студентам предоставлены образцы исполнения документов (в приложении и на сайте), а также шаблоны для построения документов (на сайте). Документы могут быть выполнены в виде отдельных материалов и в виде отдельного документа, включающего в себя в качестве отдельных разделов все документы ДЗ. Для проведения сдачи, защиты и приемно-сдаточных испытаний материалы ДЗ должны быть предоставлены в электронном виде и распечатаны. Следующие документы (их восемь документов) должны быть разработаны в рамках КЛР/ДЗ и практикума по дисциплине:

1. **Техническое задание (ТЗ)** на систему классов, в котором важными разделами являются технические требования (ТТ – 5-й раздел ТЗ), в них включаются собственные функциональные требования к разрабатываемой системе классов. Это функциональные требования важны при разработке ТЗ (п.п. 5.1) и определяют последующие этапы проработки ПО.

2. **Описание применения** системы классов, поясняющее, как и для чего может использоваться Ваша система классов. Условно, можно сказать, что это краткая информация для потребителя – реклама продукта.

3. **Техническое описание** системы классов и ПО(см. также выше и ниже). Оно должно включать: диаграммы классов, подробное описание каждого класса. Описание классов, их свойств и методов должно быть сделано на понятийном уровне (в отдельном разделе) и на техническом уровне, желательно в виде таблиц. Обязательно, кроме подробного, вначале должно быть выделено краткое смысловое описание каждого класса – одной фразой. Например: "класс изображений служить для описания объектов – изображений на экране компьютера в форматах …", и т.д. Для важных алгоритмов разработанного программного обеспечения должны быть построены блок-схемы процедур, у нас методов классов.

4. **Руководство пользователя** системы классов, а не тестового примера, где должно быть описано, как и для чего, программист может использовать вашу систему классов. Должны быть даны простые примеры описания объектов и их использования, продемонстрировано применение всех методов, разработанных классов и применение свойств объектов. Описание классов должно быть сделано на понятийном уровне (в отдельном разделе) и на техническом уровне, желательно в виде таблиц. Могут быть приведены ссылки на документ техническое описание (ТО).

5. **Руководство системного программиста** должно определять действия системного программиста для установки (или удаления) системы классов, проверки правильности установки, требуемые ресурсы для работы системы классов и т.д.

6. **Описание тестового примера** (включает подробное описание тестового примера, включая и диаграмму объектов примера и особенности применения разработанных классов.). Обратите внимание, что описание тестового примера это отдельный документ и

не является программой и методикой испытаний (ПМИ). Однако, в ПМИ (см. ниже) можно ссылаться на пункты описания тестового примера.

7. **Листинг программ** системы классов (распечатка всех файлов, включая и заголовочные) и листинги программ тестового примера и листинг результатов его работы.

8. **Программа методика испытаний** (ПМИ) на основе тестового примера и ТЗ.

Примечание: Шаблоны и образцы оформляемых документов приведены ниже. Кроме того вам необходимо познакомиться с общими требованиями к разработке данных документов, которые также приведены в методических указаниях ЛР12-ЛР15 и специальных методических указаниях по оформлению отдельных документов (см. ниже и на сайте). Кроме этого шаблоны документов представлены на сайте в отдельном архиве.

Замечание: Документы должны разрабатываться в соответствии с ГОСТ, должны содержать всю необходимую информацию, написаны ясно и конкретно, соответствовать основной цели написания каждого конкретного документа. Документы могут разрабатываются отдельно. В этом случае каждый документ сопровождается титульным листом, и должен иметь свое оглавление. Вариант задания (номер и название) нужно указать на каждом титульном листе документа. Можно объединить все документы в один документ, при этом должно быть сформировано общее оглавление.

Более подробно содержание разрабатываемых документов изложено в п.9 данных МУ и в лабораторных работах по разработке документации (ЛР 12-15 смотрите на сайте в разделах ЛР и ДЗ). Кроме этого в этом документе (МУ ДЗ) и на сайте вы найдете образцы и шаблоны для подготовки документов по ДЗ, а также методические указания по оформлению всех документов.

Особые требования к ТО и ОТП по КЛР/ДЗ

Техническое описание разработанных или используемых классов должно включать:

1. Описание структуры классов (диаграммы связей классов, включая отношения наследования, использования, включения и дружественные связи);
2. Для каждого класса должно быть дано общее описание одной фразой !!! Это понятийное описание классов.
3. Модульную структуру программы, поясняющую, в каких исходных модулях описаны классы и их методы. Оформляется в виде декомпозиционной диаграммы (FDD).
4. Описание данных всех классов (в виде таблицы с указанием имен, понятий и типов данных);
5. Описание функций членов классов (методов) и их семантика (назначение, название функции и пример применения, желательно поместить в таблицу, см. выше);

При описании тестового примера нужно учесть и включить в документацию:

1. Описание объектов тестового примера (диаграмма объектов тестового примера, включается с техническое описание и описание тестового примера).
2. Описание тестового примера для демонстрации работоспособности системы классов.
3. Описание процесса отладки классов, тестового примера и ошибок при отладке программы (помещается в техническое описание в учебных целях)
4. Распечатки всех модулей системы классов и тестового примера: *.c , *.h , *.cpp , *.hpp (и др.). С комментариями по тексту (в комментариях идентификация разработчика как в ЛР. Комментариев не должно быть много, в противном случае они будут трактоваться как шпаргалки!).

7. Порядок выполнения КЛР/ДЗ

В данном разделе изложен возможный вариант последовательности шагов для выполнения задания.

1. Взять тему домашнего задания в соответствии с номером студента по журналу группы в текущем семестре. Если студент решил использовать дополнительные требования или усложненные темы заданий (Помеченные - д.т. - дополнительные требования), то он должен в начале семестра согласовать это вопрос с преподавателем. Темы вариантов даны в отдельном разделе данных методических указаний. А последние (актуальные) варианты тем ДЗ размещаются на сайте дисциплины (www.sergebolshakov.ru) их нужно уточнить для каждого текущего семестра.

Пример. В тема в нашем случае домашнего задания “Разработать систему классов улиц и домов … Предусмотреть …”.

2. Проработать предметную область домашнего задания. Нужно выделить объекты и классы для реализации ДЗ. Проработать их свойства и методы. Более подробно это изложено выше, в разделе “Методические пояснения к темам ДЗ ”.

Пример. В тема домашнего задания “Разработать систему классов улиц и домов … Предусмотреть …”. Класс домов (Home) и класс улиц (Street). Классы создаются для подсчета числа жителей и оценки ремонта всех домов улиц.

3. Выполнить модульную декомпозицию, определив назначение и содержание каждого из модулей, и размещение описание классов, их методов в модулях. Дать названия модулям нужного типа (*.cpp, *.h или *.hpp).

Пример. В нашем случае выделим четыре модуля проекта (для удобства основные функции main объединены в одном из модулей, а с помощью двойного комментирования включается необходимый модуль для отладки или для сдачи):

Модуль - **DZ_Array.cpp (DZ_List.cpp)** - модули для **отладки** системы классов (раскомментирован заголовок функции **main** совместно комментированием заголовка функции **mainDEBUG**, а также комментируется заголовок функции **main** совместно раскомментированием **mainMETOD.**)

Модуль - **DZ_Array.cpp (DZ_List.cpp)** - модуль для **сдачи** системы классов на основе ПМИ (раскомментирован заголовок функции **main** совместно комментированием заголовка функции **mainMETOD**, а также комментируется заголовок функции **main** совместно раскомментированием **mainDEBUG.**).

Модуль - **DZ_LIB.cpp** -содержит **методы** для всех классов проекта
Модуль - **DZ_Class.h** - содержит для **описания** классов проекта
Модуль - **DZ.h** - модуль **описания** общих данных проекта

4. По данным методическим указания, в рамках выполняемых ЛР по дисциплине, литературе и лекциям изучить теоретические вопросы ДЗ. В частности вопросы: описания классов, наследования в классах, перегрузки операций в классах, требования к документированию программных проектов.

5. Изучить требования к ДЗ, требования к документации ДЗ и испытаниям ПО ДЗ. Познакомиться с примерами оформления документов ДЗ, примерами программ.

Пример. Примеры оформления документов ДЗ, а также примеры программ проекта ДЗ размещены в приложении к данному документу и на сайте дисциплины (www.sergebolshakov.ru).

6. Разработать техническое задание (ТЗ) для своего проекта и утвердить его у преподавателя. Особое внимание нужно обратить на раздел ТЗ с функциональными требованиями (раздел 5.1). Пункты ТЗ должны быть сформулированы на содержательном уровне, не должны использоваться конкретные названия: классов, их свойств и их методов.

Пример. Пример оформления документа ТЗ размещены в приложении к данному документу и на сайте дисциплины (www.sergebolshakov.ru). Изложение ТЗ должно соответствовать требованиям и стандартам. Раздел 5.1 может выглядеть так:

“Система классов улиц и домов должна обеспечивать выполнение следующих функции для работы с этими объектами:

5.1.1. Создание контейнерных объектов для улиц города, в которых могут размещаться дома этой улицы.

5.1.2. Создание объектов для домов улицы, с параметрами и без параметров.

…

”

7. Разработать предварительный вариант описания применения для своего проекта. После завершения проекта необходимо этот документ просмотреть и изменить при необходимости.

Пример. Пример оформления документа ТЗ размещены в приложении к данному документу и на сайте дисциплины (www.sergebolshakov.ru).

8. Создать диаграмму классов, описания классов в виде таблицы и текста на языке программирования, и разместить эти проработки предварительно в документ ТО.

Пример. Пример составления диаграммы классов, их обобщенного описания приведены в разделе данных методических указаний “ Методические пояснения к темам ДЗ ”. В нашем случае разработаны два основных и два дополнительных абстрактных класса для наследования:

Абстрактный класс для Дома - **AbstrHome**
Абстрактный класс для Улицы - **AbstrStreet**
Класс Дом - **Home**
Класс Улица - **Street**

В проекте используются следующие базовые классы стандартных библиотек в зависимости от варианта:

CObject – для объектов включаемых в контейнеры.
CObArray – для наследования контейнера по вариантам групп
CArray -для наследования контейнера по вариантам групп
CObList -для наследования контейнера по вариантам групп
CList -для наследования контейнера по вариантам групп

9. Подробно описать классы, их свойства и методы, разместив их в нужном модуле, для отладки свойств и методов классов нужно использовать специальный пример для отладки.

- конструкторы классов (5-ть разных)
- методы классов (нужно перечислить базовые методы!!!)
- примеры использования классов

Пример. Пример составления диаграммы классов, их подробного описания приведены в разделе данных методических указаний “ Методические пояснения к темам ДЗ ”. Пример описания классов, отладочного примера даны в приложении к данному докумен-

ту и на сайте дисциплины (www.sergebolshakov.ru). Это модули: DZ_Class.h, DZ_Array.cpp (DZ_List.cpp) и DZ_LIB.cpp. Меню для отладки системы классов может иметь вид:

Меню тестового примера системы классов улиц.

1.Конструкторы Home

2.Методы Home

3.Операции Home

4.Конструкторы Street

5.Методы Street

6.Операции Street

0.Выход

Выберете номер режима (0-6):

10. Разработать и создать документ техническое описание (ТО). При необходимости вносить изменения в другие документы и программы. Существенные изменения в ТЗ требуют дополнительного согласования с преподавателем.

Пример. Пример оформления документа ТО размещены в приложении к данному документу и на сайте дисциплины (www.sergebolshakov.ru).

11. Выполнить детальную отладку и тестирования каждого из пунктов рассмотренного выше меню (соответственно пунктов ТЗ). С этой целью нужно активно использовать отладчик. Выполнить проверку каждого конструктора, метода и свойства класса. Результаты отладки ввести в консольное окно для проверки. При необходимости, по мере отладки, допускается вносить изменения в документы ТО и ОП. Существенные изменения в ТЗ требуют дополнительного согласования с преподавателем.

12. Разработать и создать документ программу и методику испытаний (ПМИ). При необходимости, по мере отладки и разработки других документов, допускается вносить изменения в документы ТО и ОП. Существенные изменения в ТЗ требуют дополнительного согласования с преподавателем. Разработать одновременно тестовый пример и документ описание тестового примера (ОТП).

Пример. Примеры оформления документов ПМИ и ОТП размещены в приложении к данному документу и на сайте дисциплины (www.sergebolshakov.ru). Для этого нужно создать специальный тестовый пример для проверки функциональных требований ТЗ. Текстовое меню тестового примера для ПМИ может иметь следующий вид (меню здесь представлено сокращенно):

1. ТЗ - 5.1.1 Создание улиц с домами
2. ТЗ - 5.1.2 Создание объектов для домов улицы
3. ТЗ - 5.1.3 Создание объектов для домов улицы на основе других
4. ТЗ - 5.1.4 Учет свойств дома(см. ТЗ)
5. ТЗ - 5.1.5 Задание и получение характеристик дома
6. ТЗ - 5.1.6 Сложение двух домов
7. ТЗ - 5.1.7 Перегрузить оператор присваивания для домов

...

18. ТЗ - 5.1.18 Перегрузка оператора присваивания для улиц

0.Выход

Выберете номер режима (0-18):

13. Отладить программную систему на основе тестового примера, проверив детально все пункты меню функциональных требований ТЗ. Результаты отладки тестового примера поместить в документ ПМИ.

14. Разработать и создать документы Руководство пользователя (РП) и руководство системного программиста (РСП). При необходимости, по мере отладки и разработки других документов, допускается вносить изменения в документы ТО и ОП. Все документы должны быть согласованы между собой. Существенные изменения в ТЗ требуют дополнительного согласования с преподавателем.

Пример. Примеры оформления документов РП и РСП размещены в приложении к данному документу и на сайте дисциплины (www.sergebolshakov.ru).

15. Сформировать отдельный документ исходный текст программы (ИТ или листинги программ – ЛП – ИТ/ЛП), поместив в него исходные тексты всех модулей проекта. Исходные тексты модулей (листинги) размещаются в данном документе обязательно в последней редакции. Так что не нужно торопиться распечатывать программы заранее, а сделать это нужно после оформления и проверки всех остальных документов. В каждом документе должно быть представлено оглавление, а каждый исходный модуль должен начинаться с отдельной страницы. При необходимости дать комментарии в документе.

16. Проверить состав разработанной документации: ТЗ, ОП, ТО, ОТП, ПМИ, РП, РСП, ИТ/ЛП. Всего должно быть восемь документов. Для каждого из документов есть образец его оформления и шаблон для его оперативного построения. Шаблоны и образцы для оформления на сайте дисциплины в разделе ДЗ и ЛР (www.sergebolshakov.ru).

17. Проверить самостоятельно проведение приемно-сдаточных испытаний на основе ПМИ и , возможно, провести такие испытания совместно с другими студентами группы предварительно без преподавателя. Проверка у самих себя!

18. Выполнить у преподавателя контроль правильности оформления и состава документации КЛП/ДЗ. Провести приемно-сдаточные испытания по пунктам ТЗ на основе ПМИ (Допускается случайная выборочная проверка отдельным по пунктам ТЗ на основе меню тестового примера и выборочная проверка отдельных документов).

8. Варианты КЛР/ДЗ

Ниже приведены примерные варианты тем заданий для выполнения комплексной лабораторной работы (КЛР) или домашнего задания (ДЗ) по дисциплине “Программирование на основе классов и шаблонов” на **2015/2016** учебный год. Номера тем заданий соответствуют номерам студента по журналу группы на текущий семестр. Номера вариантов индивидуальных заданий меняются ежегодно, поэтому нужно обязательно уточнить тему заданий и требования к групповому варианту можно в специальном документе, доступном студентам на сайте дисциплины: www.sergebolshakov.ru.

Номер варианта	Тема задания
1.	Класс строк переменной длинны и класс массивов этих строк, предусмотреть операции над строками и массивами строк (объединение и разбиения массивов) и т.д.
2.	Класс перечней (списков) программных продуктов инсталлированных на разных ЭВМ и класс программных продуктов с атрибутами. Предусмотреть возможность объединения списков, классификации продуктов и замены версий программных продуктов в списке и т.д.
3.	Класс публикаций (статей) в разных журналах и класс каталогов этих публикаций. Предусмотреть операции объединения каталогов и сортировки по авторам, журналам, году издания и названиям статей и т.д.
4.	Класс факультетов и класс кафедр факультета, предусмотреть операции слияния/разделения кафедр и факультетов, перевода кафедр с одного факультета в другой.
5.	Класс слов и класс предложений. Предусмотреть операции объединения предложений, добавления, удаления и замены слов на заданных позициях и т.д.
6.	Класс файлов и класс каталогов файлов, для хранения файлов. Предусмотреть поиск по имени файла. Предусмотреть операции перемещения файлов, их добавления и удаления, поиска, переименования, сравнения и объединения каталогов и т.д.
7.	Класс строительных бригады и класс работников бригад. Предусмотреть объединение и разделение бригад, добавление и удаление работников.
8.	Класс словарей (русско-английских) и класс гнезд словарей (отдельных ячеек словарей), предусмотреть поиск и сортировку. Предусмотреть объединение словарей, сортировкку и разделение по алфавитному принципу и т.д.
9.	Класс множеств абстрактных объектов и класс элементов множеств (отдельных объектов), предусмотреть весь известный из теории набор операций над множествами: объединение, вычитание, пересечение и др. Включить и операции из курса дискретной математики.
10.	Класс групп студентов и класс студентов, предусмотреть операции слияния/разделения групп и редактирование всех видов объектов и т.д.
11.	Класс аннотированных ссылок на ИНТЕРНЕТ ресурсы и класс Интернет страниц (перечней), на которых они расположены, с возможностью поиска, предусмотреть операции объединения и очистки страниц и т.д.

Номер варианта	Тема задания
12.	Класс программных продуктов и класс компакт дисков, где они записаны/расположены. Предусмотреть возможность слияния компакт дисков, их корректной очистки и разбиения на каталоги (каталог – возможно, это новый класс и объекты) и т.д.
13.	Класс стеллажей книг в библиотеке и класс книг на стелажах. Предусмотреть объединение разделение стеллажей, добавление и удаление книг, проверку нахождения книг и т.д.
14.	Класс таблиц баз данных и класс записей в таблице (в виде реляционной таблицы). Предусмотреть операции выборки записей по заданным признакам. Предусмотреть класс связей между таблицами (д.т.).
15.	Класс наборов компьютеров в локальной сети, класс компьютеров и класс связей (д.т.) между компьютерами в сети. Предусмотреть операции объединения и разделения наборов компьютеров сети, добавления, удаления и замены компьютеров в списках, изменения соединений, структуры сети и т.д.
16.	Класс изображений и класс слайдов презентаций (последовательностей слайдов). Класс люъектов на слайдах (д.т.). Изображения не выводятся на экран, а распечатываются в поток с названиями и характеристиками. Предусмотреть возможность включения вложенных презентаций. Слайды не воспроизводить!
17.	Класс отделов/подразделений сотрудников и класс сотрудников, предусмотреть операции приема на работу, увольнения, изменение окладов и должностей. Операции объединения и разделения отделов, подсчета фонда зарплаты и средних характеристик отдела: стаж, возраст зарплата и т.д..
18.	Класс документов и класс папок (каталогов) с множеством документов, предусмотреть операции слияния папок, добавления и удаления из них документов.
19.	Класс перечня (списка) блоков занятой и свободной оперативной памяти (ОП) и класс отдельных блоков (ОП), предусмотреть сборку мусора объединения списков памяти, объединения и разделения блоков памяти и т.д.
20.	Класс комплектующих компьютера и класс их наборов (перечней комплектующих – фактически сборки компьютера). Предусмотреть операции замены комплектующих по типу и множественность некоторых элементов компьютеров (например, диски). Предусмотреть очистку набора и изменение типа комплектующих элементов.
21.	Классы заголовков текста в документах и классы документов. Предусмотреть возможности объединения документов. Документы на экран не выводятся! В заголовках должна быть указана позиция расположения в документах, они могут иметь разные уровни (не менее 3-х уровней заголовков). Предусмотреть распечатку оглавления на основе заголовков в перечне.
22.	Класс карточек учета товаров и класс картотек с разными характеристиками товаров: по названию, фирмы их продающих и т.д.. Учесть даты их поступления. Предусмотреть операции добавления карточек, слияния фирм и их очистки.
23.	Класс информации о различных событиях и класс наборов (перечней событий). Обеспечить поиск события в тексте названия или содержания, интервала дат и типу событий, которых должно быть не менее пяти. Объединение списков событий и т.д.

Номер варианта	Тема задания
24.	Класс очередей задач к ресурсам операционной системы и класс задач. Задача рассматривается как элемент с различными атрибутами (требуемая ОП, время счета и т.д.). Очередь организуется в режиме FIFO. Предусмотреть операции объединения очередей и изменения последовательности расположения задач в зависимости от характеристик и т.д.
25.	Класс звуковых сигналов (нот) и класс мелодий из них, предусмотреть операции слияния мелодий и включения новых звуков в них (необязательно проигрывать мелодии и звуки, достаточно давать их буквенное обозначение). Звуки не воспроизводить!
26.	Класс мультимедиа объектов (звуки, рисунки, текст и т.д.) и класс их хранилищ (типа "холста", для построения рисунков). Объекты не воспроизводятся, нужно предусмотреть стандартный вывод названий и свойств. Предусмотреть текстовую распечатку перечня объектов на холсте. Предусмотреть операции объединения и расслоения холстов по введенным признакам (как в фотошопе), например номер слоя.
27.	Класс окон интерфейса и класс управляющих элементов в этих окнах (кнопки, поля и т.д.). Сами окна и элементы интерфейса не выводятся на экран. Возможна распечатка списка элементов и окон. Предусмотреть операции объединения окон и проверки корректности расположения элементов (отсутствие наложения друг на друга, нахождения в рамке окна и т.д.).
28.	Класс векторов и класс их списков (трехмерные вектора), предусмотреть операции над векторами и списками векторов (сложение и т.д. из математики).
29.	Класс складов и класс товаров, которые хранятся на данном складе. Предусмотреть завоз новых товаров, продажу товаров. Объединение складов и выделение филиалов складов с товарами.
30.	Класс списка литературы и класс элементов списка литературы (название, год, авторы и т.д.). Предусмотреть возможность объединения списков, их сортировки по разным критериям и распечатки и т.д.
31. (Д.т.)	Класс двунаправленных списков и их произвольных элементов: строк, дат и чисел (нужно использовать указатели). Операции со списками их сортировки и их объединения. Создать собственные классы, а не копировать из VS и т.д. Необходимо придумать содержательный контейнерный класс для наследования от двунаправленных списков и элементный класс, объекты которого включаются в него.
32. (Д.т.)	Классы таблиц (строки и столбцы – произвольной размера) и их содержимого в виде текстовых ячеек. Операции объединения текста в ячейках, добавление текста, обмена ячейками и т.д.
33. (Д.т.)	Классы векторов и классы массивов векторов (трехмерные вектора) предусмотреть операции над векторами и массивами векторов (сложение и т.д. из математики)
34. (Д.т.)	Классы массив целых чисел и классы их массивов (массивов указателей для массивов), предусмотреть операции над массивами и числами: сложения, вычитания и объединения.
35. (Д.т.)	Класс баз данных и класс таблиц баз данных (реляционная таблица). Предусмотреть класс записей в таблицах.
36. (Д.т.)	Класс баз данных и класс таблиц баз данных (реляционная таблица). Предусмотреть класс связей между таблицами.

Примечание: Если в задании требование указано с признаком дополнительных требований, то оно не обязательное (д.т.). При его выполнении на титульном листе ТЗ на систему в документах на ДЗ/КЛР, то этот факт должен быть отмечен.

Примечание: Для того чтобы задания каждого студента были индивидуальными введены специальные требования для каждой из групп (2015/2016 учебный год):

гр. ИУ5-21 (СУЦ -2-й курс) - обязательным является дополнительная перегрузка операции вывода (<<) в стандартный поток **cout** для элементного класса. (Пример. Класс домов. Смотрите документ требований к КЛР). Для описания контейнерного класса необходимо использовать базовый класс **CObList**.

гр. ИУ5-22 – обязательным требованием является использование дополнительной, перегруженной операций над объектами контейнерного класса. Операция выполняет удаление объектов для контейнерного класса (использовать знак "-" - минус). (Пример. Удаление дома с улицы). Для описания контейнерного класса необходимо использовать базовый шаблонный класс **CList**.

гр. ИУ5-23 - обязательным является дополнительная перегрузка операции ввода (>>) из стандартного потока **cin** для элементного класса. (Пример. Класс домов). Для описания контейнерного класса необходимо использовать базовый шаблонный класс **CArray**.

гр. ИУ5-24 - обязательным является перегрузка дополнительная операции вычитания ("-") двух контейнерных классов для получения нового объекта контейнерного класса, содержащего только те элементы, которые отсутствуют во втором контейнером класса, но присутствуют в первом. (Пример - создание новой улицы). Для описания контейнерного класса необходимо использовать базовый класс **CObArray**.

Основные требования к вариантам ДЗ изложены в разделе общих требований выше, которые размещены в специальном документе “**Методические указания для выполнения ДЗ/КЛРпо дисциплине Программирование на основе классов и шаблонов кафедры ИУ5** “ 2015г. (см. на сайте)

Дополнительные требования для сильных студентов - обязательным является перегрузка операций ввода/вывода файловых потоков (ввода -istream и вывода - ostream). применительно к контейнерному и элементному классам (см. Пример. Классы домов и улиц). Для реализации контейнерного класса необходимо создать свой базовый контейнерный класс типа (массив и список: Array или List). По результатам разработки, кроме исполнимого модуля тестового примера, нужно создать библиотеку (*.LIB) объектных модулей для своих классов и подключить ее в проект, продемонстрировав ее использование. Для утверждения дополнительных требований нужно: выбрать и согласовать с преподавателем вариант с дополнительными требованиями (31-36).

Примечание 2: Номера вариантов заданий специально продублировал в отдельной колонке, так как они, почему-то не у всех высвечиваются!!!

9. Программная документация на КЛР/ДЗ

9.1. Программная документация

Любой программный проект должен сопровождаться разработкой комплекса программной документации. Время, которое затрачивается на разработку такой документации соизмеримо со временем самой программной разработки (не менее 50% времени всего проектирования). Иногда это время превышает время программирования и отладки проекта. Некоторые документы разрабатываются до начала разработки. Например, техническое задание (ТЗ) на программную разработку. Без создания грамотного ТЗ проект вообще нереализуем или, скорее всего, не будет успешным. Кроме того, без хорошо разработанного технического задания, невозможно успешно предъявить и сдать работу заказчику, а также, успешно закрыть работу по проекту (включая и получения оплаты за выполненную работу). Программисты (или специальные разработчики документации) разрабатывая документы, по сути, являются “техническими писателями”, а освоение таких умений является неотъемлемой частью получаемых навыков в рамках подготовки по нашей специальности.

9.2. Программная документация, разрабатываемая в домашнем задании

Для получения основных навыков в разработке технической документации на программное обеспечение выбраны главные документы из значительного множества документов, требуемых в ГОСТ [8]. Этот комплект документов содержит следующие документы:

- Техническое задание (**ТЗ**);
- Описание применения программного продукта (ОП);
- Техническое описание программного продукта (ТО);
- Руководство пользователя (РП);
- Руководство системного программиста (РСП);
- Программа и методика испытаний на основе тестового примера (ПМИ);
- Описание тестового примера (ОПТ);
- Исходные тексты программ системы классов и тестового примера (ИТ);

Последний документ (“Описание тестового примера”) может не выделяться отдельно и входить составной частью других документов: “ Руководство пользователя ”, ” Техническое описание программного продукта ”, “ Программа и методика испытаний ”. В рамках домашнего задания и КЛР студенты должны разработать весь комплект документации на программный продукт, разрабатываемый по их варианту. По каждому документу у вас предусмотрена отдельная лабораторная работа, которую вы можете выполнить и заранее в рамках часов, отводимых под самостоятельных работу студентов.

9.3. Особенности и принципы разработки программной документации (ПД)

При разработке ПД нужно учесть следующие обстоятельства:

- Кто из участников процесса разрабатывает данный документ ПД.
- Для кого разрабатывается данный документ ПД
- Каков стиль изложения данного документа ПД
- Каково содержание данного документа ПД
- На каком этапе проектирования разрабатывается данный документ ПД
- Каковы требования к данному документу.

Четкие ответы на перечисленные вопросы должны быть у каждого разработчика данного документа. В процессе разработки документации на программный продукт могут участвовать следующие группы и разновидности специалистов:

- Руководители проекта.
- Разработчики программисты (или исполнители проекта).
- Заказчики проекта.
- Пользователи разрабатываемого программного обеспечения (ПО).
- Системные программисты, устанавливающие ПО для пользователя.
- Технические писатели (разработчики технической документации).

Эти специалисты могут выполнять следующие роли по отношению к документам комплекта ПД: писать или разрабатывать документы, контролировать содержание документов, согласовывать (подписывать) документы и утверждать документы (подписи). В нашем случае, в игровом обучающем режиме студенты могут выполнять разные роли: руководителей, писателей и разработчиков, а преподаватели – роль заказчиков проекта.

В каждой ЛР из цикла КЛР мы рассмотрим все эти особенности, применительно к каждому типу рассматриваемого документа. Здесь выделим общие особенности стиля изложения и разработки ПД:

- Документы должны содержать информацию, имеющую отношение только к данному проекту и данному документу.
- Документы должны быть написаны на техническом языке и не должны содержать неопределенности и неоднозначности.
- Документы должны быть написаны короткими предложениями, и они должны включать минимум распространенных предложений (с придаточными предложениями).
- Стиль документа должен соответствовать стилю документа данного типа.
- Документы не должны содержать “воды” и отступлений от содержания. Никакой “лирики”.
- Все пункты должны быть пронумерованы (разделы, позиции и т.д.) и допускать ссылки из других документов. Не допускается перечисления в виде маркированных списков (**bullets**).
- При ссылках на другие документы должны указываться прямые ссылки, содержащие: полное название документа и позицию (пункт), на который производиться ссылка.
- В пределах проекта устанавливается требования к оформлению документов: редактор текста, шрифт, кегль шрифта, интервал, способы форматирования текста, размеры страниц, содержание колонтитулов, способы рисования иллюстраций документа (рисунков), обязательная нумерация страниц в документе.

9.4. Требования к оформлению ПД в ДЗ

В нашем случае требования к оформлению следующие:

- Текстовый редактор - **MS WORD** (не Open Office!),
- Шрифт - **Times New Roman**.
- Кегль шрифта - **12**,
- Интервал между строками - **одинарный**,
- Способы форматирования текста – **по ширине**,
- Размеры страниц – **А4 (верх - 2 , низ - 2 , слева - 3 , справа - 2)**,

– Содержание колонтитулов (**вариант, группа, ФИО студента)**,

– Способы рисования иллюстраций документа (Предпочтительнее **MS Visio** можно **MS WORD**),

– Нумерация страниц – **центр – верх**.

Данные требования необходимо соблюдать при разработке всех документов домашнего задания.

9.5. Техническое задание

9.5.1. Документ техническое задание (ТЗ) и его назначение

Документ техническое задание (ТЗ) разрабатывается на ранних этапах проектирования программного продукта и содержит требования к разработке: функции, сроки, требования к продукту в различных аспектах и порядок их реализации. Этот документ является важнейшим и иногда разрабатывается до заключения финансового договора с заказчиком. Считается, что после утверждения ТЗ (его подписания сторонами), оно может быть изменено только по обоюдному согласию заказчика и разработчика и только в определенном порядке.

Документ ТЗ разрабатывается совместно заказчиком и разработчиками (руководителем проекта и исполнителями) и утверждается (подписывается). После этого, не выполнение отдельных пунктов ТЗ может являться основанием для расторжения договора (по закону). Более того, утверждается, что невыполнение ТЗ преследуется по закону.

Примечание. Студенты часто стараются разработать ТЗ после завершения разработки. Это не правильно, так как заказчик (преподаватель у нас) может ТЗ не утвердить это очень рискованно.

ТЗ является, кроме того, основой для самой разработки и основой для разработки других документов.

9.5.2. Стиль изложения в ТЗ

Стиль изложения документа ТЗ – декларативный (предписывающий): все предложения должны соответствовать предписывающему стилю ("программа должна обеспечивать …" или "в процедуре необходимо обеспечить" или "система должна выполнять" и т.д.). Кроме этого в полной мере при разработке ТЗ нужно учесть общие требования к стилю документов.

9.5.3. Требования к ТЗ

Документ ТЗ является важнейшим документом для проектирования и реализации проектов. В частности, проведение приемно-сдаточных испытаний программного продукта основывается (см. пояснения к ЛР по “ Программе и методике испытаний ”). Поэтому все пункты требования должны быть четкими и не допускать разных трактовок. Они должны быть такими, чтобы их можно было бы проверить. В противном случае их из ТЗ нужно исключить, обоснованно доказав это заказчику.

При разработке ТЗ реально сталкиваются разнонаправленные интересы заказчиков и разработчиков (фактически возникают противоречия, которые нужно разрешить). Заказчики хотят с минимальными затратами сделать как можно больше, а разработчики хотят сделать минимум также со своими минимальными затратами трудоемкости.

9.5.4. Разработка ТЗ

Разработка ТЗ должна производиться на ранних стадиях проектирования программных продуктов. Иногда, перед разработкой ТЗ выполняют работы по пробному макетированию отдельных частей программы или ее в целом. Макетирование технических решений позволяет проверить правильность формулировок отдельных пунктов ТЗ, определить трудоемкость и сроки выполнения работ. Результаты макетирования можно предъявить заказчику для подтверждения реальности сроков выполнения работ и пояснения согласованности и наглядности функций, возложенных на программный продукт. В макете может быть продемонстрирован интерфейс пользователя с будущей системой.

Для сложных программных разработок (программных систем) предварительно могут проводиться дополнительные научно - исследовательские работы (НИР) или опытно-конструкторские разработки (ОКР). Результатами таких работ должны быть предложения к разработке ТЗ на систему или сам документ ТЗ.

В любом случае, в тексте ТЗ должны содержаться такие формулировки требований, которые не ограничивают разработчика и предельно понятны заказчику. Так, например, в нашем случае не следует в разделе функциональных требований указывать имена классов, методов и т.д. В процессе разработки они могут измениться, но это фактически может привести к формальному невыполнению пунктов ТЗ, на реализации которых заказчик, возможно, будет настаивать. С другой стороны, может быть внесена такая формулировка, которая понимается заказчиком и разработчиком по-разному. Это не правильно. Таких текстов в документе следует избегать.

9.5.5. Содержание ТЗ

Содержание документа ТЗ по пунктам приведено ниже. В образце документа ТЗ приведено ТЗ для варианта улиц и домов, описанного в общем пособии по курсу [3]. В шаблоне документа ТЗ (см. в конце данных методических указаний) даны методические указания к написанию и приспособлению документа применительно к конкретному варианту студента. Содержание документа ТЗ:

1. НАИМЕНОВАНИЕ
2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ
3. НАЗНАЧЕНИЕ РАЗРАБОТКИ
4. ИСПОЛНИТЕЛЬ
5. ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ
5.1. Требования к функциональным характеристикам
5.2. Требования к программному обеспечению
5.3. Требования к условиям эксплуатации
5.4. Требования к информационному обеспечению
5.5. Требования к надежности
5.6. Требования к составу и характеристикам технических средств
5.7. Требования к программной совместимости
6. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ
6.1. Разрабатываемые технические и эксплуатационные документы
7. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ
8. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ
8.1. Сроки выполнения отдельных этапов работ
9. ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ ЗАДАНИЯ
9.1. Требования к сдаче и условия приемки
10. ДОПОЛНИТЕЛЬНЫЕ ТРЕБОВАНИЯ

Дополнительно к методическим указаниям шаблона выделим главные требования к основным разделам ТЗ (на выполнение этих требований будет обращаться повышенное внимание при предъявлении программ и защите документов):

В пункте 3 (НАЗНАЧЕНИЕ РАЗРАБОТКИ) очень кратко (2-3 предложения) формулируется назначение разработки: какие преимущества возникают при его применении, какие новые возможности появляются у пользователя, какие операции автоматизируются при использовании программного продукта, для решения каких задач может использоваться данная система классов.

В раздел 5.1 на должны быть включены основные функции системы классов, включающие способы создания всех типов объектов, использования методов классов. Требования к функциям должны формулироваться на содержательном уровне (а не программистском). Эти основные функции должны быть размещены в начале раздела 5.1. Кроме того, должны быть отражены и дополнительные функции.

Все позиции в ТЗ (для ссылок на них) должны быть пронумерованы с помощью многоуровневой нумерации (5.1.1, 5.1.2 и т.д.). В разделах п.7 и п.10 информация может отсутствовать при выполнении ДЗ/КЛР. В раздел 5 могут быть добавлены новые позиции по соглашению с заказчиком (Например, функции защиты информации, безопасность использования программного и технического обеспечения и т.д.).

9.5.6. Раздел ТЗ – 5. Технические требования (общее)

В этом разделе ТЗ перечисляются технические требования к программному изделию. Эти требования должны быть выполнены при реализации программного проекта. Формулировки должны быть однозначными. Требования должны быть выполнимы разработчиком и проверяемы заказчиком при проведении приемно – сдаточных испытаниях. В главный раздел могут выноситься требования общего характера, которые должны быть выполнены (Смотрите шаблон и образец ТЗ).

9.5.7. Раздел ТЗ – 5.1 Требования к функциональным характеристикам

Данный раздел – один из основных документа ТЗ. Он разрабатывается на основе изучения поставленной задачи. Набор функциональных требований (или функциональных характеристик) должен быть полным. Требования не должны противоречить друг другу и ясно восприниматься заказчиком – пользователем. (Смотрите шаблон и образец ТЗ).

9.5.8. Разделы ТЗ – Другие технические требования

Разделы ТЗ, связанные с другими техническими требованиями, задают требования к реализации и эксплуатации программного продукта. Их правильная разработка очень важна. Здесь определяются средства реализации проекта, условия его эксплуатации, форматы данных и файлов, требования к надежности ПО. В нашем случае (ДЗ/КЛР) эти требования, за исключением некоторых дополнительных вариантов, могут быть одинаковыми для всех заданий. Однако их внимательно необходимо прочитать, проверить и осмыслить. (Смотрите шаблон и образец ТЗ).

9.5.9. Раздел ТЗ – требования к документации

В данном разделе определяется перечень документации, разрабатываемый в программном проекте. В нашем случае данный перечень уже задан, если необходимо для реа-

лизации проекта в него можно внести дополнения (см. дополнительные варианты). В курсовых работах и проектах в данный перечень добавляются листы: диаграммы классов и объектов, блок-схемы алгоритмов, структуры данных и т.д. В нашем случае диаграммы и блок-схемы можно оформить в виде рисунков, вставляемых в соответствующие документы. (Смотрите шаблон и образец ТЗ).

9.5.10. Раздел ТЗ – этапы сроки выполнения

Этапы и сроки выполнения ДЗ определяются учебным планом и длительностью семестра. В нашем случае приведенный список этапов и сроков можно сохранить в неизменном виде. Нужно только учесть, что формально ЛР № 12 выполняется в конце семестра, а работу над заданием необходимо начать как можно раньше. Поэтому желательно познакомиться с темой задания и разработать ТЗ в сроки, определенные в образце и шаблоне документа. (Смотрите шаблон и образец ТЗ).

9.5.11. Раздел ТЗ – порядок приема и контроля

В данном разделе ТЗ формулируются требования к приемке программного продукта. Определяются принципы проверки и его этапы. Проверка и сдача программного проекта может выполняться на основе различных принципов, и даже в несколько этапов:

– Проверка принципов построения продукта на макете и ТЗ (ранние стадии).

– Проверка работоспособности продукта – целенаправленно проверяется наличие ошибок в ПО (завершение разработки).

– Проверка документации на соответствие требованиям (завершение разработки).

– Выборочная проверка работоспособности на основе ТЗ и его выполнения, на основе программы и методики испытаний (ПМИ). По завершению разработки.

– Полная комплексная проверка работоспособности и документации на программный продукт. Такая проверка выполняется на основе специального согласованного документа ПМИ.

– Выборочная проверка работоспособности и документации на программный продукт. Такая проверка выполняется на основе специального согласованного документа ПМИ.

В нашем случае (сдача ДЗ студентами) мы выполняем выборочную проверку программ и документации на основе специального документа ПМИ. Для проведения проверки разрабатывается специальный тестовый пример, который позволяет выборочно проверить каждый пункт ТЗ из раздела 5.1 – требования к функциональным характеристикам или, по-другому, функциональным требованиям. Разработка этого примера и документа ПМИ будет рассмотрена в специальной лабораторной работе по курсу. (Смотрите шаблон и образец ТЗ).

9.5.12. Порядок изменения и корректировки ТЗ

Документ техническое задание может быть, в порядке исключения, изменен или откорректирован даже после его согласования и утверждения (подписи заказчиком и разработчиком). Для этого разрабатывается отдельный документ с названием: “Корректировка ТЗ”. Далее, после всех согласований документ может быть заново переоформлен либо документ корректировки просто прикладывается к основному ТЗ. Эти действия (переоформление или прикладывание) выполняются по обоюдному согласию разработчика и заказчика.

9.6. Описание применения

9.6.1. Документ описание применения (ОП) ПО и его назначение

Документ “Описание применения” (ОП) программного продукта (данные документ ориентируется на потенциального пользователя ПП и потенциального покупателя ПП). Документ должен отвечать на вопросы, как и при каких условиях можно использовать ПП. Условно можно считать, что данный документ имеет рекламное назначение: прочитав документ, пользователь должен оценить возможности программного продукта и определиться с его приобретением.

9.6.2. Стиль изложения ОП

Стиль изложения документа “Описание применения” должен быть описательным ("программа имеет возможности …", "программа работает в среде WINDOWS …", "Система позволяет автоматизировать …", "Объекты, создаваемые на основе классов, обеспечивают построение …" и т.д.). Текст в данном документе должен быть написан ясно, без сложных для понимания технических терминов, простым языком. В случае необходимости в нем оформляются наглядные таблицы и диаграммы. Понимание данного материала должно быть доступно не только для специалистов, а и для менеджеров организаций, для которых предлагается использовать программный продукт.

9.6.3. Разработка ОП

Разработка документа ОП выполняется в конце всей разработки программного продукта специалистами по рекламе и по внедрению программного обеспечения. Техническая информация передается им специалистами разработчиками. Основой для разработки ОП является комплект всей документации на ПО.

9.6.4. Содержание ОП

Содержание документа ОП по пунктам приведено ниже. В образце документа ОП приведено ОП для варианта улиц и домов, описанного в общем пособии по курсу [3]. В шаблоне документа ОП (см. в конце данных методических указаний) даны методические указания к написанию и приспособлению документа шаблона применительно к конкретному варианту студента.

Содержание документа ОП:

1. НАЗНАЧЕНИЕ ПРОГРАММНОГО ПРОДУКТА
2. ВОЗМОЖНОСТИ ПРОГРАММНОГО ПРОДУКТА
   2.1. Общие сведения о программном продукте
   2.2. Диаграмма классов программного продукта
3. ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ПРОГРАММНОГО ПРОДУКТА
4. УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММНОГО ПРОДУКТА
   4.1. Требования к составу и параметрам технических средств
   4.2. Требования к информационной совместимости
   4.3. Требования к программному обеспечению
   4.4. Требования к условиям эксплуатации

4.5. Требования к маркировке и упаковке
4.6. Требования к хранению
5. ОБЩИЕ ХАРАКТЕРИСТИКИ ПРОГРАММНОГО ПРОДУКТА

9.6.5. Требования к ОП

Главные требования к основным разделам документа “Описания применения” (на выполнение этих требований будет обращаться повышенное внимание при сдаче ПО):

В п.1 (“**НАЗНАЧЕНИЕ ПРОГРАММНОГО ПРОДУКТА**”) дается более подробное, чем в ТЗ, назначение разрабатываемого программного продукта (ПО). Стиль должен быть простым и ориентированным на неподготовленного пользователя, как в рекламных объявлениях и рекламной информации. Документ может быть ориентирован на менеджеров и руководящих работников.

В п.2 (“**ВОЗМОЖНОСТИ ПРОГРАММНОГО ПРОДУКТА**”) приводятся все положительные свойства ПО, с пояснением, если нужно, условий функционирования и выгод его использования по сравнению с подобными программными продуктами. В нашем случае приводится описание классов на содержательном уровне и основных возможностей их применения. Не допускается использовать латинские названия методов, данных и перегруженных операций. Здесь, в нашем случае, приводиться диаграмма классов разработанного программного продукта.

В п.3 (“**ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ПРОГРАММНОГО ПРОДУКТА**”) приводятся основные технические характеристики ПО. Эти характеристики можно оформить, для наглядности, в виде таблицы. Здесь отображаются: размеры используемой оперативной памяти, задействованные прерывания, ограничения на возможности создания объектов на основе системы классов (для нашего случая), объемы в ОП и т.д., все, что характеризует и отличает данный программный продукт. Можно давать сравнительные характеристики по сравнению с аналогичными программными продуктами.

В п.4 (“**УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММНОГО ПРОДУКТА**”) приводятся все ограничения и требования к применению ПО. В частности в п.4.2 может уточняться тип кодировки символов, типы форматы файлов для хранения информации и т.д., в зависимости от назначения и конструкции ПО. В этом разделе в обобщенном виде приводятся данные из ТЗ, связанные с условиями эксплуатации, хранением, транспортировкой ПО.

В п.5 (“**ОБЩИЕ ХАРАКТЕРИСТИКИ ПРОГРАММНОГО ПРОДУКТА**”) сведены общие и самые существенные характеристики ПО из предыдущих разделов. Эту информацию желательно тоже оформить в виде таблицы.

9.7. Техническое описание

9.7.1. Документ техническое описание (ТО) ПО и его назначение

Документ “Техническое описание” (ТО) программного продукта. Это фактически материалы технического проектирования ПП. В данном документе описывается конструкция изделия (модульный состав и связи). Здесь должно быть описано на техническом языке: как устроен ПП, как он сделан, из каких частей состоит, какие связи есть, какие внешние данные использует и т.д. Описываются все данные и методы классов. В данном

документе даются все необходимые диаграммы для описания ПО: блок-схемы алгоритмов (можно ссылаться на листы), диаграммы классов, временные диаграммы, диаграммы объектов, диаграммы состояний и т.д. Документ предназначен для разработчиков или специалистов, которые будут сопровождать программный продукт, его изучать или модифицировать.

9.7.2. Стиль изложения ТО

Стиль изложения документа “Техническое описание” должен быть описательным, но основан на строгом техническом языке, принятом программистами и специалистами по разработке ПО (программистский жаргон здесь недопустим). Например, "файл имеет следующую структуру: …", "Процедура … имеет следующие входные и выходные параметры …", "класс имеет следующее назначение", "создаваемые объекты могут …" и т.д. Материал может быть (желательно) организован в виде наглядных таблиц, диаграмм, блок-схем и других формализованных описаний. В случае необходимости схемы и чертежи проекта могут быть вынесены на отдельные листы конструкторской документации, оформленные по ГОСТу. Из документа ТО, в этом случае, должны быть сделаны однозначные ссылки на содержание листов: например, “На Листе 2 представлена модульная структура программной системы …”.

9.7.3. Содержание ТО

Содержание документа ТО по пунктам приведено ниже. В образце документа ТО приведен документ ТО для варианта улиц и домов, описанного в общем пособии по курсу [3]. В шаблоне документа ТО (см. в конце данных методических указаний) даны методические указания к написанию и приспособлению документа применительно к конкретному варианту студента.

Содержание документа ТО:

1. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММНОМ ОБЕСПЕЧЕНИИ
2. МОДУЛЬНАЯ СТРУКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
3. ДИАГРАММА КЛАССОВ ПО
4. ОПИСАНИЕ МЕТОДОВ И ДАННЫХ КЛАССОВ ПО
5. ДАННЫЕ И ФАЙЛЫ ДАННЫХ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
6. ОСНОВНЫЕ АЛГОРИТМЫ МЕТОДОВ КЛАССОВ ПО
6.1. АЛГОРИТМ ...
6.2. АЛГОРИТМ ...
…
7. ОПИСАНИЕ ПРОЦЕДУР И ФУНКЦИЙ ПО
7.1. ПРОЦЕДУРА ...
7.2. ПРОЦЕДУРА ...
…
8. ОПИСАНИЕ ПРОЦЕССА ОТЛАДКИ КЛАССОВ.
9. КЛАССЫ И МЕТОДЫ, ПЕРЕОПРЕДЕЛЯЕМЫЕ В ПО

Разделы 6 и 7 могут иметь несколько подпунктов, в зависимости от состава ПО.

9.7.4. Требования к ТО

Главные требования к основным разделам технического описания ПП (на выполнение этих требований будет обращаться повышенное внимание при предъявлении ПП). Ниже требования, характерные для нашего конкретного случая КР/ДЗ:

В п.1 (“**ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММНОМ ОБЕСПЕЧЕНИИ** ”), приводятся общие характеристики программного обеспечения, дается назначение ПО, его шифр и краткая характеристика содержания документа ТО.

В п.2 (“**МОДУЛЬНАЯ СТРУКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** ”) приводится модульная структура ПО, либо в виде чертежа-схемы, либо в виде таблицы, если модулей в проекте немного. Возможно наличие и таблицы и чертежа. Дается перечень и связность модулей (исходных, объектных, библиотек и включаемых). В нашем случае это будет описание классов и их реализация. Если используются разные файлы, то дается их описание и назначение, а также распределение описаний по исходным модулям.

В п.3 (“**ДИАГРАММА КЛАССОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** ”) должна быть представлены диаграмма классов ПО и краткое их описание на содержательном уровне. Можно располагать диаграмму классов на отдельном чертеже, но желательно в формате рисунка ее повторить и в данном документе.

В п.4 (“**ОПИСАНИЕ МЕТОДОВ И ДАННЫХ КЛАССОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** ”) желательно в виде структурированной таблицы дается детальное описание классов ПО, их методов и свойств. Если класс наследует свойства базового класса, то нельзя делать ссылки на базовый класс, а нужно повторить описание данных и методов.

Примечание: Материалы с описанием процедур и членов классов необходимо размещать в специальной таблице с графами типа (см. методические указания к КЛР/ДЗ): название процедуры, способ обращения, входные параметры, выходные параметры, назначение, примечание и т.д.

В п.5 (“**ДАННЫЕ И ФАЙЛЫ ДАННЫХ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** ”) описываются структуры данных и файлов, используемых в ПО. Желательно оформить каждую структуру и файл в виде отдельной таблицы и снабдить необходимыми пояснениями полей данных.

В п.6 (“**ОСНОВНЫЕ АЛГОРИТМЫ МЕТОДОВ КЛАССОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** ”) дается общее описание алгоритмов функционирования процедур По и методов классов, если эти алгоритмы имеют уникальный характер. В частности должно быть приведено описание блок-схем программ методов, которые могут быть представлены в тексте документа или быть оформлены в виде приложении к документу ТО или отдельных листов конструкторской документации.

В п.7 (“**ОПИСАНИЕ ПРОЦЕДУР И ФУНКЦИЙ ПО** ”) должны быть описаны все процедуры и функции ПП. Описание каждой функции сводиться в таблицу, в которой отображаются: назначение, входные, промежуточные и выходные данные процедур (как в п.4).

В п.8 (“**ОПИСАНИЕ ПРОЦЕССА ОТЛАДКИ КЛАССОВ** ”) приводится описание процесса отладки ПО и специальные прогрммы, разработанные для отладки.

В п.9 (“**КЛАССЫ И МЕТОДЫ, ПЕРЕОПРЕДЕЛЯЕМЫЕ В ПО** ”) приводится краткое описание классов переопределяемых в программе и дается ссылка о том, в каких библиотеках базовые классы описаны. В документе приводиться также список библиотечных классов с указанием стандартной библиотеки.

9.7.5. Разработка ТО

Разработка ТО выполняется разработчиками ПО или специалистами по разработке документации совместно с разработчиками ПО. Начинается разработка ТО фактически с началом проекта, а завершается (или уточняется содержание) в конце проекта. Поэтому желательно сразу, на основе шаблона, создать такой документ и постоянно в него добав-

лять информации. Например, завершена разработка и отладка одного класса, можно его описание отобразить в ТО. Иногда, и даже более правильно, сделать описание класса или алгоритма в формате ТО, а только затем выполнить его реализацию, внося, при необходимости, уточнения и изменения в документ ТО.

9.8. Руководство пользователя

9.8.1. Документ руководство пользователя (РП) ПО и его назначение

Документ “Руководство пользователя” (РП) программного продукта (данные документ ориентируется на потенциального пользователя ПО). Он является, своего рода, инструкцией по эксплуатации программного изделия. Документ должен описывать, включая примеры, все свойства и методы классов, разрабатываемых в проекте. Кроме этого в документе приводится: условия применения и общая характеристики ПО. Данный документ ориентирован на конечного пользователя, который применяет данный программный продукт. В нашем случае – это пользователь применяющий – программист, использующий систему классов в своих программных разработках.

9.8.2. Стиль изложения РП

Стиль изложения руководства пользователя должен быть описательным и исчерпывающим. Описание необходимо давать простым языком, без злоупотреблений техническими терминами. Должны рассматриваться различные примеры и рисунки для различных режимов работы, рассматриваться ошибочные ситуации и диагностические сообщения. Данный документ должен понимать тот пользователь, на которого рассчитан программный продукт. В тексте документа не должно содержаться пространных рассуждений и материала, не относящегося к программному продукту. Должна быть дана подробная инструкция применения программы, включая запуск, использование и выгрузку. системы классов: подключения в программу, создания и манипулирования объектами, использование всех методов. Должны быть даны наглядные примеры для всех случаев использования.

9.8.3. Разработка РП

Разработка документа РП выполняется в конце всей разработки программного продукта разработчиками совместно с техническими “писателями”. Для создания РП возможно разрабатывается специальная программа с использованием классов, примеры из которой должны быть помещены в документ. Все примеры в РП должны быть тщательно проверены. Они должны быть наглядными и охватывать весь спектр возможных применений программного обеспечения.

9.8.4. Содержание РП

Содержание документа РП по пунктам приведено ниже. В образце документа РП приведено РП для варианта улиц и домов, описанного в общем пособии по курсу [3]. В шаблоне документа РП (см. в конце данных методических указаний) даны методические указания к написанию и приспособлению документа шаблона применительно к конкретному варианту студента.

Содержание документа РП:

1. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ СИСТЕМЫ КЛАССОВ
2. ПОДКЛЮЧЕНИЕ СИСТЕМЫ КЛАССОВ И ДИАГРАММА КЛАССОВ
3. РАБОТА С СИСТЕМОЙ КЛАССОВ
4. КЛАСС XXX

4.1 Использование свойств XXX класса XXX с примером…

4.2 Использование метода XXX класса XXX с примером…

5. ОТКЛЮЧЕНИЕ СИСТЕМЫ КЛАССОВ
6. ОПИСАНИЕ СООБЩЕНИЙ ОБ ОШИБКАХ И ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ ПРИ РАБОТЕ С ПРОГРАММОЙ

9.8.5. Требования к РП

Главные требования к основным разделам документа “Руководства пользователя” (на выполнение этих требований будет обращаться повышенное внимание при сдаче ПО) рассмотрены ниже:

В п.1 (“**НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ СИСТЕМЫ КЛАССОВ**”) приводится назначение ПО и подчеркиваются качественные характеристики изделия. В этом разделе характеризуются предметные области применения программного продукта.

В п.2 (“**ПОДКЛЮЧЕНИЕ СИСТЕМЫ КЛАССОВ И ДИАГРАММА КЛАССОВ**”) рассматривается по шагам процесс подключения программного обеспечения с собственные проекты.

В п.3 (“**РАБОТА С СИСТЕМОЙ КЛАССОВ**”) приводятся качественное описание системы классов и дается диаграмма классов проекта. Рассмотрены вопросы подключения системы классов в программы, а также используемые общие переменные и функции во всех классах.

В п.4 (“**КЛАСС XXX**”) приводятся описание конкретного класса с методами и свойствами.

В п.4.1-X (“**Использование метода/свойства XXX класса XXX с примером**”) дается описание с примерами использования свойств и методов класса.

В п.5 (“**ОТКЛЮЧЕНИЕ СИСТЕМЫ КЛАССОВ**”) приведено описание действий, необходимых для отключения системы классов из проекта.

В п.6 (“**ОПИСАНИЕ СООБЩЕНИЙ ОБ ОШИБКАХ И ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ ПРИ РАБОТЕ С ПРОГРАММОЙ** ”) описываются диагностические сообщения системы и дается описание этих сообщений и ситуаций. Желательно эти сведения собрать в таблицу. Должны быть описаны действия, которые нужно предпринять при возникновении исключительных ситуаций.

9.9. Руководство системного программиста

9.9.1. Документ руководство системного программиста (РСП) ПО и его назначение

Документ Руководство системного программиста разрабатывается программистом для системного программиста или системного администратора той организации, в которой программный продукт будет использоваться. Этот специалист отвечает за работоспособность техники и операционных систем и выполняет функции по установке и сопровождению ПП на конкретных компьютерах. В этот документ должна быть собрана вся необходимая информация для выполнения этих работ, в том числе и та, которая присутствует в других документах. Поэтому ряд пунктов в этом документе совпадает с пунктами в других документах комплекта ПД.

9.9.2. Стиль изложения РСП

Стиль изложения руководства системного программиста должен быть предписывающим (все термины должны быть техническими): "Для установки ПП нужно выполнить следующие действия …", "ПП состоит из набора следующих компонент …". Здесь могут использоваться специальные термины, но не жаргон программистов.

9.9.3. Содержание РСП

Содержание документа РСП по пунктам приведено ниже. В образце документа РСП приведен документ РСП для варианта улиц и домов, описанного в общем пособии по курсу [3]. В шаблоне документа РСП (см. в конце данных методических указаний) даны методические указания к написанию и приспособлению документа применительно к конкретному варианту студента.

Содержание документа РСП:

1. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММЫ
2. ТРЕБОВАНИЯ К СОСТАВУ И ПАРАМЕТРАМ ТЕХНИЧЕСКИХ СРЕДСТВ
3. ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ
4. СОСТАВ ПРОГРАММНОГО ПРОДУКТА
5. УСТАНОВКА ПРОГРАММНОГО ПРОДУКТА
6. УДАЛЕНИЕ ПРОГРАММНОГО ПРОДУКТА
7. ЗАПУСК ПРОГРАММЫ
8. ЗАВЕРШЕНИЕ РАБОТЫ ПРОГРАММЫ
9. СООБЩЕНИЯ ОБ ОШИБКАХ ПРОГРАММНОГО ПРОДУКТА
10. УСЛОВИЯ ЭКСПЛУАТАЦИИ ПРОГРАММНОГО ПРОДУКТА
11. ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО ПРОДУКТА
12. ПОДГОТОВКА К РАБОТЕ С ПРОГРАММОЙ
13. ОБШИЕ ХАРАКТЕРИСТИКИ ПРОГРАММНОГО ПРОДУКТА

9.9.4. Требования к РСП

Главные требования к основным разделам руководства системного программиста программного обеспечения приведены ниже.

В п.1 (“**НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММЫ**”) рассмотрено назначение и условия применения программного продукта.

В п.2 (“**ТРЕБОВАНИЯ К СОСТАВУ И ПАРАМЕТРАМ ТЕХНИЧЕСКИХ СРЕДСТВ**”) приводятся требования к техническим характеристикам для работы программного обеспечения.

В п.3 (“**ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ**”) приводятся требования к программному обеспечению для функционирования программного продукта.

В п.4 (“ **СОСТАВ ПРОГРАММНОГО ПРОДУКТА**”) приводится перечень составляющих модулей программного продукта.

В п.5 (“ **УСТАНОВКА ПРОГРАММНОГО ПРОДУКТА**”) описывается процесс развертывания программного продукта на отдельном компьютере в среде программирования C++. Подключения системы классов в используемую систему программирования по шагам. Описываются все ситуации, включая и нештатные, когда продукт установить нельзя. Если, при установке, для конкретного шага, выдается информационное сообщение, то оно должно быть тоже приведено.

В п.6 (“**6. УДАЛЕНИЕ ПРОГРАММНОГО ПРОДУКТА**”) описывается процесс удаления программного продукта с отдельного компьютера. Описание нужно дать по шагам. Описываются все ситуации, включая и нештатные, когда продукт удалить нельзя. Если, при удалении, для конкретного шага, выдается информационное сообщение, то оно должно быть тоже приведено.

В п.7 (“**7. ЗАПУСК ПРОГРАММЫ**”) рассмотрены действия для запуска программы тестового примера.

В п.8 (“**8. ЗАВЕРШЕНИЕ РАБОТЫ ПРОГРАММЫ**”) описаны действия для завершения работы программы тестового примера.

В п.9 (“**9. СООБЩЕНИЯ ОБ ОШИБКАХ ПРОГРАММНОГО ПРОДУКТА**”) приводятся диагностические сообщения об ошибках и исключительных ситуациях, которые могут появиться при эксплуатации программного продукта.

В п.10 (“**10. УСЛОВИЯ ЭКСПЛУАТАЦИИ ПРОГРАММНОГО ПРОДУКТА**”) дается описание условий эксплуатации программного продукта.

В п.11 (“**11. ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО ПРОДУКТА**”) описывается процесс проверки работоспособности программного продукта.

В п.12 (“**12. ПОДГОТОВКА К РАБОТЕ С ПРОГРАММОЙ**”) рассмотрены вопросы подготовки использования программного продукта.

В п.13 (“**13. ОБШИЕ ХАРАКТЕРИСТИКИ ПРОГРАММНОГО ПРОДУКТА**”) приводятся общие характеристики программного продукта.

Описания пунктов 3, 4, 9, 10, 11 и 13 более были рассмотрены при изучении других программных документов, поэтому здесь мы не повторяем это описание.

Примечание. Последнее замечание нужно учитывать в двух аспектах: нужно согласовывать описание одинаковых пунктов в разных документах и не тратить лишнее время на повторную разработку раздела документа.

9.9.5. Разработка РСП

Разработка РСП выполняется грамотным системным программистом после завершения всей разработки и программного обеспечения и программной документации. Все пункты РСП должны быть проверены на практике. Они должны корректироваться при изменении условий эксплуатации и доработках программного обеспечения.

9.10. Программа и методика испытаний

9.10.1. Документ программа и методика испытаний (ПМИ) ПО и его назначение

Документ “Программа и методика испытаний” (ПМИ) разрабатывается специально для проведения приемно-сдаточных испытаний для сдачи ПО. В нашем случае этот документ разрабатывается для проведения испытаний на основе пунктов ТЗ.

Данный документ ориентирован на заказчика, который будет выполнять приемку программного продукта. Возможны различные варианты построения документа (ПМИ), который устанавливается по соглашению с заказчиком: приемка на основе проверки выполнения пунктов ТЗ (5.1 раздел ТЗ в первую очередь); приемка ориентированная на проверку работоспособности системы; комбинированная приемка по ТЗ и работоспособности и т.д. Для нашего случая используется вариант приемки на основе пунктов ТЗ и проверки работоспособности системы, поэтому, отметим еще раз, что в ТЗ все позиции должны быть пронумерованы, для того, чтобы на них можно было ссылаться в ПМИ.

Документ ПМИ должен быть фактически пошаговой инструкцией для проведения испытаний, специалист, принимающий испытания должен без подсказок разработчика самостоятельно выполнить действия по проверке и удостоверится в работоспособности программного продукта и выполнения всех пунктов ТЗ.

9.10.2. Стиль изложения документа ПМИ

Стиль изложения должен быть предписывающим, и основан на техническом языке, принятом программистами и специалистами по разработке ПП (программистский жаргон здесь недопустим). Недопустимо, также, использование бытовых терминов. Все тексты должны трактоваться однозначно. Например, "нажмите … клавишу", "получите на экране …", "испытания должны проводиться в режиме командной строки для CMD.EXE" и т.д. Неоднозначностей и ошибок в тексте документа ПМИ не должно быть. Данный документ должен быть лаконичным и четким. Не нужно строить сложные предложения.

9.10.3. Разработка документа ПМИ

Разработка документа ПМИ выполняется в конце всей разработки программного продукта специалистами по программированию и заказчиком, но ее план задумывается уже при составлении ТЗ на программное обеспечение. Перед разработкой ПМИ должен быть подготовлен специальный тестовый пример, ориентированный для проверки пунктов ТЗ. Нужно иметь в виду, что при разработке ТЗ вы уже должны понимать, как могут быть проверены эти пункты в документе ПМИ. Для проверки правильности документа необходимо провести испытания самостоятельно, использую в роли заказчика представителя не связанного с вашим проектом.

9.10.4. Содержание документа ПМИ

Содержание документа ПМИ по пунктам приведено ниже. В образце документа ПМИ приведен документ ПМИ для варианта улиц и домов, описанного в общем пособии по курсу [3]. В шаблоне документа ПМИ (см. в конце данных методических указаний) даны методические указания к написанию и приспособлению документа шаблона применительно к конкретному варианту студента.

Содержание документа ПМИ:

1. ОБЪЕКТ ИСПЫТАНИЙ
2. ЦЕЛЬ ИСПЫТАНИЙ
3. СОСТАВ ПРЕДЪЯВЛЯЕМОЙ ДОКУМЕНТАЦИИ
4. ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ К ИСПЫТАНИЯМ И УСЛОВИЯ ПРОВЕДЕНИЯ ИСПЫТАНИЯ
   4.1. Требования к условиям проведения испытаний
   4.2. Требования к техническим средствам
5. ПОРЯДОК ПРОВЕДЕНИЯ ИСПЫТАНИЙ
   5.1. Состав и структура технических и программных средств для проведения испытаний программного продукта.
   5.2. Последовательность испытаний (в виде таблицы)
6. МЕТОДЫ ИСПЫТАНИЯ
7. РЕЗУЛЬТАТЫ ИСПЫТАНИЙ

9.10.5. Требования к документу ПМИ

Главные требования к основным разделам документа “Программы и методики испытания” (на выполнение этих требований будет обращаться повышенное внимание при сдаче ПО) следующие:

В п.1 (“**ОБЪЕКТ ИСПЫТАНИЙ**”) описывается, что будет испытываться, включая и наименование программного изделия. В учебных целях здесь, не допускаются ссылки на другие документы проекта (ДЗ/КЛР), особенно при задании условий проведения испытаний.

В п.2 (“**ЦЕЛЬ ИСПЫТАНИЙ**”) устанавливается цель испытания, то есть фактически устанавливается вариант проведения приемно-сдаточных испытаний программного изделия.

В п.3 (“**СОСТАВ ПРЕДЪЯВЛЯЕМОЙ ДОКУМЕНТАЦИИ**”) приводится полный перечень представляемой документации для проведения испытаний.

В п.4 (“**ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ К ИСПЫТАНИЯМ**”) определяются условия проведения испытаний, включая и технические (ОС, требования к компьютеру и т.д.). При формировании условий проведения испытаний необходимо учитывать возможности заказчика (преподавателя), которые он может предоставить для проведения приемки. Сдача программного продукта на технических и программных средствах разработчика недопустима.

В п.5 (“**ПОРЯДОК ПРОВЕДЕНИЯ ИСПЫТАНИЙ**”) В пункте 5.1 определены конкретные условия проведения испытания для проверки. В пункте 5.2 в виде таблицы, формат которой приведен ниже, дается перечень действий и результатов, необходимых для проведения испытаний. Пример таблицы:

№ п.п.	№ пункта ТЗ	Выполняемые действия	Ожидаемый результат	Дополнительные требования
1	2	3	4	5
…	Запуск программ	…	…	

№ п.п.	№ пункта ТЗ	Выполняемые действия	Ожидаемый результат	Дополнительные требования
1	**2**	**3**	**4**	**5**
1.	5.1.1 Должна быть обеспечена возможность сложения улиц	Выполните пункт меню № 3. (Выполняется фрагмент тестового примера, где задается сложение. См. п 2.7 "Описание тестового примера" S3 = S1+S2;)	Результат выполнения представлен в п.п. 4.7 данного документа. (**Примечания для студентов:** в п.п. 4.7 отображаются все результаты, которые проверяющий должен проверить – детальный вывод на экран и другие результаты).	
ИЛИ	5.1.1 Должно быть обеспечено создание объекта типа дом	Выполните пункт меню № 1. Введите “1” и нажмите ENTER. (Выполнется фрагмент тестового примера, где создается объект. См. п 2.1 "Описание тестового примера" – Home H1("Жилой", "д.10", 7,2,10, multiple , 8, false);)	На экране будет распечатано содержимое объекта: **Номер сп. - 0 Имя - Жилой** **Номер сим. - д.10 Номер - 7** **Этажей - 2 Жителей - 10** **Ремонт не нужен! Тип дома - много строений Число квартир – 8** (**Примечания для студентов:** ячейке таблицы отображаются все результаты, которые проверяющий должен проверить – детальный вывод на экран и другие результаты).	
…	…	…	…	…
…	Завершение программ	…	…	

Примечания: 1. В колонке 2 можно не раскрывать полностью текстовое содержание пункта ТЗ, но желательно это сделать. Ссылка на пункт ТЗ обязательна. Номера по порядку в таблице должны соответствовать логически выделенным шагам проведения испытаний. Они не должны объединять одновременно несколько пунктов ТЗ.

2. Выполняемые действия должны быть точными, расписанными в точности до нажатия отдельной клавиши.

3. Если результат трудно поместить в ячейке данной таблицы, то его можно разместить в конце данного документа (в разделе результаты испытаний), а в таблице дать ссылку на раздел и страницу, где размещены эти результаты.

4. Каждая строка таблицы проверки пунктов ТЗ должна содержать описание действий для выборочной проверки этого пункта. Поэтому нужно продумать все необходимое и возможно продублировать действия других пунктов или выполнить ссылку на них (по пункту в колонке №1).

5. Колонка 5 таблицы не является обязательной.

В п.6 (“**МЕТОДЫ ИСПЫТАНИЯ**”) в данном разделе описываются специальные методы, которые используются при проведении испытаний (Например, способы измерения или измерительные инструменты). Рассматриваются методики проведения испытаний, которые трудно отобразить в таблице п.5 (например, выводимые результаты на эк-

ран, сложные эксперименты для проверки работоспособности и т.д.). В этом случае пункты должны быть пронумерованы, а в таблице ПМИ будут сделаны ссылки на эти пункты.

В п.7 (“**РЕЗУЛЬТАТЫ ИСПЫТАНИЙ**”) размещаются результаты испытаний с возможностью ссылки на них из таблицы испытаний. Они даются с стиле рисунков или текста результатов, полученных при проведении испытаний.

9.11. Описание тестового примера

9.11.1. Документ описание тестового примера (ОТП) ПО и его назначение

В данном документе дается описание тестового примера, который передается пользователю для проверки программного продукта. Этот тестовый пример может быть также использован при оформлении документа “Руководство пользователя” (РП). Однако в целях ясности и наглядности РП тестовый пример, ориентированный на проведение испытаний может отличаться. В этом документе, для нашего случая, должны приведены фрагменты текста программ с использованием собственных классов и результаты выполнения этих программ. Приводиться краткое описание текста и результатов.

9.11.2. Стиль изложения ОТП

Стиль изложения документа “Описание тестового примера” должен быть описательным. Он должен ориентироваться на заказчика и специалиста, который будет проводить приемно-сдаточные испытания. В тестовом примере, для нашего случая, лучше использовать переключатель и меню, для проверки пунктов ТЗ из раздела функциональных требований.

9.11.3. Содержание ОТП

Содержание документа ОТП по пунктам приведено ниже. В образце документа ОТП приведен документ ОТП для варианта улиц и домов, описанного в общем пособии по курсу [3]. В шаблоне документа ОТП (см. в конце данных методических указаний) даны методические указания к написанию и приспособлению документа применительно к конкретному варианту студента.

Содержание документа ОТП:

1. ОПИСАНИЕ НАЗНАЧЕНИЯ ТЕСТОВОГО ПРИМЕРА
2. ПОЯСНЕНИЕ КОДА ПРОГРАММЫ ТЕСТОВОГО ПРИМЕРА
   - 2.1. Первоначальные описания в тестовом примере
   - 2.2. Структура главной программы
   - 2.3. Фрагмент текста программы для проверки п.п.5.1.1 ТЗ
   - …

9.11.4. Требования к ОТП

Главные требования к основным разделам описания тестового примера рассмотрены ниже.

В п.1 (“**ОПИСАНИЕ НАЗНАЧЕНИЯ ТЕСТОВОГО ПРИМЕРА**”) описывается назначение создания тестового примера. Приводятся имена исходных файлов и проектов, которые необходимы, чтобы тестовый пример был выполнен заказчиком и при испытаниях ПО. Даются ссылки на документы необходимые для выполнения тестового примера.

В п.2 (“**ПОЯСНЕНИЕ КОДА ПРОГРАММЫ ТЕСТОВОГО ПРИМЕРА**”) рассматриваются все фрагменты текста тестового примера, результаты их работы и даются необходимые описания для создания проекта в VS.

В п.2.1 (“**Первоначальные описания в тестовом примере**”) рассматриваются необходимые описания в тестовом примере и описания заголовочных файлов, подключаемых библиотек.

В п.2.2 (“**Структура главной программы**”) описывается структура главной программы тестового примера с пояснениями. Для удобной проверки в ПМИ, в нашем случае, она должна иметь циклическую структуру с тестовым меню и переключателем, организованным по пунктам ТЗ проверяемым в ПМИ.

В п.2.3-X (“**Фрагмент текста программы для проверки п.п.5.1.1 ТЗ** ”) описываются фрагменты текста тестового примера для проверки каждого пункта ТЗ. Даются результаты работы этих фрагментов и краткие пояснения к ним.

9.11.5. Разработка ОТП

Разработка документа ОТП выполняется совместно с ПМИ. Разработка самого тестового примера, исходный текст которого помещается в документ “Исходные тексты программ” выполняется при отладке программного обеспечения. Возможно, он будет являться частью программы, которая использовалась при разработке и отладке системы классов. Либо он разрабатывается отдельно, по завершению отладочных мероприятий при разработке ПО.

10. Вопросы для самопроверки

После выполнения КЛР/ДЗ студенты должны отвечать на следующие контрольные вопросы.

1. Дайте краткое определение понятия класса.
2. Дайте развернутое определение понятия класса.
3. Что такое конструктор и для чего они нужно?
4. Что такое деструктор и для чего они нужны?
5. Что такое инкапсуляция?
6. Дайте определение понятия наследования.
7. Что такое базовый и производный классы?
8. Что такое множественное наследование?
9. Какие изменения в порожденном классе модно сделать при наследовании?
10. Чем различаются классы и структуры данных?
11. Что такое перегруженные операции в классе?
12. Что такое статическое связывание?
13. Что такое динамическое связывание?
14. Какие виды перегрузки операций Вы знаете?
15. Каково назначение перегрузки операций с точки зрения программиста?
16. Что такое контейнерный объект?
17. Какие разновидности контейнеров вы знаете?
18. Какие операции выполняются с контейнерами?
19. Какие операции с массивами вы знаете?
20. Перечислите основные методы классов CArray и CObArray.
21. В каком отношении элементный класс находиться с контейнерным классом?
22. Какие признаки классификации контейнеров вы знаете?
23. Что такое позиция? Как выполняется навигация по спискам.
24. Что такое итератор и для чего он используется?
25. Какие операции со списками вы знаете?
26. Перечислите основные методы классов CList и CObList.
27. Для чего нужна программная документация?
28. Что включается в комплект ПД для ДЗ/КЛР?
29. Кто разрабатывает ТЗ?
30. Какой из разделов ТЗ является самым сложным для разработки и почему?

31. Для чего нужно ОП и его назначение?
32. Каково содержание документа ТО?
33. Почему необходима нумерация пунктов ТО?
34. Для чего нужно РП и его назначение?
35. Для чего нужно РСП и его назначение?
36. Для чего нужно ПМИ и его назначение?
37. Каковы основные требования к тексту документа ПМИ?
38. Какой из разделов ПМИ является самым сложным для разработки и почему?
39. Что такое предметная область?
40. Как формируются функциональные требования к решаемой задаче.

11. Литература для ДЗ

1. Г. Шилдт “C++ Базовый курс”: Пер. с англ.- М., Издательский дом “Вильямс”, 2011 г. – 672с
2. Г. Шилдт “C++ Руководство для начинающих” : Пер. с англ. - М., Издательский дом “Вильямс”, 2005 г. – 672с
3. Г. Шилдт “Полный справочник по C++”: Пер. с англ.- М., Издательский дом “Вильямс”, 2006 г. – 800с
4. Бьерн Страуструп "Язык программирования C++"- М., Бином, 2010 г.
5. MSDN Library for Visual Studio 2005 (Vicrosoft Document Explorer – входить в состав дистрибутива VS. Нужно обязательно развернуть при установке!)
6. Общее методическое пособие по курсу для выполнения ЛР и КЛР/ДЭ (см. на сайте 1-й курс www.sergebolshakov.ru) – см. кнопку в конце каждого раздела сайта!!!
7. Г.С.Иванова, Т.Н. Ничушкина, Е.К.Пугачев "Объектно-ориентированное программирование". – М., МГТУ, 2001 г.
8. Другие методические материалы по дисциплине с сайта www.sergebolshakov.ru.
9. Конспекты лекций по дисциплине “Программирование на основе классов и шаблонов”.
10. Страуструп Б. "Дизайн и эволюция C++. Классика CS" – СПб,: Питер , 2007. – 445с.
11. Глаголев В.А. Разработка технический документации. Руководство для технических писателей и локализаторов ПО – СПб.: Питер, 2008. – 192с.:ил.
ISBN 978-5-388-00101-6
12. Эккель Б. Философия C++. Введение в стандартный C++. 2-е изд.- СПб.: Питер, 2004.- 572с.: ил.
13. Эккель Б. Эллисон Ч. Философия C++. Практическое программирование. - СПб.: Питер, 2004.- 608с.: ил.
14. Страуструп, Бьярн. Программирование: принципы и практика с использованием C++, 2-е изд. : Пер. с англ. - М . : ООО "И . Д. Вильяме", 2016. - 1328с. : ил . - Парал. тит. англ.
ISBN 978-5-8459- 1 949-6 (рус .)

Приложение 1 Образцы документов

Московский государственный технический университет им. Н.Э.Баумана

УТВЕРЖДАЮ:

Большаков С.А.

"__"_____201X Г.

Комплексная лабораторная работа/ДЗ по дисциплине ПКШ

"Система классов улиц и домов"

Техническое задание

(вид документа)

писчая бумага

(вид носителя)

7

(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-XX _____

Большаков С.А.

"__"_____201X г.

Москва - 201X

СОДЕРЖАНИЕ

1.НАИМЕНОВАНИЕ

Система классов для работы с улицами и домами. Шифр разработки программной системы – **DZ_PCT**.

2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

Основанием для разработки данного программного продукта является учебный план кафедры "Системы обработки информации и управления" МГТУ им. Н.Э. Баумана на 2-м семестре.

3. НАЗНАЧЕНИЕ РАЗРАБОТКИ

Разрабатываемая система классов предназначена для автоматизации работы с объектами улиц и домов в программных проектах и предметной области, где необходимо это учитывать. В частности система классов должна обеспечивать решение задач: оценки ремонта домов и улиц, подсчета числа жителей, квартир и этажей в домах. Система классов должна обеспечить удобную работу с этими объектами, высокий уровень надежности программ, функциональных возможностей, а также сокращение сроков разработки и реализации программных продуктов, где необходимо использовать подобные объекты.

4. ИСПОЛНИТЕЛЬ

Студент группы ИУ5-XX МГТУ им. Н.Э. Баумана Большаков Сергей Алексеевич.

5. ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

5.1.Требования к функциональным характеристикам

Система классов должна предоставлять пользователю необходимые методы для использования объектов аннотированных ссылок и их списков в программах. Система классов должна быть иерархической, то есть связанной при помощи механизма наследования классов. В вершине иерархии должен быть абстрактный класс.

Система классов улиц и домов должна обеспечивать выполнение следующих функции для работы с этими объектами:

5.1.1. Создание контейнерных объектов для улиц города, в которых могут размещаться дома этой улицы.

5.1.2. Создание объектов для домов улицы, с параметрами и без параметров.

5.1.3. Создание объектов для домов улицы на основе других.

5.1.4. Учет следующих свойств дома: номер дома, число жителей, число квартир, число этажей, признак необходимости ремонта, тип дома.

5.1.5. Задание и получение характеристик дома.

5.1.6. Сложение двух домов.

5.1.7. Перегрузить оператор присваивания для домов.

5.1.8. Распечатка характеристик дома.

5.1.9. Учет следующих свойств улицы: название улицы, признак ремонта необходимости улицы, признак необходимости ремонта домов улицы, число домов на улице, тип улицы.

5.1.10. Распечатка содержания улицы и ее свойств.

5.1.11. Задание характеристик улицы.

5.1.12. Получение характеристик улицы.

5.1.13. Сложение двух улиц.

5.1.14. Добавление дома на улицу.

5.1.15. Удаление дома с улицы.

5.1.16. Установка и снятие признака ремонта улицы.

5.1.17. Автоматическое получение признака ремонта домов улицы.

5.1.18. Перегрузка оператора присваивания для улиц.

5.2. Требования к программному обеспечению

5.2.1. Данная система классов предназначена для использования в программах, выполняемых на компьютере под управлением системы Microsoft Windows 2000 и выше. Использование разрабатываемой библиотеки требует наличия компилятора языка C++ и системы программирования (MS VS 2005/2008/2010).

5.3. Требования к условиям эксплуатации

5.3.1.Данная система классов должна эксплуатироваться совместно с языком программирования C++ в среде MS VS 2005/2008/2010. Для работы с данной системой классов программист должен быть знаком с навыками объектно-ориентированного программирования.

5.3.2. В остальном требования к эксплуатации точно такие же, как к программной реализации языка C++, используемой совместно с данной системой классов.

5.3.3. Программа тестового примера для проведения испытаний должна работать в среде компьютера, без установленной системы программирования MS VS 2005.

5.4. Требования к информационному обеспечению

5.4.1. Система классов должна быть реализована на языке C++ в среде MS VS (или BC++ 3.1) на основе файловой системы.

5.4.2. Информация о специальных структурах данных выноситься на листы и в документ Техническое Описание.

5.4.3. Система классов должна быть представлена набором файлов с исходными текстами с расширением *.hpp и *.cpp (или *.h и *.cpp).

5.4.4. Вывод и ввод данных в/из программы должен выполняться на русском языке.

5.4.5. Комментарии в исходном тексте программ должны быть на русском языке.

5.4.6. Разрабатываемые классы и наследуемые от них классы не должны конфликтовать по именам с уже существующими в VS MS 2005 (библиотеки MFC, ATL, FCL и CRL).

5.5. Требования к надежности

5.5.1. Система классов будет правильно функционировать при условии правильной ее эксплуатации пользователем (программистом) и при отсутствии сбоев операционной системы и технического обеспечения.

5.5.2. В результате использования данной системы классов не допускается выделение, а потом не освобождение программой участков памяти.

5.5.3. Контроль входной и выходной информации должен производиться компилятором, совместно с которым будет использоваться данная система классов.

5.6. Требования к составу и характеристикам технических средств

Данная система классов должна использоваться на компьютерах следующей конфигурации:

5.6.1. IBM-совместимый компьютер с процессором 80486 и выше;

5.6.2. Не менее 1 Мбайт свободной оперативной памяти;

5.6.3. VGA-совместимый видеоадаптер и монитор;

5.6.4. Стандартная клавиатура;

5.6.5. Свободного места на жёстком диске не менее 400 Кбайт.

5.7. Требования к программной совместимости

5.7.1.Система классов должна сопровождаться демонстрационной программой в виде *.exe файла;

5.7.2.Система классов может использоваться только с компилятором языка C++.

5.7.3.Компьютер должен быть оснащен русской таблицей символов знакогенератора.

5.7.4. Все тексты, комментарии и ввод/вывод информации должны осуществляться на русском языке.

5.8. Требования к маркировке и упаковке программы

5.8.1. Программа предоставляется на дискете 3,5" или CD/DVD носителе.

5.9. Требования к транспортированию и хранению

5.9.1. Программа предоставляется на дискете 3,5" или CD/DVD носителе.

6. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

6.1. По окончанию работы должны быть предъявлены следующие документы:
6.1.1. Техническое задание;
6.1.2. Описание применения программного продукта;
6.1.3. Техническое описание программного продукта;
6.1.4. Руководство пользователя;
6.1.5. Руководство системного программиста;
6.1.6. Исходные тексты программ системы классов и тестового примера;
6.1.7. Программа и методика испытаний;
6.1.8. Описание тестового примера;

7. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

7.1.Требования по данным показателям при выполнении ДЗ по ООП не предъявляются.

8. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

8.1. Сроки выполнения отдельных этапов работ по ДЗ:

8.1.1. Получение и уточнение задания –2-4 недели семестра.
8.1.2. Подписание ТЗ –6-7 недели семестра.
8.1.3. Разработка программ – 3-7 недели семестра.
8.1.4. Кодирование и отладка – 8-9 недели семестра.
8.1.5. Разработка документации – 9-12 недели семестра.
8.1.6. Защита и проведение испытаний – 12-13 недели семестра.

9. ПОРЯДОК КОНТРОЛЯ И ПРИЁМКИ ЗАДАНИЯ

9.1. Требования к сдаче и условия приемки

9.1.1. Тестирование программного продукта будет осуществляться на основании тестового примера в соответствии с документом "Программа и методика испытаний" (ПМИ) на компьютере, который удовлетворяет требованиям, указанным в пунктах "Требования к составу и характеристикам технических средств" и "Требования к программному обеспечению" данного технического задания. Испытания проводятся по пунктам настоящего ТЗ, в том числе и выборочно.

10. ДОПОЛНИТЕЛЬНЫЕ ТРЕБОВАНИЯ

Данное техническое задание может уточняться в установленном порядке.

Московский государственный технический университет им. Н.Э.Баумана

УТВЕРЖДАЮ:

Большаков С.А.
"__"_____201X Г.

Комплексная лабораторная работа/ДЗ по дисциплине ПКШ

"Система классов улиц и домов"

Описание применения

(вид документа)

писчая бумага
(вид носителя)

5
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-XX _____
Большаков С.А.
"__"_____201X г.

Москва - 201X

СОДЕРЖАНИЕ

1. НАЗНАЧЕНИЕ ПРОГРАММНОГО ПРОДУКТА

Система классов домов и улиц предназначена для автоматизации работы с объектами улиц и домов в программных проектах и предметной области, где необходимо это учитывать. В частности, система классов предназначена для программирования задач включающих действия: оценки ремонта домов и улиц, подсчета числа жителей, квартир и этажей в домах. Система классов обеспечивает удобную работу с этими объектами, высокий уровень надежности программ, функциональных возможностей, а также сокращение сроков разработки и реализации программных продуктов, где необходимо использовать подобные объекты.

2. ВОЗМОЖНОСТИ ПРОГРАММНОГО ПРОДУКТА

2.1. Общие сведения о программном продукте

Система классов описывает дома и улицы для разработки программ, в которых учет сведений об этих объектах необходим.

Дом – объекты данного типа содержат информацию о номере дома, этажности, числе жителей и квартир, необходимости ремонта дома и типе дома. Предусматривается возможность изменения параметров дома.

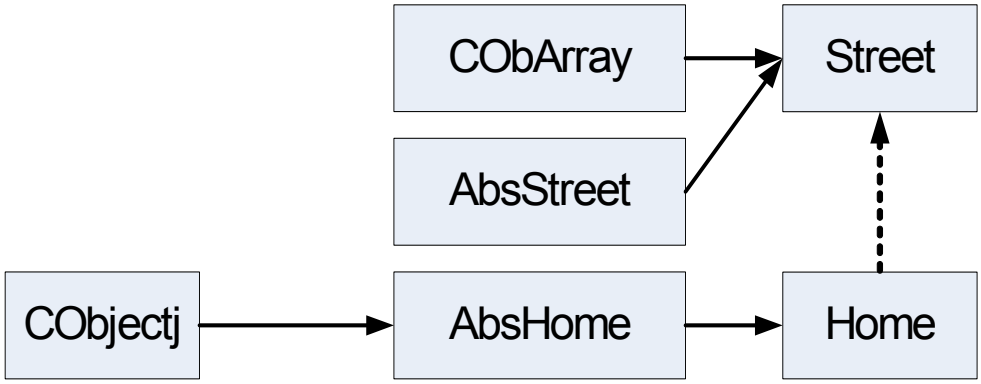
Улица – объекты данного типа в упорядоченном виде содержат информацию о домах улице, названии улицы, типе улицы, соседних улицах (как резерв), необходимости ремонта улицы, числе домов, номера улицы (как резерв). Предусматривается возможность добавления домов на улицу и их удаления, изменения других параметров улицы.

Система классов позволяет программировать следующие операции и функции:

- Создание и задание свойств объектов типа дом и улица;
- Установка признака ремонта домов;
- Сложение двух домов;
- Получение параметров домов (номера, числа жителей, числа квартир, типа дома);
- Установки и изменения параметров дома (номера, числа жителей, числа квартир, типа дома);
- Удаление объектов типа дом и улица;
- Распечатки свойств объектов типа дом и улица;
- Добавление домов на конкретную улицу;
- Удаление дома с конкретной улицы;
- Сложение двух улиц (при объединении улиц);
- Переименование улиц и новую нумерацию домов;
- Распечатки списка домов улицы с их характеристиками;
- Получение параметров улицы (названия, числа жителей, числа квартир, типа улицы);
- Установки и изменения параметров улицы (названия, числа жителей, числа квартир, типа улицы);
- Автоматическое получение признаков необходимости ремонта домов на улице.

2.2. Диаграмма классов программного продукта

Ниже приставлена диаграмма классов системы классов улиц и домов программного обеспечения (ПО).



Классы системы имеют следующее назначение:

Класс **CObject** - системный абстрактный класс для наследования общих свойств объектов системы классов.

боту со списками (включение, просмотр, удаление, печать и др.).

Класс **AbsStreet** - абстрактный класс, в котором учтены общие свойства улиц.

Класс **AbsHome** - абстрактный класс, в котором учтены общие свойства домов, размещаемых на улицах.

Класс **Street** - класс улиц, для создания объекта типа улица, позволяющий создавать объекты, учитывающие свойства: название, список домов, число домов и др.

Класс **Home** - класс дома, позволяющий создавать объекты описывающие дома, в которых учтены свойства: номер, число квартир, число жителей, признак ремонта и др.

3. ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ПРОГРАММНОГО ПРОДУКТА

Основные характеристики программного обеспечения сведены в таблицу, расположенную ниже. Содержание файлов представлено в документе “Исходные тексты программ”. Описание возможностей программного обеспечения даны в документе “Руководство пользователя” и “Описание тестового примера”. Действия необходимые для установки программного продукта представлены в документе “Руководство системного программиста”. Состав набора исходных файлов:

Название	Размер и тип	Описание	Примечание
DZ_Class.h	5,2 Кб, текстовый файл	Описания классов	Содержит все необходимые описания классов для использования программного продукта
DZ_Array.cpp	24,8 Кб, текстовый файл	Тестовая программа	Программа предназначена для проверки работоспособности системы классов и проведения приемно-сдаточных испытаний

DZ_LIB.cpp	19,1 Кб, текстовый файл	Описание методов классов и общих данных	Модуль библиотек методов
DZ.h	360 б, текстовый файл	Описание общих данных	Модель общих описаний
DZ_Array.exe	2,36 Мб, исполнимый файл	Тестовая программа	Содержит все необходимое для автономного выполнения в режиме командной строки.

4. УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММНОГО ПРОДУКТА

4.1. Требования к составу и параметрам технических средств

Данная система классов должна использоваться на компьютерах следующей конфигурации:

4.1.1. IBM-совместимый компьютер с процессором 80486 и выше;

4.1.2. Не менее 1 Мбайт свободной оперативной памяти;

4.1.3. VGA-совместимый видеоадаптер и монитор;

4.1.4. Стандартная клавиатура;

4.1.5. Свободного места на жёстком диске не менее 200 Кбайт.

4.2. Требования к информационной совместимости

Разрабатываемые классы и наследуемые от них классы не должны конфликтовать по именам с уже существующими классами в VS MS 2005 (библиотеки MFC, ATL, FCL и CRL).

4.3. Требования к программному обеспечению

4.3.1. Данная система классов предназначена для использования в программах, выполняемых на компьютере под управлением системы Microsoft Windows 2000 и выше. Использование разрабатываемой библиотеки требует наличия компилятора языка C++ и системы программирования (MS VS 2005).

4.4. Требования к условиям эксплуатации

4.4.1. Данная система классов должна эксплуатироваться совместно с языком программирования C++ в среде MS VS 2005/2008/2010. Для работы с данной системой классов программист должен быть знаком с навыками объектно-ориентированного программирования.

4.4.2.В остальном требования к эксплуатации точно такие же, как к программной реализации языка C++, используемой совместно с данной системой классов.

4.4.3.Программа тестового примера для проведения испытаний должна работать в среде компьютера, без установленной системы программирования MS VS 2005/2008/2010.

4.5. Требования к маркировке и упаковке

Программа предоставляется на дискете 3,5" или CD/DVD носителе.

4.6. Требования к хранению

Программа хранится на дискете 3,5" или CD/DVD носителе.

5. ОБЩИЕ ХАРАКТЕРИСТИКИ ПРОГРАММНОГО ПРОДУКТА

Система классов позволяет хранить списки домов на улицах.

Занимаемый объем на ЖД исходными текстами: 200 Кбайт

Занимаемый объем на ЖД для повторения сборки проекта: 45 Мбайт
Количество пользовательских классов: 2
Общее количество классов: 7

Язык программирования	C++
Компилятор, компоновщик	MS VS 2005/2008/2010
Файл проекта (*.vcproj)	DZ_Array.vcproj
Стандартные заголовочные файлы библиотеки MS VS 2005.	iostream, string.h, stdafx.h, conio.h

Московский государственный технический университет им. Н.Э.Баумана

УТВЕРЖДАЮ:

Большаков С.А.
"__"_____201X Г.

Комплексная лабораторная работа/ДЗ по дисциплине ПКШ

"Система классов улиц и домов"

Техническое описание

(вид документа)

писчая бумага
(вид носителя)

15
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-XX _____
Большаков С.А.
"__"_____201X г.

Москва - 201X

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММНОМ ОБЕСПЕЧЕНИИ

В данном документе дано техническое описание программного продукта: система классов для работы с улицами и домами. Шифр разработки программной системы – **DZ_OOP**. Техническое описание включает: описание модулей; описание классов, их методов и свойств; описание диаграммы классов; описание алгоритмов методов и процедур.

Данная система классов предназначена для решения задач, в которых необходимо учитывать объекты типа улица и дом с соответствующими свойствами. В частности система классов может обеспечивать решение задач: оценки ремонта домов и улиц, подсчета числа жителей, квартир и этажей в домах. Система классов предназначена для обеспечения удобной работы с этими объектами, высокий уровень надежности программ, функциональных возможностей, а также сокращение сроков разработки и реализации программных продуктов, где необходимо использовать подобные объекты.

2. МОДУЛЬНАЯ СТРУКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

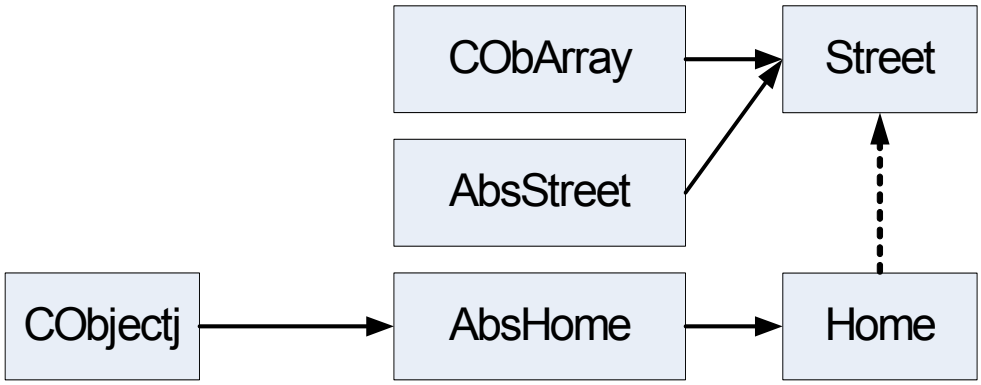
Система классов включена в один основной и один вспомогательный модуль, характеристики которых приведены в таблице, расположенной ниже.

Состав набора исходных файлов:

Название	Размер и тип	Описание	Примечание
DZ_Class.h	5,2 Кб, текстовый файл	Описания классов	Содержит все необходимые описания классов для использования программного продукта
DZ_Array.cpp	24,8 Кб, текстовый файл	Тестовая программа	Программа предназначена для проверки работоспособности системы классов и проведения приемно-сдаточных испытаний
DZ_LIB.cpp	19,1 Кб, текстовый файл	Описание методов классов и общих данных	Модуль библиотек методов
DZ.h	360 б, текстовый файл	Описание общих данных	Модель общих описаний
DZ_Array.exe	2,36 Мб, исполнимый файл	Тестовая программа	Содержит все необходимое для автономного выполнения в режиме командной строки.

3. ДИАГРАММА КЛАССОВ ПО

Ниже приставлена диаграмма классов системы классов улиц и домов разработанного программного обеспечения (ПО).



Классы имеют следующее назначение:

Класс **CObject** - системный абстрактный класс для наследования общих свойств объектов системы классов.

Класс **AbsStreet** - абстрактный класс, в котором учтены общие свойства улиц.

Класс **AbsHome** - абстрактный класс, в котором учтены общие свойства домов, размещаемых на улицах.

Класс **Street** - класс улиц, для создания объекта типа улица, позволяющий создавать объекты, учитывающие свойства: название, список домов, число домов и др.

Класс **Home** - класс дома, позволяющий создавать объекты описывающие дома, в которых учтены свойства: номер, число квартир, число жителей, признак ремонта и др.

4. ОПИСАНИЕ МЕТОДОВ И ДАННЫХ КЛАССОВ ПО

Общие данные и переменные

В программах, использующих систему классов улиц и домов, применяются следующие глобальные переменные и перечисления.

№	Описание данных и перечислений	Назначение	Параметры и значения
1.	enum BOOL {**false**=0 , **true**};	Логические значения	**false , true**
2.	static BOOL **DestructorDeleteObj**;	Глобальный Признак удаления объектов	**false , true**
3.	enum **HomeType**{**fast, multiple , complex**};	Тип дома	**Fast -** простой, **multiple** - многокорпусный, **complex -** сложный
4.	enum **StreetType**{**one, two , mo**re };	Тип улицы:	**one** - односторонняя, **two -** двухсторонняя, **mo**re - много полос

№	Описание данных и перечислений	Назначение	Параметры и значения
5.	enum **TypeAddDel** {**tail, head, Number ,NumbAfter, NumbCurrent, NumbBefore, AssbAfter, AssCurrent, AssBefore , createObj , nocreateObj , deleteObj, nodeleteObj** };	Режим добавления и удаления объектов	**head** - голова (начало), **tail** – хвост (конец), **Number –** номер (по номеру). **createObj** – новый объект **nocreateObj –** не создается **deleteObj -** удаляется **nodeleteObj –** не удаляется

КЛАСС ДОМОВ - Home

Класс Home. **Дом** – объекты данного типа содержат информацию о номере дома, этажности, числе жителей и квартир, необходимости ремонта дома и типе дома. Предусматривается возможность изменения параметров дома.

Данные и переменные класса Home

В классе **Home** объявлены следующие свойства доступные пользователю:

Переменная	Тип свойства	Защита	Назначение
Home_Number	char *	public	Номер дома (имя)
iHome	int	public	Номер дома числовой
EtagCount	int	public	Число этажей
MenCount	int	public	Число жителей
TypeHome	HomeType	public	Тип дома (перечисление: простой, многокорпусный, сложный)
NumbApartament	int	public	Число квартир
HomeRemont	BOOL	public	Признак необходимости ремонта дома

Конструкторы класса Home

В классе **Home** описаны следующие конструкторы:

№ п/п	Прототип	Тип возврата	Назначение /Параметры
1.	**Home**()	**Home &**	Нет
2.	**Home**(Home & H)	**Home &**	Новый на основе ссылки на объект типа дом: **Home &**
3.	**Home**(Home * pH)	**Home &**	Новый на основе указателя на объект типа дом: **Home ***

4.	**Home**(const char *HomName, const char *Number)	**Home &**	Новый с параметрами: **HomName** - имя **Number** - номер дома
5.	**Home**(const char *HomName, const char *Number, int Numb)	**Home &**	Новый с параметрами: **HomName** - имя дома, **Number** - номер дома **Numb** - номер для поиска
6.	**Home**(const char *HomName, const char *Number, int Numb, int Etag, int Men=0,HomeType Type = fast,int Apart=0)	**Home &**	Новый с параметрами: **HomName** - имя дома, **Number** - номер дома **Numb** - номер для поиска, **Etag** - этажность, **Men** - число жителей, **Type** – тип дома, **Apart** - число квартир

Методы класса Home

В таблице представлены методы класса **Home**.

№ п/п	Прототип	Тип возврата	Назначение	Параметры
1.	void **setName**(const char *HomName , const char *Number=NULL)	void	Установить имя дома и номер дома	**HomName** - имя дома и **Number** - номер дома
2.	const char ***getName**()	const char *	Получить имя дома	Нет
3.	const char ***getNumb**()	const char *	Получить номер дома	Нет
4.	int getNo()	int	Получить номер дома для поиска	Нет
5.	void **getParam**(int & iH, int & Etag ,int & Men ,HomeType & Type, int & Apart)	void	Получить параметры дома	**iH** -номер дома, **Etag** - этажность, **Men** - число жителей, **Type** – тип (значение типа дома выбирается из набора перечисления **HomeType** см. выше), **Apart** - число квартир

№ п/п	Прототип	Тип возврата	Назначение	Параметры
6.	void **setParam**(int iH, int Etag ,int Men ,HomeType Type, int Apart)	void	Задать новые параметры дома	**iH** -номер дома, **Etag** - этажность, **Men** - число жителей, **Type** – тип (значение типа дома выбирается из набора перечисления **HomeType** см. выше), **Apart** - число квартир
7.	void **setAllParam**(const char *HomName , const char *Number, int iH, int Etag , int Men ,HomeType Type, int Apart , BOOL rem = **false**)	void	Задать все новые параметры дома	**HomName** - имя дома, **Number** - имя дома для поиска **iH** -номер дома, **Etag** - этажность, **Men** - число жителей, **Type** – тип (значение типа дома выбирается из набора перечисления **HomeType** см. выше), **Apart** - число квартир rem -признак ремонта
8.	virtual void **printOn**(ostream & out)	void	Печать в стандартный поток	Out – тип ostream
9.	friend Home & **operator** +(Home & H1 , Home & H2)	Home &	Дружественная функция - операция	Складываются два дома
10.	Home **operator** =(Home & H);	Home	Операция	Перегрузка оператора присваивания домов
11.	~**Home**()	-	деструктор	

При сложении двух домов суммируются: число квартир, число жителей, имена домов, устанавливается признак ремонта по логике “ИЛИ”. Этажность определяется по числу этажей первого дома. Тип дома задается как сложный (**complex**).

КЛАСС Улиц - Street

Класс Street. Улица – объекты данного типа в упорядоченном виде содержат информацию о домах улице, названии улицы, типе улицы, соседних улицах (как резерв), необходимости ремонта улицы, числе домов, номера улицы (как резерв). Предусматривается возможность добавления домов на улицу и их удаления, изменения других параметров улицы.

Данные и переменные класса Street

В таблице приведены свойства класса улиц (**Street**).

Название	Тип свойства	Защита	Назначение
char ***Name**_Street;	char *	public	Название улицы
int **NumberStreet**;	int	public	Номер улицы

Название	Тип свойства	Защита	Назначение
int **Homes_num**;	int	public	Число домов на улице
BOOL **Remont**;	BOOL	public	Признак необходимости ремонта домов улицы
BOOL **RemontStreet**;	BOOL	public	Признак ремонта самой улицы
StreetType **StrType**;	StreetType	public	Тип улицы: **one (**односторонняя**)**, **two (**два направления**)** , **more (**много полос**)**
Street * **ListOfNear**;	Street *	public	Список соседних улиц (зарезервировано)

Конструкторы класса Street

Ниже в таблице приведен список конструкторов класса **Street**.

№ п/п	Прототип	Тип возврата	Назначение /Параметры
1.	**Street**();	**Street &**	Нет
2.	**Street**(const char *sName);	**Street &**	Создание улицы с названием
3.	**Street**(const char *sNumbSearch, const char *sName);	**Street &**	Создание улицы с названием и именем для поиска
4.	**Street**(int Num);	**Street &**	Создание улицы с номером
5.	**Street**(const char *sName , int Num);	**Street &**	Создание улицы с именем и номером
6.	**Street**(Street & S);	**Street &**	Создание улицы на основе другой (на основе ссылки)

Методы класса Street

№ п/п	Прототип	Тип возврата	Назначение	Параметры
1.	void **add**(Home *pH, TypeAddDel T= tail , int Numb = 1 , TypeAddDel TC = createObj);	void	Добавление дома на улицу	**pH** - указатель на дом **T** - куда добавить (head, tail, Number), **TC** - создавать ли новый (createObj, nocreateObj)

№ п/п	Прототип	Тип возврата	Назначение	Параметры
2.	void **del**(Home *pH , TypeAddDel T= tail , int Numb = 1 , TypeAddDel TD= nodeleteObj);	void	Удаления дома с улицы	**pH** - указатель на дом куда выбирается **T** - куда добавить (head, tail, Number), **TD** - удалять ли объект (deleteObj, nodeleteObj)
3.	virtual void **printOn**(ostream & out);	void	Печать объекта улицы в стандартный поток	**out** - ostream стандартный поток
4.	int **GetNumberHome**();	int	Получить число домов на улице	нет
5.	int **GetNumberMens**();	int	Получить число жителей на улице	нет
6.	int **GetNumberApart**();	int	Получить число квартир на улице	нет
7.	char * **GetNameStreet**();	char *	Получить название улицы	нет
8.	char * **GetKeyNameStreet**();	char *	Получить номер дома символьный	нет
9.	int **GetNumbStreet**(){ return NumberStreet;};	int	Получить номер дома числовой	нет
10.	int **GetKeyNumbStreet**();	int	Получить номер дома числовой для поиска	нет
11.	void **SetNameStreet**(const char * NameStr);	void	Установить название улицы	**NameStr** - имя улицы
12.	void **SetKeyNameStreet**(const char * sName);	void	Установить имя улицы для поиска	**sName** - имя улицы для поиска
13.	void **SetNumbStreet**(int n);	void	Установить номер улицы	**n** - номер улицы
14.	void **SetKeyNumbStreet**(int n);	void	Установить номер улицы для поиска	**n** - номер улицы для поиска
15.	BOOL **GetRemont**();	BOOL	Получить признак ремонта домов на улице	нет

№ п/п	Прототип	Тип возврата	Назначение	Параметры
16.	BOOL **GetRemontStr**();	BOOL	Получить признак ремонта улицы	нет
17.	void **SetRemontStr**(BOOL rS) ;	void	Установить признак ремонта улицы (**false**, **true**)	**rS** - признак ремонта улицы
18.	StreetType **GetStreetType**();	StreetType	Получить тип улицы: **one** (односторонняя), **two** (два направления) , **more** (много полос)	нет
19.	void **SetStreetType** (StreetType t);	void	Установить тип улицы	**t** – новый тип улицы: **one**, **two** , **more**
20.	friend Street & **operator** +(Street & X , Street & Y);	Street &	Дружественная функция	Операция сложения двух улиц
21.	Street **operator** =(Street & S);	Street	Операция	Перегрузка операции присваивания двух улиц
22.	~**Street**() { };		Деструктор	

5. ДАННЫЕ И ФАЙЛЫ ДАННЫХ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В системе классов домов и улиц не формируется отдельных файлов. Структуры данных и перечисления для задания параметров приведены в таблице предыдущего раздела (“Общие описания ”).

6. ОСНОВНЫЕ АЛГОРИТМЫ МЕТОДОВ КЛАССОВ ПО

Некоторые важные алгоритмы системы классов представлены ниже на рисунках.

6.1. Алгоритм вычисления признака ремонта.

Алгоритм вычисления признака ремонта домов улицы (этот параметр задается в объекте улица отдельно) приведен ниже (функция GetRemont() – метод класса Street):

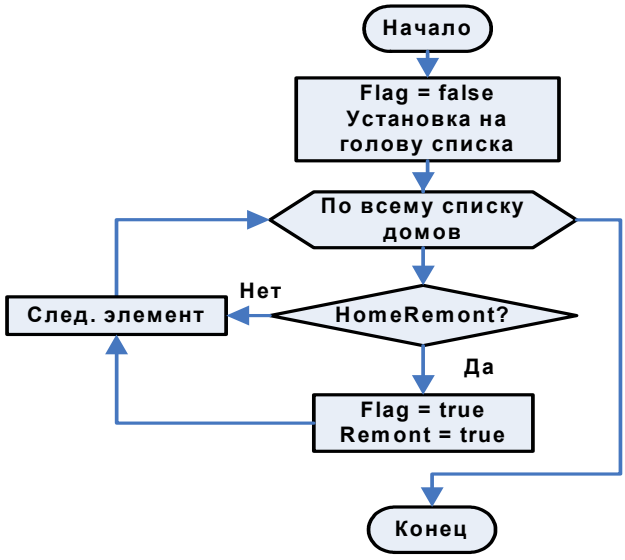


Рис. 1 Вычисление признака ремонта домов улицы

Для вычисления признака ремонта домов улицы выполняется просмотр всех домов из списка и , при наличии, хотя бы одного признака ремонта у одного из домов признак ремонта (Remont) устанавливается в истину. Функция в этом случае возвращает тоже значение истина.

6.2. Алгоритм сложения домов.

Алгоритм сложения двух домов (H1 и H2) приведен ниже (перегруженная с помощью внешней функции операция сложения – дружественная классу Home):

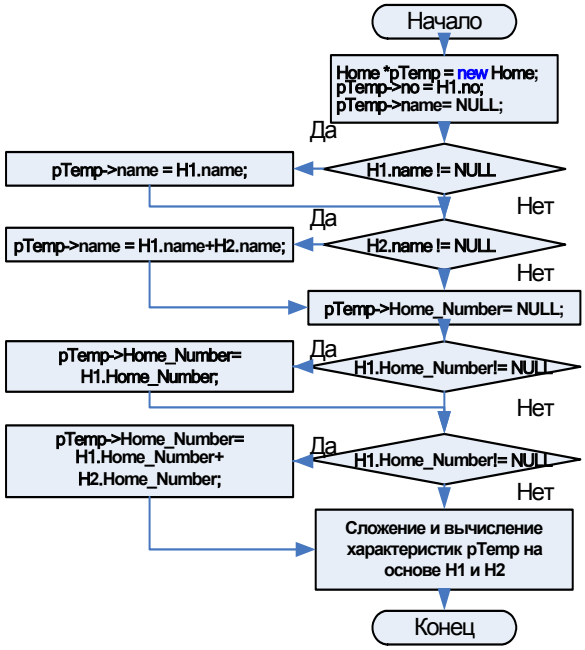


Рис. 2 Сложение двух домов

Сначала проверяются значения полей с именами, после чего выполняется конкатенация имен для номера дома и поиска. Другие характеристики домов, кроме этажности, суммируются.

6.3. Алгоритм сложения двух улиц.

Алгоритм сложения двух улиц (X и Y) приведен ниже (перегрузка выполнена внешней дружественной классу Street операцией):

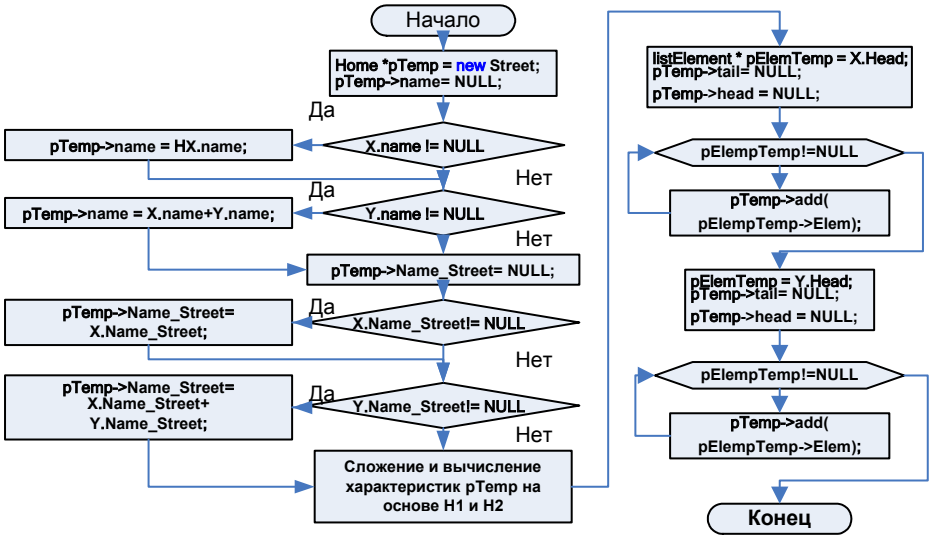


Рис. 3 Сложение двух улиц

Первоначально выполняется проверка и формирование названия улицы, как сложение двух имен. Кроме того, аналогичным образом вычисляется имя для поиска в списке. Далее выполняются два цикла добавления на новую улицу всех домов из первой улицы (X) и второй улицы (H).

6.4. Алгоритм перегрузки операции присваивания улиц.

Алгоритм перегрузки операции присваивания двух улиц (текущей – this и из параметра S) приведен ниже (перегрузка выполнена внутренней для класса **Street** операцией "="):

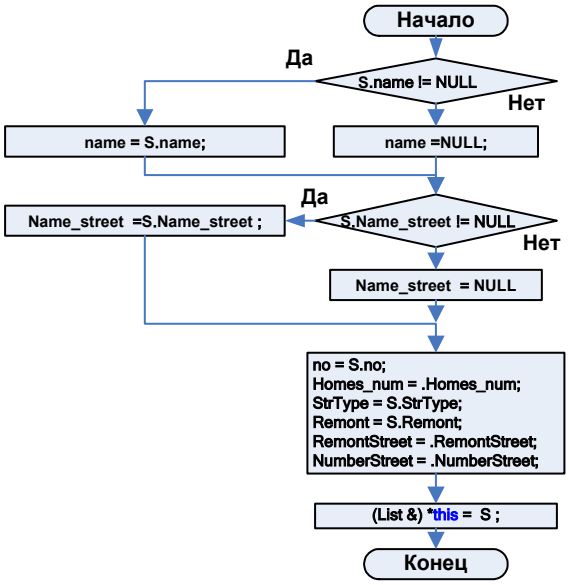


Рис. 4 Перегрузка оператора присваивания улиц

В данном алгоритме первоначально выполняется определение параметров name (имя для поиска улиц в списках) и Name_Street (название улиц). Затем присваиваются все параметры улицы. Для копирования списков домов используется перегруженный оператор присваивания списков, который вызывается с помощью следующего присваивания:

(List &) *this = S ;

Явное приведение к типу ссылки на список является в данном случае необходимым.

6.5. Алгоритм очистки списка.

Алгоритм очистки списка (метод класса List - ClearList) приведен ниже:

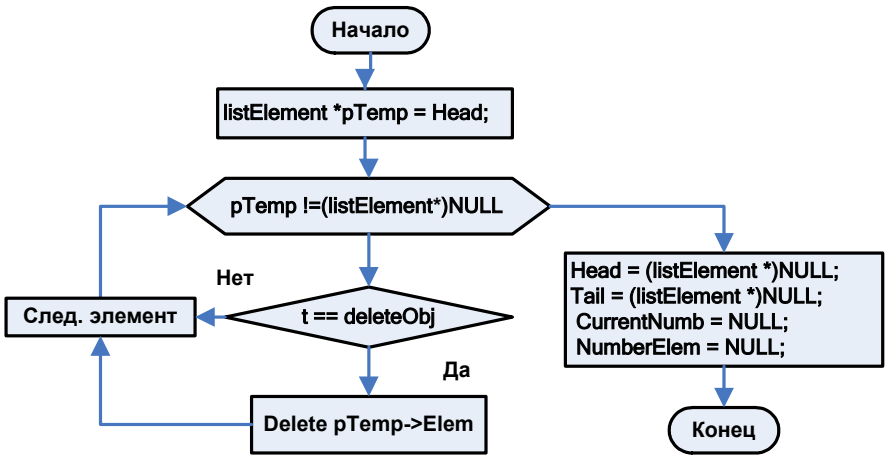
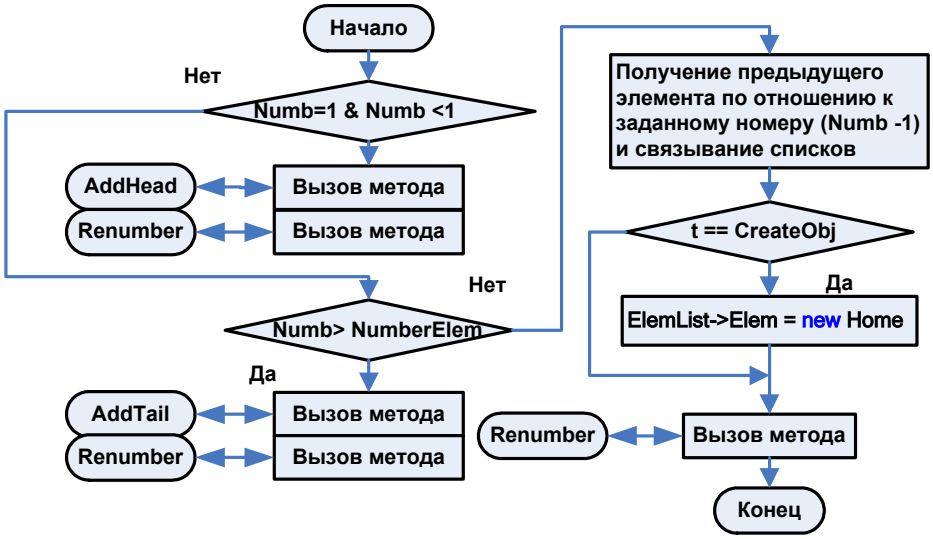


Рис. 5 Очистка списка

6.6. Алгоритм добавление дома по номеру.

Алгоритм добавления дома по номеру (метод класса List - AddNumber) приведен на



рисунке расположенном ниже:

Рис. 6 Добавление дома по номеру

В этом алгоритме после любого вызова метода нижнего уровня добавления (AddHead, AddTail и связывания списков) выполняется новая нумерация списка(Renember). Если параметр t (тип - TypeAddDel) имеет значение CreateObj, то создается новый объект.

7. ОПИСАНИЕ ПРОЦЕДУР И ФУНКЦИЙ ПО

Система классов улиц и домов содержит две процедуры являющиеся дружественными функциями классов Home и Street

Класс	**Прототип**	**Параметры**	**Назначение**	**Примечания**
Класс Home	friend Home & **operator** +(Home & H1 , Home & H2)	Ссылки на дома (H1 и H2), подлежащие объединению	Складываются два дома	При сложении домов объединяются: их символьные названия, вычисляются все характеристики нового дома: число жителей и квартир, признак ремонта. Число этажей устанавливается по первому дому
Класс **Street**	friend Street & **operator** +(Street & X , Street & Y);	Ссылки на улицы (H1 и H2), подлежащие объединению	Операция сложения двух улиц	При сложении улиц имена их складываются, списки складываются, вычисляются: признаки ремонта улиц и домов на улице и типы улиц

8.ОПИСАНИЕ ПРОЦЕССА ОТЛАДКИ КЛАССОВ.

В процессе отладки были проверены все методы и свойства. Для этого был разработан специальный тестовый пример, листинг которого включен в документ, содержащий исходные тексты программ и комплект поставки программного продукта. Отладка проводилась стандартными средствами MS VS.

9. КЛАССЫ И МЕТОДЫ, ПЕРЕОПРЕДЕЛЯЕМЫЕ В ПО

В данном программном обеспечении никакие стандартные классы и методы стандартных классов не переопределяются.

Московский государственный технический университет им. Н.Э.Баумана

УТВЕРЖДАЮ:

Большаков С.А.

"__"_____201X Г.

Комплексная лабораторная работа/ДЗ по дисциплине ПКШ

"Система классов улиц и домов"

Программа и методика испытаний

(вид документа)

писчая бумага
(вид носителя)

19
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-XX _____
Большаков С.А.

"__"_____201X г.

Москва - 201X

СОДЕРЖАНИЕ

1. ОБЪЕКТ ИСПЫТАНИЙ

Объектом испытаний является система классов строк переменной длины и массивов строк, в дальнейшем называемая просто «система классов» или «строки». Данный программный продукт разработан для того, чтобы обеспечить пользователю (программисту) необходимые средства для различной работы со строками.

2. ЦЕЛЬ ИСПЫТАНИЙ

Целью проведения испытаний является проверка правильности работы всех указанных в техническом задании функций системы классов.

3. СОСТАВ ПРЕДЪЯВЛЯЕМОЙ ДОКУМЕНТАЦИИ

3.1.При сдаче домашнего задания предъявляются следующие документы:

3.1.1. Техническое задание
3.1.2. Описание применения программного продукта;
3.1.3. Техническое описание программного продукта;
3.1.4. Руководство пользователя;
3.1.5. Руководство системного программиста;
3.1.6. Исходные тексты программ системы классов и тестового примера;
3.1.7. Программа и методика испытаний;
3.1.8. Описание тестового примера;

3.2.При проведении испытаний предъявляются документы:

3.2.1. Техническое задание
3.2.2. Описание тестового примера
3.2.3. Программа и методика испытаний
3.2.4. Исходные тексты программ системы классов и тестового примера

4.ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ К ИСПЫТАНИЯМ И УСЛОВИЯ ПРОВЕДЕНИЯ ИСПЫТАНИЯ

4.1.Требования к условиям проведения испытаний.

4.1.1. Испытания проводятся на основе тестового примера, который должен поставляться вместе с системой классов.

4.1.2. Тестовый пример должен работать и в операционной среде, где не инсталлирована система программирования (MS VS).

4.1.3. Тестовый пример должен работать в режиме эмуляции командной строки (cmd.exe), при этом вывод и ввод информации должен быть обеспечен на русском языке.

4.1.4. При необходимости и наличии системы программирования (MS VS), сборка проекта тестового примера должна быть выполнена без ошибок и предупреждений.

4.2. Требования к техническим средствам.

4.2.1. Используемая операционная система: Windows XP/Win7/8.
4.2.2. Компилятор C++: MS VS 2005 и выше.
4.2.3.Технические характеристики оборудования, на котором проводятся испытания:
4.2.3.1. IBM-совместимый компьютер с процессором Intel Pentium II 400 MHz не ниже.
4.2.3.2. Более 300 килобайт свободной обычной оперативной памяти.
4.2.3.3. SVGA видеоадаптер и монитор.
4.2.3.4. Не менее 500 килобайт на диске.
4.2.3.5. Клавиатура со 101-ой клавишей.

5.ПОРЯДОК ПРОВЕДЕНИЯ ИСПЫТАНИЙ

5.1.Состав и структура технических и программных средств.

5.1.1. Запуск программы тестового примера выполняется в режиме командной строки (cmd.exe) или при запуске программы из любого файл менеджера.

5.1.2. Запуск режима командной строки производиться из главного меню ОС: ПУСК-> ВЫПОЛНИТЬ-> cmd.exe.

5.1.3. Программа для испытаний запускается с: дискеты, CD/DVD диска или другого сменного носителя информации (карта памяти или флеш сменный накопитель). Для запуска программы из текущей директории вводится: DZ_Array.EXE.

5.2. Последовательность испытаний системы классов.

№ п/п	№ пункта Т.З.	Выполняемые действия	Ожидаемый результат	Примечание
1.	Запуск тестового примера	В текущем каталоге, где находиться программа ввести: **DZ_OOP.EXE** и нажать **Enter**	Первоначально выводится меню тестового примера (Рис. 1).	
2.	5.1.1 Создание улиц с домами	Ввести **1**, нажать **Enter,** нажать **Enter**	Первоначально выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 2 в разделе 6 данного документа.	
3.	5.1.2 Создание объектов для домов улицы	Ввести **2**, нажать **Enter,** нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 3 в разделе 6 данного документа.	
4.	5.1.3 Создание объектов для домов улицы на основе других	Ввести **3**, нажать **Enter,** нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 4 в разделе 6 данного документа.	
5.	5.1.4 Учет свойств дома(см. ТЗ)	Ввести **4**, нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 5 в разделе 6 данного документа.	
6.	5.1.5 Задание и получение характеристик дома	Ввести **5**, нажать **Enter,** нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 6 в разделе 6 данного документа.	
7.	5.1.6 Сложение двух домов	Ввести **6**, нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 7 в разделе 6 данного документа.	
8.	5.1.7 Перегрузить оператор присваивания для домов	Ввести **7**, нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 8 в разделе 6 данного документа.	

№ п/п	№ пункта Т.З.	Выполняемые действия	Ожидаемый результат	Примечание
9.	5.1.8 Распечатка характеристик дома	Ввести **8**, нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 9 в разделе 6 данного документа.	
10.	5.1.9 Учет свойств улицы(см. ТЗ)	Ввести **9**, нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 10 в разделе 6 данного документа.	
11.	5.1.10 Распечатка содержания улицы и ее свойств	Ввести **10**, нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 11 в разделе 6 данного документа.	
12.	5.1.11 Задание характеристик улицы	Ввести **11**, нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 12 в разделе 6 данного документа.	
13.	5.1.12 Получение характеристик улицы	Ввести **12**, нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 13 в разделе 6 данного документа.	
14.	5.1.13 Сложение двух улиц	Ввести **13**, нажать **Enter,** нажать **Enter,** нажать **Enter,** нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 14 в разделе 6 данного документа.	
15.	5.1.14 Добавление дома на улицу	Ввести **14**, нажать **Enter,** нажать **Enter,** нажать **Enter,** нажать **Enter,** нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 15 в разделе 6 данного документа.	
16.	5.1.15 Удаление дома с улицы	Ввести **15**, нажать **Enter,** нажать **Enter,** нажать **Enter,** нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 16 в разделе 6 данного документа.	
17.	5.1.16 Установка и снятие признака ремонта улицы	Ввести **16**, нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 17 в разделе 6 данного документа.	

№ п/п	№ пункта Т.З.	Выполняемые действия	Ожидаемый результат	Примечание
18.	5.1.17 Автоматическое получение признака ремонта домов улицы	Ввести **17**, нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 18 в разделе 6 данного документа.	
19.	5.1.18 Перегрузка оператора присваивания для улиц	Ввести **18**, нажать **Enter,** нажать **Enter,** нажать **Enter**	Первоначально на экран выводится меню тестового примера (Рис. 1). Результаты показаны на Рис. 19 в разделе 6 данного документа.	
20.	Завершение работы тестового примера	Ввести **0**, нажать **Enter**	Программа тестового примера завершает свою работу	Сообщений не выдается

6.Результаты испытаний по пунктам ТЗ

6.1. Рисунок 1. Меню тестового примера

Ниже представлен рисунок 1 (рис.1) с меню тестового примера:

Меню тестового примера системы классов улиц.

1. ТЗ - 5.1.1 Создание улиц с домами
2. ТЗ - 5.1.2 Создание объектов для домов улицы
3. ТЗ - 5.1.3 Создание объектов для домов улицы на основе других
4. ТЗ - 5.1.4 Учет свойств дома(см. ТЗ)
5. ТЗ - 5.1.5 Задание и получение характеристик дома
6. ТЗ - 5.1.6 Сложение двух домов
7. ТЗ - 5.1.7 Перегрузить оператор присваивания для домов
8. ТЗ - 5.1.8 Распечатка характеристик дома
9. ТЗ - 5.1.9 Учет свойств улицы(см. ТЗ)
10. ТЗ - 5.1.10 Распечатка содержания улицы и ее свойств
11. ТЗ - 5.1.11 Задание характеристик улицы
12. ТЗ - 5.1.12 Получение характеристик улицы
13. ТЗ - 5.1.13 Сложение двух улиц
14. ТЗ - 5.1.14 Добавление дома на улицу
15. ТЗ - 5.1.15 Удаление дома с улицы
16. ТЗ - 5.1.16 Установка и снятие признака ремонта улицы
17. ТЗ - 5.1.17 Автоматическое получение признака ремонта домов улицы
18. ТЗ - 5.1.18 Перегрузка оператора присваивания для улиц

0.Выход

6.2. Рисунок 2. Создание улиц с домами

Ниже представлен рисунок 2 без меню тестового примера (рис.1). Для сокращения текста меню тестового примера не повторяется:

```
1
ТЗ - 5.1.1  Создание улиц с домами
{{{*******
Улица -  Ленинский проспект  Ключ для поиска -  Ленинский проспект
Номер улицы -  0  Номер для поиска -  0
Число домов на улице -  0  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Список List пуст
}
*******}}}
{{{*******
Улица -  Ленинский проспект  Ключ для поиска -  Ленинский проспект
Номер улицы -  0  Номер для поиска -  0
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
```

6.3. Рисунок 3. Создание объектов для домов улицы

```
2
5.1.2  Создание объектов для  домов улицы
Номер сп. -0  Имя не задано
  Символьное имя не задано
  Номер -0
Этажей -0  Жителей -0
Тип дома - простой   Ремонт не нужен!   Число квартир - 0

Номер сп. -0  Имя - Жилой
Номер сим. -д.2
  Номер -0
Этажей -0  Жителей -0
Тип дома - простой   Ремонт не нужен!   Число квартир - 0

Номер сп. -0  Имя - Жилой
Номер сим. -д.3
  Номер -3
Этажей -0  Жителей -0
Тип дома - простой   Ремонт не нужен!   Число квартир - 0

Номер сп. -0  Имя - Жилой
Номер сим. -д.4а
  Номер -4
Этажей -2  Жителей -0
Тип дома - простой   Ремонт не нужен!   Число квартир - 0


Номер сп. -0  Имя - ДЭЗ
Номер сим. -д.5
  Номер -5
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 0

Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
```

```
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 0

Номер сп. -0  Имя - Магазин
Номер сим. -д.7
  Номер -7
Этажей -2  Жителей -3
Тип дома - много строений   Ремонт не нужен!   Число квартир - 5
```

6.4. Рисунок 4. Создание объектов для домов улицы на основе других

```
3
5.1.3  Создание объектов для  домов улицы на основе других
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 100

Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 100


Указатель!!!
Номер сп. -0  Имя - Магазин
Номер сим. -д.7
  Номер -7
Этажей -2  Жителей -3
Тип дома - много строений   Ремонт не нужен!   Число квартир - 5

Номер сп. -0  Имя - Магазин
Номер сим. -д.7
  Номер -7
Этажей -2  Жителей -3
Тип дома - много строений   Ремонт не нужен!   Число квартир - 5
```

6.5. Рисунок 5. Учет свойств дома(см. ТЗ)

```
4
5.1.4  Учет свойств дома(см. ТЗ)
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 100

Номер сп. -0  Имя - Магазин
Номер сим. -д.7
  Номер -7
Этажей -2  Жителей -3
Тип дома - много строений   Ремонт не нужен!   Число квартир - 5
```

6.6. Рисунок 6. Задание и получение характеристик дома

```
5
5.1.5  Задание и получение характеристик дома
Номер сп. -0  Имя - Жилой
```

```
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 100

Номер -6  Этажей -2  Жителей -3
Тип дома - простой   Число квартир - 100
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -11
Этажей -12  Жителей -13
Тип дома - простой   Ремонт не нужен!   Число квартир - 15


Признак ремонта!!!
Номер сп. -0  Имя - Магазин
Номер сим. -д.7
  Номер -7
Этажей -2  Жителей -3
Тип дома - много строений   Ремонт не нужен!   Число квартир - 5

Номер сп. -0  Имя - Аптека
Номер сим. -10/8
  Номер -1
Этажей -2  Жителей -3
Тип дома - простой   Требуется ремонт!   Число квартир – 5
```

6.7. Рисунок 7. Сложение двух домов

```
6
5.1.6  Сложение двух домов
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 3

Номер сп. -0  Имя - Ашан
Номер сим. -д.7
  Номер -9
Этажей -10  Жителей -11
Тип дома - много строений   Ремонт не нужен!   Число квартир - 5

Номер сп. -0  Имя - Жилой + Ашан
Номер сим. -д.6 + д.7
  Номер -6
Этажей -2  Жителей -14
Тип дома - сложный   Ремонт не нужен!   Число квартир – 8
```

6.8. Рисунок 8. Перегрузить оператор присваивания для домов

```
7
5.1.7  Перегрузить оператор присваивания для домов
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 3

Номер сп. -0  Имя - Жилой
Номер сим. -д.6
```

```
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 3

Номер сп. -0  Имя - Перекресток
  Символьное имя не задано
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 3

Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир – 3
```

6.9. Рисунок 9. Распечатка характеристик дома

```
8
5.1.8  Распечатка характеристик дома
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 3
```

6.10. Рисунок 10. Учет свойств улицы (см. ТЗ)

```
9
5.1.9  Учет свойств улицы (см. ТЗ)
{{{*******
Улица -  Улица  Ключ для поиска -  Улица
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  0  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Список List пуст
}
*******}}}
{{{*******
Улица -  Улица  Ключ для поиска -  Улица
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
```

6.11. Рисунок 11. Распечатка содержания улицы и ее свойств

```
10
5.1.10 Распечатка содержания улицы и ее свойств
{{{*******
Улица -  Улица  Ключ для поиска -  Улица
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  0  Улица отремонтирована.
Все эти дома отремонтированы.
```

```
Список {
Список List пуст
}
*******}}}
{{{*******
Улица -  Улица  Ключ для поиска -  Улица
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
```

6.12. Рисунок 12. Задание характеристик улицы

```
11
5.1.11 Задание характеристик улицы
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
******* Изменения параметров *********
{{{*******
Улица -  Новая  Ключ для поиска -  Новая ключ
Номер улицы -  33  Номер для поиска -  77
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
******* Параметры *********
Название улицы ->  Новая
Номер улицы ->  33
Название улицы для поиска->  Новая ключ
Номер улицы для поиска->  77
Число домов на улице = 3
Число жителей на улице = 3
Число квартир на улице = 5
На улице не нужен ремонт домов!
Тип улицы -> двухсторонне  движение
```

6.13. Рисунок 13. Получение характеристик улицы

```
12
5.1.12 Получение характеристик улицы
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
```

```
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
******* Параметры *********
Название улицы ->  Улица с параметрами
Номер улицы ->  15
Название улицы для поиска->  Улица с параметрами
Номер улицы для поиска->  15
Число домов на улице = 3
Число жителей на улице = 3
Число квартир на улице = 5
На улице не нужен ремонт домов!
Тип улицы -> двухсторонне  движение
```

6.14. Рисунок 14. Сложение двух улиц

```
13
5.1.13 Сложение двух улиц
{{{*******
Улица -  Первая  Ключ для поиска -  Первая
Номер улицы -  20  Номер для поиска -  20
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название ДЭЗ
Номер - 2 Название Магазин
Номер - 3 Название Жилой
}
*******}}}
{{{*******
Улица -  Вторая   Ключ для поиска -  Вторая
Номер улицы -  20  Номер для поиска -  20
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Аптека
Номер - 2 Название Перекресток
Номер - 3 Название Детский сад
}
*******}}}


******* Сложение *********
{{{*******
Улица -      Ключ для поиска -
Номер улицы -  20  Номер для поиска -  20
Число домов на улице -  0  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Список List пуст
}
*******}}}
{{{*******
Улица -  Первая + Вторая   Ключ для поиска -  Первая + Вторая
Номер улицы -  20  Номер для поиска -  0
```

```
Число домов на улице -  6  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название ДЭЗ
Номер - 2 Название Магазин
Номер - 3 Название Жилой
Номер - 4 Название Аптека
Номер - 5 Название Перекресток
Номер - 6 Название Детский сад
}
*******}}}

******* Параметры *********
Название улицы ->  Первая + Вторая
Номер улицы ->  20
Название улицы для поиска->  Первая + Вторая
Номер улицы для поиска->  0
Число домов на улице = 6
Число жителей на улице = 6
Число квартир на улице = 10
На улице не нужен ремонт домов!
Тип улицы -> двухсторонне  движение
```

6.15. Рисунок 15. Добавление дома на улицу

```
14
5.1.14 Добавление дома на улицу
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}

{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  4  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Перекресток
Номер - 2 Название Аптека
Номер - 3 Название Жилой
Номер - 4 Название Магазин
}
*******}}}

{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  5  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Перекресток
Номер - 2 Название Аптека
Номер - 3 Название Жилой
Номер - 4 Название Магазин
```

```
Номер - 5 Название Детский сад
}
*******}}}

{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  6  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Перекресток
Номер - 2 Название Жилой 3
Номер - 3 Название Аптека
Номер - 4 Название Жилой
Номер - 5 Название Магазин
Номер - 6 Название Детский сад
}
*******}}}
```

6.16. Рисунок 16. Удаление дома с улицы

```
15
5.1.15 Удаление дома с улицы
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  6  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Перекресток
Номер - 2 Название Жилой 3
Номер - 3 Название Аптека
Номер - 4 Название Жилой
Номер - 5 Название Магазин
Номер - 6 Название Детский сад
}
*******}}}

Удаление дома с улицы конец !!!
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  5  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Перекресток
Номер - 2 Название Жилой 3
Номер - 3 Название Аптека
Номер - 4 Название Жилой
Номер - 5 Название Магазин
}
*******}}}

Удаление дома с улицы начало!!!
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  4  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой 3
```

```
Номер - 2 Название Аптека
Номер - 3 Название Жилой
Номер - 4 Название Магазин
}
*******}}}

Удаление дома с улицы второго!!!
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой 3
Номер - 2 Название Жилой
Номер - 3 Название Магазин
}
*******}}}
```

6.17. Рисунок 17. Установка и снятие признака ремонта улицы

```
16
5.1.16 Установка и снятие признака ремонта улицы
{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
Улице не нужен ремонт!
После установки!!!!
Улице нужен ремонт!
После снятия признака ремонта!!!!
Улице не нужен ремонт!
```

6.18. Рисунок 18. Автоматическое получение признака ремонта домов улицы

```
17
5.1.17 Автоматическое получение признака ремонта домов улицы
До установки признака ремонта дома и вычисления признака ремонта домов улицы!!!!

{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
На улице не нужен ремонт домов!
Номер сп. -0  Имя - Магазин
Номер сим. -д.3
  Номер -1
Этажей -2  Жителей -3
```

```
Тип дома - простой   Требуется ремонт!   Число квартир - 3

После вычисления признака ремонта домов улицы!!!!
{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Нужен ремонт домов улицы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
На улице нужен ремонт домов!
```

6.19. Рисунок 19. Перегрузка оператора присваивания для улиц

```
18
5.1.18 Перегрузка оператора присваивания для улиц
{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Нужен ремонт домов улицы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
{{{*******
Улица -  Улица  Ключ для поиска -  Улица
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  0  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Список List пуст
}
*******}}}
{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Нужен ремонт домов улицы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}

После изменения S1 (название и удален первый)!!!!
S1!!!!
{{{*******
Улица -  Новое название S1   Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  1  Улица отремонтирована.
Нужен ремонт домов улицы.
```

```
Список {
Номер - 1 Название Магазин
}
*******}}}
SNew!!!!
{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Нужен ремонт домов улицы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
```

Московский государственный технический университет им. Н.Э.Баумана

УТВЕРЖДАЮ:

Большаков С.А.
"__"_____201X Г.

Комплексная лабораторная работа/ДЗ по дисциплине ПКШ

"Система классов улиц и домов"

Руководство системного программиста

(вид документа)

писчая бумага
(вид носителя)

7
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-XX _____
Большаков С.А.
"__"_____201X г.

Москва - 201X

СОДЕРЖАНИЕ

1. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММЫ

Система классов домов и улиц предназначена для автоматизации работы с объектами улиц и домов в программных проектах и предметной области, где необходимо это учитывать. В частности, система классов предназначена для программирования задач включающих действия: оценки ремонта домов и улиц, подсчета числа жителей, квартир и этажей в домах. Система классов обеспечивает удобную работу с этими объектами, высокий уровень надежности программ, функциональных возможностей, а также сокращение сроков разработки и реализации программных продуктов, где необходимо использовать подобные объекты.

Данная система классов должна эксплуатироваться совместно системой программирования C++ (MS VS 2005/2008/2010). Для работы с данной системой классов может программист с основами и навыками объектно-ориентированного программирования.

В остальном требования к эксплуатации и условия к применению точно такие же, как к программной реализации языка C++, используемой совместно с данной системой классов.

2.ТРЕБОВАНИЯ К СОСТАВУ И ПАРАМЕТРАМ ТЕХНИЧЕСКИХ СРЕДСТВ

Данная система классов должна использоваться на компьютерах следующей конфигурации:

2.1. IBM-совместимый компьютер с процессором 80486 и выше;
2.2. Не менее 1 Мбайт свободной оперативной памяти;
2.3. VGA/EGA-совместимый видеоадаптер и монитор;
2.4. Стандартная клавиатура;
2.5. Свободного места на жёстком диске не менее 200 Кбайт.

3. ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

Данный программный продукт требует наличия русификатора дисплея для корректного отображения символов русского алфавита или ОС обеспечивающей переключение кодовой страницы для русского шрифта.

Данная система классов должна эксплуатироваться совместно системой программирования C++ (MS VS 2005/2008/2010).

В остальном требования к программному обеспечению точно такие же как к программной реализации языка C++, используемой совместно с данной системой классов.

4. СОСТАВ ПРОГРАММНОГО ПРОДУКТА

Программный продукт включает три основных модуля:

Название	Размер и тип	Описание	Примечание
DZ_Class.h	5,2 Кб, текстовый файл	Описания классов	Содержит все необходимые описания классов для использования программного продукта
DZ_Array.cpp	24,8 Кб, текстовый файл	Тестовая программа	Программа предназначена для проверки работоспособности системы классов и проведения приемно-сдаточных испытаний
DZ_LIB.cpp	19,1 Кб, текстовый файл	Описание методов классов и общих данных	Модуль библиотек методов

Название	Размер и тип	Описание	Примечание
DZ.h	360 б, текстовый файл	Описание общих данных	Модель общих описаний
DZ_Array.exe	2,36 Мб, исполнимый файл	Тестовая программа	Содержит все необходимое для автономного выполнения в режиме командной строки.

5. УСТАНОВКА ПРОГРАММНОГО ПРОДУКТА

Для того, чтобы установить данный программный продукт на компьютер следует произвести следующие действия:

1. Освободить на жестком диске как минимум 200 килобайт свободного пространства.
2. Вставить дискету или CD/DVD носитель с дистрибутивом программного продукта в дисковод или CD/DVD устройство.
3. Скопировать файлы. С помощью системной функции копирования файлы (или файл менеджера), которые перечислены в разделе 4 этого документа, с носителя на жесткий диск в нужные каталоги. Для испытаний любой доступный каталог (DZ_Array.exe). Для подключения в новый проект (**DZ_LIB.cpp –** исходные модули**, DZ_Class.h, DZ.h** – заголовочные модули) в общий каталог исходных и заголовочных файлов или в специально созданный каталог для новой сборки тестового примера (**DZ_Array.cpp, DZ_LIB.cpp, DZ_Class.h, DZ.h**).
4. Прописать в установках каталогов для используемой на компьютере системы программирования C++ полный пути к директории, куда был установлен программный продукт (DZ_LIB.cpp, DZ_Class.h, DZ.h).

6. УДАЛЕНИЕ ПРОГРАММНОГО ПРОДУКТА

Чтобы удалить данный программный продукт, надо с помощью системной функции удаления (или файл менеджера) удалить файлы, перечисленные в пункте 4 данного документа, из директорий, куда они были скопированы.

7. ЗАПУСК ПРОГРАММЫ

Для запуска программы тестового примера необходимо перейти в каталог, в который Вы скопировали данный программный продукт (см. пункт 5 этого документа) и запустить файл DZ_Array.exe.

8. ЗАВЕРЕШНИЕ РАБОТЫ ПРОГРАММЫ

Для завершения работы программы тестового примера необходимо выполнить пункт меню тестового примера, в котором предлагается завершить программу. В критических ситуациях можно принудительно завершить программу, нажав на крестик в правом верхнем углу окна.

9. СООБЩЕНИЙ ОБ ОШИБКАХ И ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ

В системе классов улиц и домов не предусмотрена обработка исключительных ситуаций и выдача диагностических сообщений. В программах, разработанных с включением этой системы классов, может быть предусмотрена обработка исключительных ситуаций и выдача сообщений об ошибках. Выдаваемые системные диагностические сообщения и возникающие системные исключительные ситуации описаны в документации на систему программирования (MS VS 2005).

10. УСЛОВИЯ ЭКСПЛУАТАЦИИ ПРОГРАММНОГО ПРОДУКТА

Требования к условиям эксплуатации программного продукта описаны в п. 5.3 документе “Техническое задание”.

11. ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО ПРОДУКТА

Проверка работоспособности системы классов выполняется на основе документа “Программа и методика испытаний” и специальной программы тестового примера (DZ_Array.exe). В отдельных случаях оговоренных с заказчиком может быть продемонстрирована сборка программного продукта на основе тестового примера, для чего предоставляется среда системы программирования (MS VS 2005/2008/2010).

12. ПОДГОТОВКА К РАБОТЕ С ПРОГРАММНЫМ ПРОДУКТОМ

Для работы с системой классов в отдельных проектах необходимо прописать пути размещения файла с описаниями классов (DZ_LIB.cpp, DZ_Class.h, DZ.h), либо, что нежелательно, скопировать файл в каталог нового проекта и подключить его в исходный текст с помощью директивы include. В более старших версиях системы программирования MS VS, можно установить зависимости между подключаемыми файлами и основными модулями.

13. ОБЩИЕ ХАРАКТЕРИСТИКИ ПРОГРАММНОГО ПРОДУКТА

Система классов позволяет хранить списки домов на улицах.

Занимаемый объем на ЖД исходными текстами: 200 Кбайт
Занимаемый объем на ЖД для повторения сборки проекта: 45 Мбайт
Количество пользовательских классов: 2
Общее количество классов: 7

Язык программирования	C++
Компилятор, компоновщик	MS VS 2005/2008/2010
Файл проекта (*.vcproj)	DZ_LIB.vcproj
Стандартные заголовочные файлы библиотеки MS VS 2005.	iostream, string.h, stdafx.h, conio.h

Московский государственный технический университет им. Н.Э.Баумана

УТВЕРЖДАЮ:

Большаков С.А.
"__"_____201X Г.

Комплексная лабораторная работа/ДЗ по дисциплине ПКШ

"Система классов улиц и домов"

Руководство пользователя

(вид документа)

писчая бумага
(вид носителя)

7
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-XX _____
Большаков С.А.
"__"_____201X г.

Москва - 201X

СОДЕРЖАНИЕ

1. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММЫ

Система классов домов и улиц предназначена для автоматизации работы с объектами улиц и домов в программных проектах и предметной области, где необходимо это учитывать. В частности, система классов предназначена для программирования задач включающих действия: оценки ремонта домов и улиц, подсчета числа жителей, квартир и этажей в домах. Система классов обеспечивает удобную работу с этими объектами, высокий уровень надежности программ, функциональных возможностей, а также сокращение сроков разработки и реализации программных продуктов, где необходимо использовать подобные объекты.

2. ПОДКЛЮЧЕНИЕ СИСТЕМЫ КЛАССОВ И ДИАГРАММА КЛАССОВ

Для того, чтобы установить данную систему классов на компьютер следует произвести следующие действия:

1. Освободить на жестком диске как минимум 200 килобайт свободного пространства.
2. Вставить дискету или CD/DVD носитель с дистрибутивом программного продукта в дисковод или CD/DVD устройство.
3. Скопировать файлы. С помощью системной функции копирования файлы (или файл менеджера), которые перечислены в разделе 4 этого документа, с носителя на жесткий диск в нужные каталоги. Для испытаний любой доступный каталог (DZ_2kurs.exe). Для подключения в новый проект (DZ_2kurs.h) в общий каталог заголовочных файлов или в специально созданный каталог для новой сборки тестового примера (DZ_2kurs.cpp, DZ_2kurs.h).
4. Прописать в установках каталогов для используемой на компьютере системы программирования C++ полный пути к директории, куда был установлены модели системы классов (DZ_2kurs.h).

3. РАБОТА С СИСТЕМОЙ КЛАССОВ

3.1. Общие сведения о системе классов улиц и домов

Система классов описывает дома и улицы для разработки программ, в которых учет сведений об этих объектах необходим.

Дом – объекты данного типа содержат информацию о номере дома, этажности, числе жителей и квартир, необходимости ремонта дома и типе дома. Предусматривается возможность изменения параметров дома.

Улица – объекты данного типа в упорядоченном виде содержат информацию о домах улице, названии улицы, типе улицы, соседних улицах (как резерв), необходимости ремонта улицы, числе домов, номера улицы (как резерв). Предусматривается возможность добавления домов на улицу и их удаления, изменения других параметров улицы.

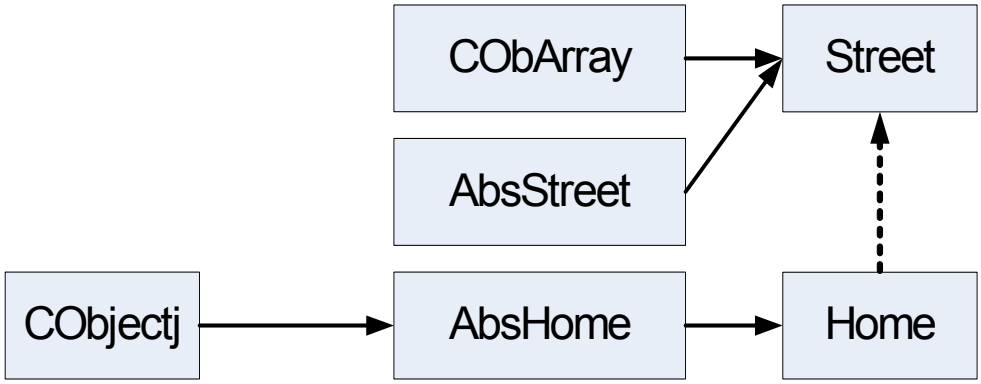
Система классов позволяет программировать следующие операции и функции:

- Создание и задание свойств объектов типа дом и улица;
- Установка признака ремонта домов;
- Сложение двух домов;
- Получение параметров домов (номера, числа жителей, числа квартир, типа дома);
- Установки и изменения параметров дома (номера, числа жителей, числа квартир, типа дома);

- Удаление объектов типа дом и улица;
- Распечатки свойств объектов типа дом и улица;
- Добавление домов на конкретную улицу;
- Удаление дома с конкретной улицы;
- Сложение двух улиц (при объединении улиц);
- Переименование улиц и новую нумерацию домов;
- Распечатки списка домов улицы с их характеристиками;
- Получение параметров улицы (названия, числа жителей, числа квартир, типа улицы);
- Установки и изменения параметров улицы (названия, числа жителей, числа квартир, типа улицы);
- Автоматическое получение признаков необходимости ремонта домов на улице.

3.2. Диаграмма классов улиц и домов

Ниже приставлена диаграмма классов системы классов улиц и домов программного обеспечения (ПО).



Классы имеют следующее назначение:

Класс **CObject** - системный абстрактный класс для наследования общих свойств объектов системы классов.

Класс **AbsStreet** - абстрактный класс, в котором учтены общие свойства улиц.

Класс **AbsHome** - абстрактный класс, в котором учтены общие свойства домов, размещаемых на улицах.

Класс **Street** - класс улиц, для создания объекта типа улица, позволяющий создавать объекты, учитывающие свойства: название, список домов, число домов и др.

Класс **Home** - класс дома, позволяющий создавать объекты описывающие дома, в которых учтены свойства: номер, число квартир, число жителей, признак ремонта и др.

Примечание 1. В данном руководстве описаны только те классы и их составляющие (методы и свойства), которые необходимы пользователю для реализации своих задач. Это классы **Street** и **Home**. Описание других составляющих системы классов улиц и домов можно найти в документах: “Техническое описание” и “Описание тестового примера”.

Примечание 2. Результаты работы фрагментов текста программ можно найти в документах: “Программа и методика испытаний” и “Описание тестового примера”.

3.3. Подключение системы классов в программу

В данном фрагменте подключаются заголовочные файлы библиотек: ввода вывода (**iostream**), стандартных классов MFC (**stdafx.h**) и собственной системы классов улиц и домов (**DZ_2kurs.h**). Кроме того, объявлено использование именованного пространство стандартных функций (**std**). В проект подключается модуль - DZ_LIB.cpp.

```
#include "stdafx.h"
#include "DZ_Class.h"
#include <iostream>
using namespace std;
```

3.4. Общие данные и переменные

В программах, использующих систему классов улиц и домов применяются следующие глобальные переменные и перечисления.

№	Описание данных и перечислений	Назначение	Параметры и значения
6.	enum BOOL {**false**=0 , **true**};	Логические значения	**false , true**
7.	static BOOL **DestructorDeleteObj**;	Глобальный Признак удаления объектов	**false , true**
8.	enum **HomeType**{**fast, multiple , complex**};	Тип дома	**Fast -** простой, **multiple** - многокорпусный, **complex -** сложный
9.	enum **StreetType**{**one, two , mo**re };	Тип улицы:	**one** - односторонняя, **two -** двухсторонняя, **mo**re - много полос
10.	enum **TypeAddDel** {**tail, head, Number ,NumbAfter, NumbCurrent, NumbBefore, AssbAfter, AssCurrent, AssBefore , createObj , nocreateObj , deleteObj, nodeleteObj** };	Режим добавления и удаления объектов	**head** - голова (начало), **tail** – хвост (конец), **Number –** номер (по номеру). **createObj** – новый объект **nocreateObj –** не создается **deleteObj -** удаляется **nodeleteObj –** не удаляется

4. КЛАСС ДОМОВ - Home

Класс Home. **Дом** – объекты данного типа содержат информацию о номере дома, этажности, числе жителей и квартир, необходимости ремонта дома и типе дома. Предусматривается возможность изменения параметров дома.

4.1. Данные и переменные класса Home

В классе **Home** объявлены следующие свойства доступные пользователю:

Переменная	Тип свойства	Защита	Назначение
Home_Number	char *	public	Номер дома (имя)
iHome	int	public	Номер дома числовой
EtagCount	int	public	Число этажей
MenCount	int	public	Число жителей
TypeHome	HomeType	public	Тип дома (перечисление: простой, многокорпусный, сложный)
NumbApartament	int	public	Число квартир
HomeRemont	BOOL	public	Признак необходимости ремонта дома

4.2. Конструкторы класса Home

В классе **Home** описаны следующие конструкторы:

№ п/п	Прототип	Тип возврата	Назначение /Параметры
1.	**Home**()	**Home &**	Нет
2.	**Home**(Home & H)	**Home &**	Новый на основе ссылки на объект типа дом: **Home &**
3.	**Home**(Home * pH)	**Home &**	Новый на основе указателя на объект типа дом: **Home ***
4.	**Home**(const char *HomName, const char *Number)	**Home &**	Новый с параметрами: **HomName** - имя **Number** - номер дома
5.	**Home**(const char *HomName, const char *Number, int Numb)	**Home &**	Новый с параметрами: **HomName** - имя дома, **Number** - номер дома **Numb** - номер для поиска
6.	**Home**(const char *HomName, const char *Number, int Numb, int Etag, int Men=0,HomeType Type = fast,int Apart=0)	**Home &**	Новый с параметрами: **HomName** - имя дома, **Number** - номер дома **Numb** - номер для поиска, **Etag** - этажность, **Men** - число жителей, **Type** – тип дома, **Apart** - число квартир

Примеры использования конструкторов:

Конструктор без параметров:

```
Home H1;
```

Конструктор с именем и номером символьным:

```
Home H2("Жилой","д.2");
```

Конструктор с именем, номером символьным и номером для поиска:

```
Home H3("Жилой", "д.3", 3);
```

Конструктор с именем, номером символьным, номером для поиска и этажностью:

```
Home H4("Жилой", "д.4а", 4,2);
```

Конструктор с именем, номером символьным, номером для поиска, этажностью и числом жителей:

```
Home H5("ДЭЗ", "д.5", 5,2,3);
```

Конструктор с именем, номером символьным, номером для поиска, этажностью, числом жителей и типом дома:

```
Home H6("Жилой", "д.6", 6,2,3, fast);
```

Конструктор с именем, номером символьным, номером для поиска, этажностью, числом жителей, типом дома и числом квартир:

```
Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
```

Примеры использования конструкторов копирования:

Конструктор копирования на основе ссылки:

```
Home H6("ДЭЗ", "д.5", 5,2,3);
Home Test(H5);
H5.printOn(cout);
Test.printOn(cout);
```

Конструктор копирования на основе ссылки и динамической памяти:

```
Home *pHome = new Home (H6);
H6.printOn(cout);
pHome->printOn(cout);
delete pHome;
```

Конструктор копирования на основе ссылки и операции присваивания “-”:

```
Home Temp;
  Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
  Temp = H7;
H7.printOn(cout);
Temp.printOn(cout);
```

Конструктор копирования на основе указателя и динамической памяти:

```
Home H6("ДЭЗ", "д.5", 5,2,3);
Home *pHome = new Home (H6);
Home *pHome1 = new Home (* pHome);
pHome->printOn(cout);
pHome1->printOn(cout);
delete pHome;
delete pHome1;
```

4.3. Деструктор класса Home

Прототип деструктора:

~**Home**();

Пример явного и неявного использования деструктора:

```
{
        Home *pHome = new Home (H6);
        pHome->printOn(cout);
        delete pHome;
 }
```

Неявный вызов деструктора производится при завершении блока операторов . Явный вызов деструктора производится при выполнении операции **delete**.

4.4. Методы класса Home

В таблице представлены методы класса **Home**.

№ п/п	Прототип	Тип возврата	Назначение	Параметры
1.	void **setName**(const char *HomName , const char *Number=NULL)	void	Установить имя дома и номер дома	**HomName** - имя дома и **Number** - номер дома
2.	const char ***getName**()	const char *	Получить имя дома	Нет
3.	const char ***getNumb**()	const char *	Получить номер дома	Нет
4.	int getNo()	int	Получить номер дома для поиска	Нет
5.	void **getParam**(int & iH, int & Etag ,int & Men ,HomeType & Type, int & Apart)	void	Получить параметры дома	**iH** -номер дома, **Etag** - этажность, **Men** - число жителей, **Type** – тип (значение типа дома выбирается из набора перечисления **HomeType** см. выше), **Apart** - число квартир
6.	void **setParam**(int iH, int Etag ,int Men ,HomeType Type, int Apart)	void	Задать новые параметры дома	**iH** -номер дома, **Etag** - этажность, **Men** - число жителей, **Type** – тип (значение типа дома выбирается из набора перечисления **HomeType** см. выше), **Apart** - число квартир
7.	void **setAllParam**(const char *HomName , const char *Number, int iH, int Etag , int Men ,HomeType Type, int Apart , BOOL rem = **false**)	void	Задать все новые параметры дома	**HomName** - имя дома, **Number** - имя дома для поиска **iH** -номер дома, **Etag** - этажность, **Men** - число жителей, **Type** – тип (значение типа дома выбирается из набора перечисления **HomeType** см. выше), **Apart** - число квартир rem -признак ремонта
8.	virtual void **printOn**(ostream & out)	void	Печать в стандартный поток	Out – тип ostream

Примеры использования методов класса **Home**:

Установка имени дома символьного:

```
Temp.setName("Университет");
Temp.printOn(cout);
```

Установка имени и номера дома символьного:

```
Temp.setName("Магазин" ,"д.11/12");
Temp.printOn(cout);
```

Получение имени и номера дома символьного:

```
cout << "Name дома = "<< Temp.getName() << endl;
```

Получение номера дома символьного:

```
cout << "Номер дома = "<< Temp.getNumb() << endl;
```

Получение номера дома для поиска:

```
cout << "Номер в списке = "<< Temp.getNo() << endl;
```

Получение параметров дома:

```
cout << "Параметры: "<< endl;
int  iH,  Etag , Men ,  Apart ;
HomeType  Type;
Temp.getParam(iH , Etag, Men, Type , Apart);
 cout << "Номер -" << iH <<
  "  Этажей -" << Etag <<  "  Жителей -" << Men <<     endl;
      if (Type == fast)
         cout << "Тип дома - простой ";
      if (Type == multiple)
         cout << "Тип дома - много строений ";
      if (Type == complex)
         cout << "Тип дома - сложный ";
      cout << "  Число квартир - " << Apart <<  endl;
```

Установка параметров дома:

```
Temp.setParam(1,2,3,fast, 5 );
Temp.printOn(cout);
```

Установка всех параметров дома:

```
Temp.setAllParam("Аптека", "10/8 кв.3", 11,12,13,fast, 15, true );
Temp.printOn(cout);
```

4.5. Операции класса Home

Прототип перегруженного оператора присваивания:

Home operator =(Home & H);

Пример использования перегруженного оператора присваивания:

```
Home Temp;
 Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
 Temp = H7;
```

```
H7.printOn(cout);
Temp.printOn(cout);
```

4.6. Дружественные функции класса Home

Прототип перегруженного оператора сложения:
friend Home & **operator** +(Home & H1 , Home & H2)

Пример использования перегруженного оператора сложения:

```
Home H1("Жилой", "д.6", 6,2,3, fast , 3);
Home H2("Жилой", "д.7", 9,10,11, fast , 5);
Home Temp;
Temp = H1 + H2;
H1.printOn(cout);
H2.printOn(cout);
Temp.printOn(cout);
```

При сложении двух домов суммируются: число квартир, число жителей, имена домов, устанавливается признак ремонта по логике “ИЛИ”. Этажность определяется по числу этажей первого дома. Тип дома задается как сложный (**complex**).

5. КЛАСС Улиц - Street

Класс Street. Улица – объекты данного типа в упорядоченном виде содержат информацию о домах улице, названии улицы, типе улицы, соседних улицах (как резерв), необходимости ремонта улицы, числе домов, номера улицы (как резерв). Предусматривается возможность добавления домов на улицу и их удаления, изменения других параметров улицы.

5.1. Данные и переменные класса Street

В таблице приведены свойства класса улиц (**Street**).

Название	Тип свойства	Защита	Назначение
char ***Name**_Street;	char *	public	Название улицы
int **NumberStreet**;	int	public	Номер улицы
int **Homes_num**;	int	public	Число домов на улице
BOOL **Remont**;	BOOL	public	Признак необходимости ремонта домов улицы
BOOL **RemontStreet**;	BOOL	public	Признак ремонта самой улицы
StreetType **StrType**;	StreetType	public	Тип улицы: **one (**односторонняя**)**, **two (**два направления**)** , **more (**много полос**)**
Street * **ListOfNear**;	Street *	public	Список соседних улиц (зарезервировано)

5.2. Конструкторы класса Street

Ниже в таблице приведен список конструкторов класса **Street**.

№ п/п	Прототип	Тип возврата	Назначение /Параметры
7.	**Street**();	**Street &**	Нет
8.	**Street**(const char *sName);	**Street &**	Создание улицы с названием
9.	**Street**(const char *sNumbSearch, const char	**Street &**	Создание улицы с на-

	*sName);		званием и именем для поиска
10.	**Street**(int Num);	**Street &**	Создание улицы с номером
11.	**Street**(const char *sName , int Num);	**Street &**	Создание улицы с именем и номером
12.	**Street**(Street & S);	**Street &**	Создание улицы на основе другой (на основе ссылки)

Примеры конструкторов класса **Street**:
Конструктор без параметров:

```
Street S1;
```

Конструктор с именем улицы:

```
Street S2("Лениский проспект");
```

Конструктор с именем улицы и именем для поиска:

```
Street S3("Ленин","Лениский проспект");
```

Конструктор с номером улицы для поиска:

```
Street S4(5);
```

Конструктор с именем улицы и номером для поиска:

```
Street S5("Горького ул." , 7);
```

Конструктор копирования на основе ссылки на объект:

```
Home H2("Жилой","д.2", 7,2,3, fast , 5);
Home H3("Магазин", "д.3", 3);
Home H4("ДЭЗ", "д.4а", 4,2);
Street SNew("Улица" , 15);
SNew.add(&H2);
SNew.add(&H3);
SNew.add(&H4);
SNew.printOn(cout);
Street SCopy(SNew);
SCopy.printOn(cout);
```

5.3. Деструктор класса Street

Прототип деструктора улицы:
~**Street**();
Пример явного и неявного использования деструктора:

```
{
        Street *pStreet = new Street (S1);
        pStreet ->printOn(cout);
        delete pStreet;
}
```

Неявный вызов деструктора производится при завершении блока операторов. Явный вызов деструктора производится при выполнении операции **delete**.

5.4. Методы класса Street

№ п/п	Прототип	Тип возврата	Назначение	Параметры
1.	void **add**(Home *pH, TypeAddDel T= tail , int Numb = 1 , TypeAddDel TC = createObj);	void	Добавление дома на улицу	**pH** - указатель на дом **T** - куда добавить (head, tail, Number), **TC** - создавать ли новый (createObj, nocreateObj)
2.	void **del**(Home *pH , TypeAddDel T= tail , int Numb = 1 , TypeAddDel TD= nodeleteObj);	void	Удаления дома с улицы	**pH** - указатель на дом куда выбирается **T** - куда добавить (head, tail, Number), **TD** - удалять ли объект (deleteObj, nodeleteObj)
3.	virtual void **printOn**(ostream & out);	void	Печать объекта улицы в стандартный поток	**out** - ostream стандартный поток
4.	int **GetNumberHome**();	int	Получить число домов на улице	нет
5.	int **GetNumberMens**();	int	Получить число жителей на улице	нет
6.	int **GetNumberApart**();	int	Получить число квартир на улице	нет
7.	char * **GetNameStreet**();	char *	Получить название улицы	нет
8.	char * **GetKeyNameStreet**();	char *	Получить номер дома символьный	нет
9.	int **GetNumbStreet**(){ return NumberStreet;};	int	Получить номер дома числовой	нет
10.	int **GetKeyNumbStreet**();	int	Получить номер дома числовой для поиска	нет
11.	void **SetNameStreet**(const char * NameStr);	void	Установить название улицы	**NameStr** - имя улицы
12.	void **SetKeyNameStreet**(const char * sName);	void	Установить имя улицы для поиска	**sName** - имя улицы для поиска

№ п/п	Прототип	Тип возврата	Назначение	Параметры
13.	void **SetNumbStreet**(int n);	void	Установить номер улицы	**n** - номер улицы
14.	void **SetKeyNumbStreet**(int n);	void	Установить номер улицы для поиска	**n** - номер улицы для поиска
15.	BOOL **GetRemont**();	BOOL	Получить признак ремонта домов на улице	нет
16.	BOOL **GetRemontStr**();	BOOL	Получить признак ремонта улицы	нет
17.	void **SetRemontStr**(BOOL rS) ;	void	Установить признак ремонта улицы (**false**, **true**)	**rS** - признак ремонта улицы
18.	StreetType **GetStreetType**();	StreetType	Получить тип улицы: **one** (односторонняя), **two** (два направления) , **more** (много полос)	нет
19.	void **SetStreetType**(StreetType t);	void	Установить тип улицы	**t** – новый тип улицы: **one**, **two** , **more**

Примеры использования методов класса Street:

Добавление домов на улицу в начало:

```
Home H2("Жилой","д.2", 7,2,3, fast , 5);
Home H3("Магазин", "д.3", 3);
Home H4("ДЭЗ", "д.4а", 4,2);
Street Astreet("Добавление head", 20);
Astreet.printOn(cout);
cout << endl << "******* Добавление head *********" <<  endl;
Astreet.add(&H2 , head);
Astreet.add(&H3 , head);
Astreet.add(&H4 , head);
Astreet.printOn(cout);
```

Добавление домов на улицу в конец списка:

```
Home H5("Жилой","д.2", 7,2,3, fast , 5);
Home H6("Магазин", "д.3", 3);
Home H7("ДЭЗ", "д.4а", 4,2);
cout << endl << "******* Добавление tail *********" <<  endl;
Street Dstreet("Добавление tail ", 20);
Dstreet.add(&H5 , tail);
Dstreet.add(&H6 , tail);
Dstreet.add(&H7 , tail);
Dstreet.printOn(cout);
```

Добавление домов на улицу по номеру (второго):

```
cout << endl << "******* Добавление NUM 2 *********" <<  endl;
 Home H8("Жилой 3","д.2", 70,20,30, fast , 50);
 Home H9("Магазин 3", "д.3", 3);
 Home H10("ДЭЗ 3", "д.4а", 4,2);
 Dstreet.add(&H8 , Number, 2);
 Dstreet.add(&H9 , Number, 2);
 Dstreet.add(&H10 , Number, 2);
 Dstreet.printOn(cout);
```

Удаление домов с улицы (с конца):

```
cout << endl << "******* Удаление *********" <<  endl;
 Dstreet.del(&H8);
```

Удаление домов с улицы (с начала):

```
 Dstreet.printOn(cout);
 Dstreet.del(&H8 , head);
 Dstreet.printOn(cout);
```

Удаление домов с улицы (по номеру - второго):

```
 Dstreet.del(&H8 , Number, 2);
 Dstreet.printOn(cout);
```

Получение параметров улицы:

```
cout <<  "******* Параметры *********" <<  endl;
cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
cout     <<     "Название     улицы     для     поиска->     "     <<
Sumstreet.GetKeyNameStreet() << endl;
cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() <<
endl;
cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
```

Получение признака ремонта домов улицы:

```
if ( Sumstreet.GetRemont() )
   cout <<  "На улице нужен ремонт домов!" << endl;
else
   cout <<  "На улице нужен ремонт домов!" << endl;
 if ( Sumstreet.GetStreetType() == one  )
  cout <<  "Тип улицы -> одностороннее движение" <<  endl;
   if ( Sumstreet.GetStreetType() == two  )
  cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;
```

Задание параметров улицы:

```
 cout <<  "******* Изменения параметров *********" <<  endl;
Sumstreet.SetNameStreet("Новая");
Sumstreet.SetKeyNameStreet("Новая ключ");
Sumstreet.SetNumbStreet( 33 );
Sumstreet.SetKeyNumbStreet( 77 );
 cout <<  "Название улицы 2->  " << Sumstreet.GetNameStreet() << endl;
 cout <<  "Номер улицы 2->  " << Sumstreet.GetNumbStreet() << endl;
```

```
        cout << "Название улицы для поиска 2-> " <<
Sumstreet.GetKeyNameStreet() << endl;
        cout << "Номер улицы для поиска 2-> " <<
Sumstreet.GetKeyNumbStreet() << endl;
        Sumstreet.printOn(cout);
            };
```

5.5. Операции класса Street

Прототип оператора присваивания улиц:
Street **operator** =(Street & S);

Пример применения перегруженного оператора присваивания улиц:

```
Home H2("Жилой","д.2", 7,2,3, fast , 5);
Home H3("Магазин", "д.3", 3);
H3.setAllParam("Магазин", "д.3", 1,2,3,fast, 3 , true);
Street S1("Улица 1" , 15);
Street SNew("Улица" , 15);
S1.add(&H2);
S1.add(&H3);
S1.printOn(cout);
SNew.printOn(cout);
  SNew = S1;
  SNew.printOn(cout);
 S1.del(&H3 , head);
 getchar();
S1.SetNameStreet("Новое название S1 ");
cout << "После изменения S1 (название и удален первый)!!!!" << endl;
cout << "S1!!!!" << endl;
S1.printOn(cout);
cout << "SNew!!!!" << endl;
  SNew.printOn(cout);
```

Создаются две улицы **S1**(со всеми параметрами и двумя домами **H2** и **H3**) и **SNew**(с минимумом параметров). Далее выполняется операция присваивания домов –“=”. Исходные улицы и улица **SNew** после присваивания распечатывается. Далее улица **S1** изменяется – из нее удаляется один дом H3. После этого обе улицы снова распечатываются (**printOn**). После этого **S1** изменяется, а **SNew** остается неизменной. При сложении улиц складываются все параметры улиц, включая и имена. Списки улиц объединяются. Формируется новый тип улицы и устанавливаются признаки ремонта домов улицы и ремонта самой улицы.

5.6. Дружественные функции класса Home

Прототип оператора сложения улиц:
friend Street & **operator** +(Street & X , Street & Y);

Пример применения перегруженного оператора сложения улиц:

```
cout << endl << "******* Создание улиц *********" << endl;
Home H2("Жилой","д.2", 7,2,3, fast , 5);
Home H3("Магазин", "д.3", 3);
Home H4("ДЭЗ", "д.4а", 4,2);
```

```
Street Astreet("Первая", 20);
 Astreet.add(&H2 , head);
 Astreet.add(&H3 , head);
 Astreet.add(&H4 , head);
Astreet.printOn(cout);
 Home H5("Аптека","д.2", 7,2,3, fast , 5);
 Home H6("Перекресток", "д.3", 3);
 Home H7("Детский сад", "д.4а", 4,2);
Street Dstreet("Вторая ", 20);
 Dstreet.add(&H5 , tail);
 Dstreet.add(&H6 , tail);
 Dstreet.add(&H7 , tail);
 Dstreet.printOn(cout);
    getchar();
  cout << endl << "******* Сложение *********" <<  endl;
  Street Sumstreet("  ", 20);
  Sumstreet.printOn(cout);
  Sumstreet = Astreet + Dstreet;
  Sumstreet.printOn(cout);
  getchar();
```

Создаются две улицы: **Astreet** с домами **H2**, **H3** и **H4,** добавленными в голову (**head**) методом (**add**), и **Dstreet** с домами **H5**, **H6** и **H7,** добавленными в хвост (**tail**) методом (**add**). Описывается улица **Sumstreet** и формируется как сумма с помощью перегруженной операции “+”. Исходные улицы и результирующая улица распечатываются стандартным методом **printOn**. Далее распечатываются параметры улицы с помощью специальных методов (см. п.п. 2.14).

6. ОТКЛЮЧЕНИЕ СИСТЕМЫ КЛАССОВ

Чтобы отключить данную систему классов, надо с помощью системной функции удаления (или файл менеджера) удалить файлы, перечисленные в пункте 4 данного документа, из каталогов , куда они были скопированы.

7. СООБЩЕНИЙ ОБ ОШИБКАХ И ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ

В системе классов улиц и домов не предусмотрена обработка исключительных ситуаций и выдача диагностических сообщений. В программах, разработанных с включением этой системы классов, может быть предусмотрена обработка исключительных ситуаций и выдача сообщений об ошибках. Выдаваемые системные диагностические сообщения и возникающие системные исключительные ситуации описаны в документации на систему программирования (MS VS 2005).

Московский государственный технический университет им. Н.Э.Баумана

УТВЕРЖДАЮ:

Большаков С.А.
"__"_____201X Г.

Комплексная лабораторная работа/ДЗ по дисциплине ПКШ

"Система классов улиц и домов"

Описание тестового примера

(вид документа)

писчая бумага
(вид носителя)

7
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-XX _____
Большаков С.А.
"__"_____201X г.

Москва - 201X

СОДЕРЖАНИЕ

1. ОПИСАНИЕ НАЗНАЧЕНИЯ ТЕСТОВОГО ПРИМЕРА

Данный тестовый пример предназначен для демонстрации возможностей системы классов улицы и дома и проведения приемно-сдаточных испытаний программного продукта. Исходные текст тестового примера приведен в документе "**Исходные тексты программ**". В документе "**Программа и методика испытаний**" показан порядок использования данной программы для проверки выполнения "**Технического задания**". Детальное описание классов и их методов дано в документе "Руководство пользователя". Порядок установки программного продукта для выполнения тестового приведен в документе "**Руководство системного программиста**". Данная программа является консольным приложением: вывод производится в окно эмуляции консоли, а ввод осуществляется с клавиатуры.

Имя исходного модуля тестового примера - **DZ_Array.cpp**.
Имя исходного модуля подключаемой системы классов - **DZ_LIB.cpp, DZ_Class.h, DZ.h.**

Примечание для студентов: Результаты работы фрагментов программ тестового примера можно не приводить. Можно сослаться на рисунки (Разделы) документа "Программа и методика испытаний", где эти результаты должны быть приведены обязательно.

2. ПОЯСНЕНИЕ КОДА ПРОГРАММЫ ТЕСТОВОГО ПРИМЕРА

2.1.Первоначальные описания тестового примера

2.1.1. Пояснения текста фрагмента

В данном фрагменте подключаются заголовочные файлы библиотек: ввода вывода (**iostream**), консольного ввода (**conio.h**), стандартных классов MFC (**stdafx.h**) и собственной системы классов улиц и домов (**DZ_Class.h, DZ.H**). Модуль **DZ_LIB.cpp** подключается в состав исходных модулей проекта. Кроме того, объявлено использование именованного пространство стандартных функций (**std**).

2.1.2. Текст фрагмента программы, а в проект должен быть включен модуль DZ_LIB.cpp

```
#include "stdafx.h"
#include " DZ_Class.h "
#include <iostream>
#include <conio.h>

using namespace std;
```

2.2.Структура главной программы

2.2.1. Пояснения текста фрагмента

В фрагменте показана структура главной функции программы (**main**). Описана переменная для ввода и управления переключателем (**iPunkt**). Построен цикл с переключателем и предварительным выводом меню проверки пунктов ТЗ. Для обеспечения корректного вывода на консоль русских символов системной функцией переключается кодовая страница (**chcp 1251>nul**), а перед каждым выводом меню (в цикле **while**) производится

очистка экрана консольного окна (**CLS**). Вывод строк меню выполняется построчно в стандартный поток (**cout**).

2.2.2. Текст фрагмента программы

```
int main()
{
int iPunkt;

system (" chcp 1251>nul ");
     while ( true ) {
system (" CLS ");
cout << endl << "Меню  тестового примера для сдачи системы классов улиц." << endl
<< endl;
// Новое меню
   cout << endl << "1.  Т3 - 5.1.1  Создание улиц с домами " <<  endl;
   cout <<  "2.  Т3 - 5.1.2  Создание объектов для  домов улицы  " <<  endl;
   cout <<  "3.  Т3 - 5.1.3  Создание объектов для  домов улицы на основе других " <<
endl;
   cout <<  "4.  Т3 - 5.1.4  Учет свойств дома(см. Т3) " <<  endl;
   cout <<  "5.  Т3 - 5.1.5  Задание и получение характеристик дома " <<  endl;
   cout <<  "6.  Т3 - 5.1.6  Сложение двух домов " <<  endl;
   cout <<  "7.  Т3 - 5.1.7  Перегрузить оператор присваивания для домов " <<  endl;
   cout <<  "8.  Т3 - 5.1.8  Распечатка характеристик дома " <<  endl;
   cout <<  "9.  Т3 - 5.1.9  Учет свойств улицы(см. Т3) " <<  endl;
   cout <<  "10. Т3 - 5.1.10 Распечатка содержания улицы и ее свойств " <<  endl;
   cout <<  "11. Т3 - 5.1.11 Задание характеристик улицы " <<  endl;
   cout <<  "12. Т3 - 5.1.12 Получение характеристик улицы " <<  endl;
   cout <<  "13. Т3 - 5.1.13 Сложение двух улиц " <<  endl;
   cout <<  "14. Т3 - 5.1.14 Добавление дома на улицу " <<  endl;
   cout <<  "15. Т3 - 5.1.15 Удаление дома с улицы " <<  endl;
   cout <<  "16. Т3 - 5.1.16 Установка и снятие признака ремонта улицы " <<  endl;
   cout <<  "17. Т3 - 5.1.17 Автоматическое получение признака ремонта домов улицы
" <<  endl;
   cout <<  "18. Т3 - 5.1.18 Перегрузка оператора присваивания для улиц " <<  endl;

  cout << endl << "0.Выход " <<  endl;

  cin >> iPunkt;

  switch(iPunkt )
  {
////////////////////
case 1:
…..
   case 0:
   case 88-48:
   case 120-48:

cout << endl << "Выход " <<  endl;
   return 0;
   default:
cout << endl << "default " <<  endl;

cout << endl << "Выход " <<  endl;
   return 0;
  } };

  return 0;
}
```

2.2.3. Результаты работы фрагмента программы

```
...
```

```
1.  ТЗ - 5.1.1  Создание улиц с домами
2.  ТЗ - 5.1.2  Создание объектов для  домов улицы
3.  ТЗ - 5.1.3  Создание объектов для  домов улицы на основе других
4.  ТЗ - 5.1.4  Учет свойств дома (см. ТЗ)
5.  ТЗ - 5.1.5  Задание и получение характеристик дома
6.  ТЗ - 5.1.6  Сложение двух домов
7.  ТЗ - 5.1.7  Перегрузить оператор присваивания для домов
8.  ТЗ - 5.1.8  Распечатка характеристик дома
9.  ТЗ - 5.1.9  Учет свойств улицы (см. ТЗ)
10. ТЗ - 5.1.10 Распечатка содержания улицы и ее свойств
11. ТЗ - 5.1.11 Задание характеристик улицы
12. ТЗ - 5.1.12 Получение характеристик улицы
13. ТЗ - 5.1.13 Сложение двух улиц
14. ТЗ - 5.1.14 Добавление дома на улицу
15. ТЗ - 5.1.15 Удаление дома с улицы
16. ТЗ - 5.1.16 Установка и снятие признака ремонта улицы
17. ТЗ - 5.1.17 Автоматическое получение признака ремонта домов улицы
18. ТЗ - 5.1.18 Перегрузка оператора присваивания для улиц

0.Выход
0
```

Enter

…

2.3.Фрагмент текста программы для проверки п.п.5.1.1 ТЗ

2.3.1. Пояснения текста фрагмента

В примере описана улица **S1** и три дома **H1,H2** и **H3**. Распечатывается пустая улица без домов. Затем добавляются эти 3 дома и снова выполняется распечатка содержания улицы с тремя домами.

2.3.2. Текст фрагмента программы

```
  case 1:
cout << endl << "ТЗ - 5.1.1  Создание улиц с домами " <<  endl;
   {
  Street S1("Лениский проспект");
  S1.printOn(cout);
  Home H1("Жилой","д.2", 7,2,3, fast , 5);
  Home H2("Магазин", "д.3", 3);
  Home H3("ДЭЗ", "д.4а", 4,2);
  S1.add(&H1);
  S1.add(&H2);
  S1.add(&H3);
  S1.printOn(cout);
  }
  getchar();
  getchar();
```

2.3.3. Результаты работы фрагмента программы

```
1
Enter
ТЗ - 5.1.1  Создание улиц с домами
{{{*******
Улица -  Ленинский проспект  Ключ для поиска -  Ленинский проспект
Номер улицы -  0  Номер для поиска -  0
Число домов на улице -  0  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Список List пуст
}
*******}}}
```

```
{{{*******
Улица -  Ленинский проспект  Ключ для поиска -  Ленинский проспект
Номер улицы -  0  Номер для поиска -  0
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
```

Enter

2.4.Фрагмент текста программы для проверки п.п.5.1.2 ТЗ

2.4.1. Пояснения текста фрагмента

Описывается семь объектов домов для вызова разных конструкторов. Далее они попарно распечатываются, для чего нужно нажимать клавишу **Enter**.

2.4.2. Текст фрагмента программы

```
  case 2:
cout << "5.1.2  Создание объектов для  домов улицы  " <<  endl;
   {
  Home H1;
  Home H2("Жилой","д.2");
  Home H3("Жилой", "д.3", 3);
  Home H4("Жилой", "д.4а", 4,2);
  Home H5("ДЭЗ", "д.5", 5,2,3);
  Home H6("Жилой", "д.6", 6,2,3, fast);
  Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
  H1.printOn(cout);
  H2.printOn(cout);
  getchar();
  H3.printOn(cout);
  H4.printOn(cout);
  getchar();
  H5.printOn(cout);
  H6.printOn(cout);
  getchar();
  H7.printOn(cout);
  }
  getchar();
```

2.4.3. Результаты работы фрагмента программы

```
2
```

Enter

```
5.1.2  Создание объектов для  домов улицы
Номер сп. -0  Имя не задано
  Символьное имя не задано
  Номер -0
Этажей -0  Жителей -0
Тип дома - простой   Ремонт не нужен!   Число квартир - 0

Номер сп. -0  Имя - Жилой
Номер сим. -д.2
  Номер -0
Этажей -0  Жителей -0
Тип дома - простой   Ремонт не нужен!   Число квартир - 0
```

Enter

```
Номер сп. -0  Имя - Жилой
Номер сим. -д.3
  Номер -3
Этажей -0  Жителей -0
Тип дома - простой   Ремонт не нужен!   Число квартир - 0

Номер сп. -0  Имя - Жилой
Номер сим. -д.4а
  Номер -4
Этажей -2  Жителей -0
Тип дома - простой   Ремонт не нужен!   Число квартир - 0
```

Enter

```
Номер сп. -0  Имя - ДЭЗ
Номер сим. -д.5
  Номер -5
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 0

Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 0
```

Enter

```
Номер сп. -0  Имя - Магазин
Номер сим. -д.7
  Номер -7
Этажей -2  Жителей -3
Тип дома - много строений   Ремонт не нужен!   Число квартир – 5
```

Enter

2.5.Фрагмент текста программы для проверки п.п.5.1.3 ТЗ

2.5.1. Пояснения текста фрагмента

Описываются 2 дома **H6** и **H7** с полным набором параметров. Далее дом **Test** создается на основе дома **H6**. Характеристики двух домов (**H6** и **Test**) распечатываются. Долее описывается указатель на дом (**pHome**). Он инициализируется новым объектом, созданным на основе дома **H7**. Характеристики двух домов (**H7** и динамического через **pHome**) распечатываются. Динамический созданный дом удаляется через указатель.

2.5.2. Текст фрагмента программы

```
    case 3:
cout << "5.1.3  Создание объектов для  домов улицы на основе других  " <<  endl;
    {
   Home H6("Жилой", "д.6", 6,2,3, fast, 100);
   Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
   Home Test(H6);
   H6.printOn(cout);
   Test.printOn(cout);
   getchar();
   getchar();

cout << "Указатель!!! " <<  endl;

   Home *pHome = new Home (H7);
   H7.printOn(cout);
   pHome->printOn(cout);
   delete pHome;
   }
   getchar();
```

```
        break;
```

2.5.3. Результаты работы фрагмента программы

```
3
5.1.3  Создание объектов для  домов улицы на основе других
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 100

Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 100
```

Enter

```
Указатель!!!
Номер сп. -0  Имя - Магазин
Номер сим. -д.7
  Номер -7
Этажей -2  Жителей -3
Тип дома - много строений   Ремонт не нужен!   Число квартир - 5

Номер сп. -0  Имя - Магазин
Номер сим. -д.7
  Номер -7
Этажей -2  Жителей -3
Тип дома - много строений   Ремонт не нужен!   Число квартир -  5
```

Enter

2.6.Фрагмент текста программы для проверки п.п.5.1.4 ТЗ

2.6.1. Пояснения текста фрагмента

Создаются два дома (**H6** и **H7**) с полным набором параметров и распечатываются.

2.6.2. Текст фрагмента программы

```
        case 4:
      cout << "5.1.4  Учет свойств дома(см. ТЗ) " <<  endl;
        {
       Home H6("Жилой", "д.6", 6,2,3, fast, 100);
       H6.printOn(cout);
       Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
       H7.printOn(cout);
       }
       getchar();
       getchar();

       break;
```

2.6.3. Результаты работы фрагмента программы

```
4
5.1.4  Учет свойств дома(см. ТЗ)
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 100
```

```
Номер сп. -0  Имя - Магазин
Номер сим. -д.7
  Номер -7
Этажей -2  Жителей -3
Тип дома - много строений   Ремонт не нужен!   Число квартир - 5
```

Enter

2.7.Фрагмент текста программы для проверки п.п.5.1.5 ТЗ

2.7.1. Пояснения текста фрагмента

Создается дом с полным набором характеристик (**H6**). С помощью метода **getParam** класса **Home** получает основные характеристики дома **iH** (номер) , **Etag** (число этажей), **Men** (число жителей), **Type** (тип дома) , **Apart** (число квартир). Потом эти параметры отдельно распечатываются. Тип дома определяет перечисление **HomeType** (**fast**, **multiple**, **complex**). Далее демонстрируется метод **setParam**, для установки новых параметров дома и выполняется печать.

Во второй части фрагмента определяется дом H7 и с помощью метода **setAllParam** устанавливается признак ремонта (**true**) – 8-й параметр и параметры распечатываются стандартным методом печати – **printOn**.

2.7.2. Текст фрагмента программы

```
    case 5:
  cout << "5.1.5  Задание и получение характеристик дома " <<  endl;
     {
    Home H6("Жилой", "д.6", 6,2,3, fast, 100);
    H6.printOn(cout);
     int  iH,  Etag , Men ,  Apart ;
    HomeType Type;
    // Получение
     H6.getParam(iH , Etag, Men, Type , Apart);
     cout << "Номер -" << iH <<
     "  Этажей -" << Etag <<  "  Жителей -" << Men <<     endl;
            if (Type == fast)
               cout << "Тип дома - простой ";
            if (Type == multiple)
               cout << "Тип дома - много строений ";
            if (Type == complex)
               cout << "Тип дома - сложный ";
           cout << "  Число квартир - " << Apart <<  endl;
    H6.setParam( 11,12,13,fast, 15 );
    H6.printOn(cout);

    getchar();
    getchar();

    cout << "Признак ремонта!!! " <<  endl;
    Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
    H7.printOn(cout);
    H7.setAllParam("Аптека", "10/8", 1,2,3,fast, 5 , true);
    H7.printOn(cout);

    }
     getchar();

    break;
```

2.7.3. Результаты работы фрагмента программы

```
5
```

```
5.1.5  Задание и получение характеристик дома
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 100

Номер -6  Этажей -2  Жителей -3
Тип дома - простой   Число квартир - 100
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -11
Этажей -12  Жителей -13
Тип дома - простой   Ремонт не нужен!   Число квартир - 15
```

Enter

```
Признак ремонта!!!
Номер сп. -0  Имя - Магазин
Номер сим. -д.7
  Номер -7
Этажей -2  Жителей -3
Тип дома - много строений   Ремонт не нужен!   Число квартир - 5

Номер сп. -0  Имя - Аптека
Номер сим. -10/8
  Номер -1
Этажей -2  Жителей -3
Тип дома - простой   Требуется ремонт!   Число квартир – 5
```

Enter

2.8.Фрагмент текста программы для проверки п.п.5.1.6 ТЗ

2.8.1. Пояснения текста фрагмента

Описываются два дома **H1** и **H2** с различными параметрами. Третий дом **Temp** вычисляется с помощью перегруженной операции “+”. Исходные дома и полученный дом распечатываются (**printOn**).

2.8.2. Текст фрагмента программы

```
                case 6:
        cout << "5.1.6  Сложение двух домов " <<  endl;
           {
          Home H1("Жилой", "д.6", 6,2,3, fast , 3);
          Home H2("Ашан", "д.7", 9,10,11, multiple , 5);
          Home Temp;
          Temp = H1 + H2;
          H1.printOn(cout);
          H2.printOn(cout);
          Temp.printOn(cout);
          }
           getchar();
           getchar();
           break;
```

2.8.3. Результаты работы фрагмента программы

```
6
5.1.6  Сложение двух домов
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
```

```
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 3

Номер сп. -0  Имя - Ашан
Номер сим. -д.7
  Номер -9
Этажей -10  Жителей -11
Тип дома - много строений   Ремонт не нужен!   Число квартир - 5

Номер сп. -0  Имя - Жилой + Ашан
Номер сим. -д.6 + д.7
  Номер -6
Этажей -2  Жителей -14
Тип дома - сложный   Ремонт не нужен!   Число квартир – 8
```

Enter

2.9.Фрагмент текста программы для проверки п.п.5.1.7 ТЗ

2.9.1. Пояснения текста фрагмента

Описываются два дома: **H1** (с параметрами) и **Temp** (без параметров). Выполняется присваивание “=” домов. Дома распечатываются стандартным методом **printOn**. Для иллюстрации того, что дома являются самостоятельными объектами изменяется название дома **H1** (на “Перекресток”) и они снова распечатываются.

2.9.2. Текст фрагмента программы

```
            case 7:
cout << "5.1.7  Перегрузить оператор присваивания для домов " <<  endl;
   {
  Home H1("Жилой", "д.6", 6,2,3, fast , 3);
  Home Temp;
  Temp = H1;
  H1.printOn(cout);
  Temp.printOn(cout);
  H1.setName("Перекресток");
  H1.printOn(cout);
  Temp.printOn(cout);

  }
   getchar();
   getchar();
   break;
```

2.9.3. Результаты работы фрагмента программы

```
7
5.1.7  Перегрузить оператор присваивания для домов
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 3

Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 3

Номер сп. -0  Имя - Перекресток
```

```
  Символьное имя не задано
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 3

Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир – 3
```

Enter

2.10.Фрагмент текста программы для проверки п.п.5.1.8 ТЗ

2.10.1. Пояснения текста фрагмента

Описывается дом **H1** с характеристиками и его характеристики распечатываются с помощью стандартного метода **printOn**.

2.10.2. Текст фрагмента программы

```
            case 8:
cout << "5.1.8  Распечатка характеристик дома " <<  endl;
   {
  Home H1("Жилой", "д.6", 6,2,3, fast , 3);
  H1.printOn(cout);
  }
   getchar();
   getchar();

            break;
```

2.10.3. Результаты работы фрагмента программы

```
8
5.1.8  Распечатка характеристик дома
Номер сп. -0  Имя - Жилой
Номер сим. -д.6
  Номер -6
Этажей -2  Жителей -3
Тип дома - простой   Ремонт не нужен!   Число квартир - 3
```

Enter

2.11.Фрагмент текста программы для проверки п.п.5.1.9 ТЗ

2.11.1. Пояснения текста фрагмента

Описана улица **SNew** и три дома **H1**, **H2** и **H3**. Улица улица распечатывается стандартным методом **printOn**. Дома добавляются на улицу (метод **add**) и затем улица распечатывается стандартным методом **printOn**. В распечатке улицы меняется число домов и появляется список названий домов.

2.11.2. Текст фрагмента программы

```
            case 9:
cout << "5.1.9  Учет свойств улицы(см. ТЗ) " <<  endl;
   {
  Home H2("Жилой","д.2", 7,2,3, fast , 5);
  Home H3("Магазин", "д.3", 3);
  Home H4("ДЭЗ", "д.4а", 4,2);
  Street SNew("Улица" , 15);
  SNew.printOn(cout);
  SNew.add(&H2);
```

```
SNew.add(&H3);
SNew.add(&H4);
SNew.printOn(cout);
}
getchar();
getchar();

   break;
```

2.11.3. Результаты работы фрагмента программы

```
9
5.1.9  Учет свойств улицы (см. ТЗ)
{{{*******
Улица -  Улица  Ключ для поиска -  Улица
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  0  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Список List пуст
}
*******}}}
{{{*******
Улица -  Улица  Ключ для поиска -  Улица
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
```

Enter

2.12.Фрагмент текста программы для проверки п.п.5.1.10 ТЗ

2.12.1. Пояснения текста фрагмента

Описана улица **SNew** и три дома **H1**, **H2** и **H3**. Улица распечатывается стандартным методом **printOn**. Дома добавляются на улицу (метод **add**) и затем улица распечатывается стандартным методом **printOn**. В распечатке улицы меняется число домов и появляется список названий домов.

2.12.2. Текст фрагмента программы

```
            case 10:
    cout << "5.1.10 Распечатка содержания улицы и ее свойств" <<  endl;
     {
     Home H2("Жилой","д.2", 7,2,3, fast , 5);
     Home H3("Магазин", "д.3", 3);
     Home H4("ДЭЗ", "д.4а", 4,2);
     Street SNew("Улица" , 15);
     SNew.printOn(cout);
     SNew.add(&H2);
     SNew.add(&H3);
     SNew.add(&H4);
     SNew.printOn(cout);

     }
     getchar();
     getchar();
            break;
```

2.12.3. Результаты работы фрагмента программы

```
10
5.1.10 Распечатка содержания улицы и ее свойств
{{{*******
Улица -  Улица  Ключ для поиска -  Улица
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  0  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Список List пуст
}
*******}}}
{{{*******
Улица -  Улица  Ключ для поиска -  Улица
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
```

Enter

2.13.Фрагмент текста программы для проверки п.п.5.1.11 ТЗ

2.13.1. Пояснения текста фрагмента

Описана улица **Sumstreet** и три дома **H2**, **H3** и **H4**. Дома добавляются на улицу (метод **add**) и затем улица распечатывается стандартным методом **printOn**. В распечатке улицы меняется число домов и появляется список названий домов. С помощью различных методов изменяются параметры улицы: **SetNameStreet** (название), **SetKeyNameStreet** (ключ для поиска), **SetNumbStreet** (номер улицы) и **SetKeyNumbStreet** (номер для поиска). Улица распечатывается стандартным методом **printOn**. Далее с помощью методов: **GetNameStreet**(название), **GetNumbStreet**(номер), **GetKeyNameStreet**(ключевое имя) ,**GetKeyNumbStreet**(ключевой номер) ,**GetNumberHome**(число домов) , **GetNumberMens**(число жителей), **GetNumberApart**(число квартир), **GetRemont**(признак ремонта домов), **GetStreetType** (тип улицы). Тип улицы определяется перечислимым типом **StreetType**: **one**, **two** или **more**.

2.13.2. Текст фрагмента программы

```
            case 11:
    cout << "5.1.11 Задание характеристик улицы  " <<  endl;
      {
     Home H2("Жилой","д.2", 7,2,3, fast , 5);
     Home H3("Магазин", "д.3", 3);
     Home H4("ДЭЗ", "д.4а", 4,2);
     Street Sumstreet("Улица с параметрами" , 15);
     Sumstreet.add(&H2);
     Sumstreet.add(&H3);
     Sumstreet.add(&H4);
     Sumstreet.printOn(cout);
      cout <<  "******* Изменения параметров *********" <<  endl;

     Sumstreet.SetNameStreet("Новая");
     Sumstreet.SetKeyNameStreet("Новая ключ");
     Sumstreet.SetNumbStreet( 33 );
     Sumstreet.SetKeyNumbStreet( 77 );
```

```
Sumstreet.printOn(cout);

cout <<  "******* Параметры *********" <<  endl;
cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
cout <<  "Название улицы для поиска->  " << Sumstreet.GetKeyNameStreet() << endl;
cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() << endl;
cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
 if ( Sumstreet.GetRemont() )
   cout <<  "На улице нужен ремонт домов!" << endl;
else
   cout <<  "На улице не нужен ремонт домов!" << endl;
 if ( Sumstreet.GetStreetType() == one  )
  cout <<  "Тип улицы -> одностороннее движение" <<  endl;
   if ( Sumstreet.GetStreetType() == two  )
  cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;


 }
 getchar();
 getchar();

    break;
```

2.13.3. Результаты работы фрагмента программы

```
11
5.1.11 Задание характеристик улицы
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
******* Изменения параметров *********
{{{*******
Улица -  Новая  Ключ для поиска -  Новая ключ
Номер улицы -  33  Номер для поиска -  77
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
******* Параметры *********
Название улицы ->  Новая
Номер улицы ->  33
Название улицы для поиска->  Новая ключ
Номер улицы для поиска->  77
Число домов на улице = 3
Число жителей на улице = 3
Число квартир на улице = 5
На улице не нужен ремонт домов!
Тип улицы -> двухсторонне  движение
```

Enter

2.14.Фрагмент текста программы для проверки п.п.5.1.12 ТЗ

2.14.1. Пояснения текста фрагмента

Описана улица **Sumstreet** и три дома **H2**, **H3** и **H4**. Дома добавляются на улицу (метод **add**) и затем улица распечатывается стандартным методом **printOn**. В распечатке улицы меняется число домов и появляется список названий домов. С помощью различных методов изменяются параметры улицы: **SetNameStreet** (название), **SetKeyNameStreet** (ключ для поиска), **SetNumbStreet** (номер улицы) и **SetKeyNumbStreet** (номер для поиска). Улица распечатывается стандартным методом **printOn**. Далее с помощью методов: **GetNameStreet**(название), **GetNumbStreet**(номер), **GetKeyNameStreet**(ключевое имя) ,**GetKeyNumbStreet**(ключевой номер) ,**GetNumberHome**(число домов) , **GetNumberMens**(число жителей), **GetNumberApart**(число квартир), **GetRemont**(признак ремонта домов), **GetStreetType** (тип улицы). Тип улицы определяется перечислимым типом **StreetType**: **one**, **two** или **more**.

2.14.2. Текст фрагмента программы

```
                case 12:
        cout << "5.1.12 Получение характеристик улицы " <<  endl;
            {
           Home H2("Жилой","д.2", 7,2,3, fast , 5);
           Home H3("Магазин", "д.3", 3);
           Home H4("ДЭЗ", "д.4а", 4,2);
           Street Sumstreet("Улица с параметрами" , 15);
           Sumstreet.add(&H2);
           Sumstreet.add(&H3);
           Sumstreet.add(&H4);
           Sumstreet.printOn(cout);
         cout <<  "******* Параметры *********" <<  endl;
         cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
         cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
         cout <<  "Название улицы для поиска->  " << Sumstreet.GetKeyNameStreet() << endl;
         cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() << endl;
         cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
         cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
         cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
          if ( Sumstreet.GetRemont() )
             cout <<  "На улице нужен ремонт домов!" << endl;
        else
             cout <<  "На улице не нужен ремонт домов!" << endl;
           if ( Sumstreet.GetStreetType() == one  )
            cout <<  "Тип улицы -> одностороннее движение" <<  endl;
              if ( Sumstreet.GetStreetType() == two  )
            cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;

           }
           getchar();
           getchar();

              break;
```

2.14.3. Результаты работы фрагмента программы

```
12
5.1.12 Получение характеристик улицы
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
```

```
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
Номер - 3 Название ДЭЗ
}
*******}}}
******* Параметры *********
Название улицы ->  Улица с параметрами
Номер улицы ->  15
Название улицы для поиска->  Улица с параметрами
Номер улицы для поиска->  15
Число домов на улице = 3
Число жителей на улице = 3
Число квартир на улице = 5
На улице не нужен ремонт домов!
Тип улицы -> двухсторонне  движение
```

Enter

2.15.Фрагмент текста программы для проверки п.п.5.1.13 ТЗ

2.15.1. Пояснения текста фрагмента

Создаются две улицы: **Astreet** с домами **H2**, **H3** и **H4,** добавленными в голову (**head**) методом (**add**), и **Dstreet** с домами **H5**, **H6** и **H7,** добавленными в хвост (**tail**) методом (**add**). Описывается улица **Sumstreet** и формируется как сумма с помощью перегруженной операции “+”. Исходные улицы и результирующая улица распечатываются стандартным методом **printOn**. Далее распечатываются параметры улицы с помощью специальных методов (см. п.п. 2.14).

2.15.2. Текст фрагмента программы

```
                case 13:
        cout << "5.1.13 Сложение двух улиц" <<  endl;
            {
           Home H2("Жилой","д.2", 7,2,3, fast , 5);
           Home H3("Магазин", "д.3", 3);
           Home H4("ДЭЗ", "д.4а", 4,2);
          Street Astreet("Первая", 20);
           Astreet.add(&H2 , head);
           Astreet.add(&H3 , head);
           Astreet.add(&H4 , head);
          Astreet.printOn(cout);
           Home H5("Аптека","д.2", 7,2,3, fast , 5);
           Home H6("Перекресток", "д.3", 3);
           Home H7("Детский сад", "д.4а", 4,2);
          Street Dstreet("Вторая ", 20);
           Dstreet.add(&H5 , tail);
           Dstreet.add(&H6 , tail);
           Dstreet.add(&H7 , tail);
           Dstreet.printOn(cout);
              getchar();
              getchar();

        cout << endl << "******* Сложение *********" <<  endl;
           Street Sumstreet("  ", 20);
           Sumstreet.printOn(cout);
           Sumstreet = Astreet + Dstreet;
           Sumstreet.printOn(cout);
           getchar();
        cout <<  "******* Параметры *********" <<  endl;
        cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
        cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
        cout <<  "Название улицы для поиска->  " << Sumstreet.GetKeyNameStreet() << endl;
        cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() << endl;
```

```
cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
 if ( Sumstreet.GetRemont() )
    cout <<  "На улице нужен ремонт домов!" << endl;
else
    cout <<  "На улице не нужен ремонт домов!" << endl;
 if ( Sumstreet.GetStreetType() == one  )
  cout <<  "Тип улицы -> одностороннее движение" <<  endl;
    if ( Sumstreet.GetStreetType() == two  )
  cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;
  }
  getchar();
  getchar();

     break;
```

2.15.3. Результаты работы фрагмента программы

```
13
5.1.13 Сложение двух улиц
{{{*******
Улица -  Первая  Ключ для поиска -  Первая
Номер улицы -  20  Номер для поиска -  20
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название ДЭЗ
Номер - 2 Название Магазин
Номер - 3 Название Жилой
}
*******}}}
{{{*******
Улица -  Вторая   Ключ для поиска -  Вторая
Номер улицы -  20  Номер для поиска -  20
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Аптека
Номер - 2 Название Перекресток
Номер - 3 Название Детский сад
}
*******}}}
```

Enter

```
******* Сложение *********
{{{*******
Улица -      Ключ для поиска -
Номер улицы -  20  Номер для поиска -  20
Число домов на улице -  0  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Список List пуст
}
*******}}}
{{{*******
Улица -  Первая + Вторая   Ключ для поиска -  Первая + Вторая
Номер улицы -  20  Номер для поиска -  0
Число домов на улице -  6  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название ДЭЗ
Номер - 2 Название Магазин
```

```
Номер - 3 Название Жилой
Номер - 4 Название Аптека
Номер - 5 Название Перекресток
Номер - 6 Название Детский сад
}
*******}}}
```

Enter

```
******* Параметры *********
Название улицы ->  Первая + Вторая
Номер улицы ->  20
Название улицы для поиска->  Первая + Вторая
Номер улицы для поиска->  0
Число домов на улице = 6
Число жителей на улице = 6
Число квартир на улице = 10
На улице не нужен ремонт домов!
Тип улицы -> двухсторонне  движение
```

Enter

2.16.Фрагмент текста программы для проверки п.п.5.1.14 ТЗ

2.16.1. Пояснения текста фрагмента

В примере описана улица **S1** и два дома **H2** и **H3**. Затем эти два дома добавляются на улицу методом **add** (со вторым параметром по умолчанию – в этом случае используется **tail**) и выполняется распечатка содержания улицы с тремя домами. Далее деменстрируются разные режимы добавления дома: **head** (в начало – H5 и H6), **tail** (в конец – H7) и **Number** (по номеру – H8). Послу каждой операции содержание улицы распечатывается.

2.16.2. Текст фрагмента программы

```
                case 14:
        cout << "5.1.14 Добавление дома на улицу" <<  endl;
           {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Street S1("Улица с параметрами" , 15);
          S1.add(&H2);
          S1.add(&H3);
          S1.printOn(cout);
          getchar();
           getchar();
          Home H5("Аптека","д.2", 7,2,3, fast , 5);
          Home H6("Перекресток", "д.3", 3);
          S1.add(&H5 , head);
          S1.add(&H6 , head);
          S1.printOn(cout);
           getchar();
          Home H7("Детский сад", "д.4а", 4,2);
          S1.add(&H7 , tail);
          S1.printOn(cout);
           getchar();
          Home H8("Жилой 3","д.2", 70,20,30, fast , 50);
          S1.add(&H8 , Number, 2);
           S1.printOn(cout);
          }
          getchar();
          getchar();
```

```
            break;
```

2.16.3. Результаты работы фрагмента программы

```
14
5.1.14 Добавление дома на улицу
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
```

Enter

```
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  4  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Перекресток
Номер - 2 Название Аптека
Номер - 3 Название Жилой
Номер - 4 Название Магазин
}
*******}}}
```

Enter

```
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  5  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Перекресток
Номер - 2 Название Аптека
Номер - 3 Название Жилой
Номер - 4 Название Магазин
Номер - 5 Название Детский сад
}
*******}}}
```

Enter

```
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  6  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Перекресток
Номер - 2 Название Жилой 3
Номер - 3 Название Аптека
Номер - 4 Название Жилой
Номер - 5 Название Магазин
Номер - 6 Название Детский сад
}
*******}}}
```

Enter

2.17.Фрагмент текста программы для проверки п.п.5.1.15 ТЗ

2.17.1. Пояснения текста фрагмента

Создается улица **S1**. В нее разными способами добавляются дома: **H2, H3, H5, H6, H7** и **H8.** Содержимое улицы распечатывается (**printOn**). Далее с помощью метода **del** дома удаляются с улицы в разных режимах: **head** (из начала), **tail** (с конца) и **Number** (по номеру). После каждого удаления содержимое улицы распечатывается.

2.17.2. Текст фрагмента программы

```
            case 15:
cout << "5.1.15 Удаление дома с улицы" <<  endl;
    {
   Home H2("Жилой","д.2", 7,2,3, fast , 5);
   Home H3("Магазин", "д.3", 3);
   Street S1("Улица с параметрами" , 15);
   S1.add(&H2);
   S1.add(&H3);
   Home H5("Аптека","д.2", 7,2,3, fast , 5);
   Home H6("Перекресток", "д.3", 3);
   S1.add(&H5 , head);
   S1.add(&H6 , head);
   Home H7("Детский сад", "д.4а", 4,2);
   S1.add(&H7 , tail);
   Home H8("Жилой 3","д.2", 70,20,30, fast , 50);
   S1.add(&H8 , Number, 2);
    S1.printOn(cout);
    getchar();
    getchar();
    cout << "Удаление дома с улицы конец !!!" <<  endl;
    Home Temp;
   S1.del(&H8, tail); //
   S1.printOn(cout);
   getchar();
    cout << "Удаление дома с улицы начало!!!" <<  endl;
   S1.del(&H8 , head);
   S1.printOn(cout);
    getchar();
   cout << "Удаление дома с улицы второго!!!" <<  endl;

   S1.del(&H8 , Number, 2);
   S1.printOn(cout);
   }
   getchar();

      break;
```

2.17.3. Результаты работы фрагмента программы

```
15
5.1.15 Удаление дома с улицы
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  6  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Перекресток
```

```
Номер - 2 Название Жилой 3
Номер - 3 Название Аптека
Номер - 4 Название Жилой
Номер - 5 Название Магазин
Номер - 6 Название Детский сад
}
*******}}}
```

Enter

```
Удаление дома с улицы конец !!!
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  5  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Перекресток
Номер - 2 Название Жилой 3
Номер - 3 Название Аптека
Номер - 4 Название Жилой
Номер - 5 Название Магазин
}
*******}}}
```

Enter

```
Удаление дома с улицы начало!!!
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  4  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой 3
Номер - 2 Название Аптека
Номер - 3 Название Жилой
Номер - 4 Название Магазин
}
*******}}}
```

Enter

```
Удаление дома с улицы второго!!!
{{{*******
Улица -  Улица с параметрами  Ключ для поиска -  Улица с параметрами
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  3  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой 3
Номер - 2 Название Жилой
Номер - 3 Название Магазин
}
*******}}}
```

Enter

2.18.Фрагмент текста программы для проверки п.п.5.1.16 ТЗ

2.18.1. Пояснения текста фрагмента

В данном фрагменте устанавливается (**true**) и сбрасывается (**false**) признак ремонта улицы. Это **выполняется** с помощью метода класса **SetRemontStr**. Проверка установленного признака выполняется методом класса **GetRemontStr**. Операции выполняются над улицей **S1**, к которой предварительно добавлены дома **H2** и **H3**.

2.18.2. Текст фрагмента программы

```
                case 16:
        cout << "5.1.16 Установка и снятие признака ремонта улицы " <<  endl;
           {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Street S1("Улица 1" , 15);
          S1.add(&H2);
          S1.add(&H3);
          S1.printOn(cout);
           if ( S1.GetRemontStr() )
           cout <<  "Улице нужен ремонт!" << endl;
            else
           cout <<  "Улице не нужен ремонт!" << endl;

          cout << "После установки!!!!" <<  endl;

           S1.SetRemontStr(true);
           if ( S1.GetRemontStr() )
           cout <<  "Улице нужен ремонтв!" << endl;
            else
           cout <<  "Улице не нужен ремонт!" << endl;

          cout << "После снятия признака ремонта!!!!" <<  endl;

           S1.SetRemontStr(false);
           if ( S1.GetRemontStr() )
           cout <<  "Улице нужен ремонт!" << endl;
            else
           cout <<  "Улице не нужен ремонт!" << endl;

          }
           getchar();
           getchar();

              break;
```

2.18.3. Результаты работы фрагмента программы

```
16
5.1.16 Установка и снятие признака ремонта улицы
{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
Улице не нужен ремонт!
После установки!!!!
Улице нужен ремонт!
После снятия признака ремонта!!!!
Улице не нужен ремонт!
```

Enter

2.19.Фрагмент текста программы для проверки п.п.5.1.17 ТЗ

2.19.1. Пояснения текста фрагмента

В данном фрагменте проверяется автоматическая установка признака ремонта домов улицы (не надо путать с признаком ремонта самой улицы). Признак устанавливается автоматически, если хотя бы один ее дом требует ремонта. Первоначально на улицу **S1** добавлено два дома, не требующих ремонта (**H2** и **H3**). Затем с помощью метода **GetRemont** получается признак ремонта домов и распечатывается. Затем у одного из домов **H3** устанавливается признак ремонта дома с помощью метода **setAllParam** класса **Home**. Для установки признака дом удаляется с улицы (**del**) и затем снова добавляется (**add**). Затем распечатываются характеристики измененного дома. После этого снова с помощью метода **GetRemont** получается признак ремонта домов улицы и распечатывается.

2.19.2. Текст фрагмента программы

```
                case 17:
        cout << "5.1.17 Автоматическое получение признака ремонта домов улицы " <<  endl;
           {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Street S1("Улица 1" , 15);
          S1.add(&H2);
          S1.add(&H3);
          cout  << "До установки признака ремонта дома и вычисления признака ремонта
домов улицы!!!!" <<  endl;
          S1.printOn(cout);
           if ( S1.GetRemont() )
           cout <<  "На улице нужен ремонт домов!" << endl;
            else
           cout <<  "На улице не нужен ремонт домов!" << endl;
           S1.del(&H3 , tail);
          H3.setAllParam("Магазин", "д.3", 1,2,3,fast, 3 , true);
          H3.printOn(cout);
          S1.add(&H3);
          S1.GetRemont();
          cout << "После вычисления признака ремонта домов улицы!!!!" <<  endl;
          S1.printOn(cout);
           if ( S1.GetRemont() )
           cout <<  "На улице нужен ремонт домов!" << endl;
            else
           cout <<  "На улице не нужен ремонт домов!" << endl;
          }
          getchar();
          getchar();

             break;
```

2.19.3. Результаты работы фрагмента программы

```
17
5.1.17 Автоматическое получение признака ремонта домов улицы
До установки признака ремонта дома и вычисления признака ремонта домов
улицы!!!!

{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
На улице не нужен ремонт домов!
```

```
Номер сп. -0  Имя - Магазин
Номер сим. -д.3
  Номер -1
Этажей -2  Жителей -3
Тип дома - простой   Требуется ремонт!   Число квартир - 3

После вычисления признака ремонта домов улицы!!!!
{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Нужен ремонт домов улицы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
На улице нужен ремонт домов!
```

Enter

2.20.Фрагмент текста программы для проверки п.п.5.1.18 ТЗ

2.20.1. Пояснения текста фрагмента

Создаются две улицы **S1**(со всеми параметрами и двумя домами **H2** и **H3**) и **SNew**(с минимумом параметров). Далее выполняется операция присваивания домов –“=”. Исходные улицы и улица **SNew** после присваивания распечатывается. Далее улица **S1** изменяется – из нее удаляется один дом H3. После этого обе улицы снова распечатываются (**printOn**). После этого **S1** изменяется, а **SNew** остается неизменной.

2.20.2. Текст фрагмента программы

```
            case 18:
cout << "5.1.18 Перегрузка оператора присваивания для улиц " <<  endl;
    {
   Home H2("Жилой","д.2", 7,2,3, fast , 5);
   Home H3("Магазин", "д.3", 3);
   H3.setAllParam("Магазин", "д.3", 1,2,3,fast, 3 , true);
   Street S1("Улица 1" , 15);
   Street SNew("Улица" , 15);
   S1.add(&H2);
   S1.add(&H3);
   S1.printOn(cout);
   SNew.printOn(cout);
    SNew = S1;
    SNew.printOn(cout);

    S1.del(&H3 , head);

   getchar();
    getchar();
   S1.SetNameStreet("Новое название S1 ");
   cout << "После изменения S1 (название и удален первый)!!!!" <<  endl;
   cout << "S1!!!!" <<  endl;
   S1.printOn(cout);
   cout << "SNew!!!!" <<  endl;
    SNew.printOn(cout);

   }
   getchar();

      break;
```

2.20.3. Результаты работы фрагмента программы

```
18
5.1.18 Перегрузка оператора присваивания для улиц
{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Нужен ремонт домов улицы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
{{{*******
Улица -  Улица  Ключ для поиска -  Улица
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  0  Улица отремонтирована.
Все эти дома отремонтированы.
Список {
Список List пуст
}
*******}}}
{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Нужен ремонт домов улицы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
```

Enter

```
После изменения S1 (название и удален первый)!!!!
S1!!!!
{{{*******
Улица -  Новое название S1   Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  1  Улица отремонтирована.
Нужен ремонт домов улицы.
Список {
Номер - 1 Название Магазин
}
*******}}}
SNew!!!!
{{{*******
Улица -  Улица 1  Ключ для поиска -  Улица 1
Номер улицы -  15  Номер для поиска -  15
Число домов на улице -  2  Улица отремонтирована.
Нужен ремонт домов улицы.
Список {
Номер - 1 Название Жилой
Номер - 2 Название Магазин
}
*******}}}
```

Enter

Приложение 2 Титульный лист

Московский государственный технический университет им. Н.Э.Баумана

УТВЕРЖДАЮ:

Большаков С.А.
"__"_____201X Г.

Комплексная лабораторная работа/ДЗ по дисциплине ПКШ

"Система классов улиц и домов"

Листинги программ
(вид документа)

писчая бумага
(вид носителя)

37
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-XX _____
Большаков С.А.
"__"_____201X г.

Москва - 201X

Приложение 3 Образцы программ

Пример программ для ДЗ/КЛР (массив и список)

// Вариант для Массивов

Модуль - DZ.H (для массивов и списков)

// DZ.H

```
// Общие константы и описания
#define StreetClass             1
#define HomeClass       2
static BOOL DestructorDeleteObj = false;
enum HomeType{fast, multiple , complex};
enum StreetType{one, two , more };
enum TypeAddDel {tail, head, Number ,NumbAfter, NumbCurrent, NumbBefore,
 AssbAfter, AssCurrent, AssBefore , createObj , nocreateObj , deleteObj, nodeleteObj };
```

Модуль - DZ_Class.H (для массивов и списков)

При включении в проект нужно раскомментировать нужную строку в описании класса **Street**, соответсвующую своему варианту.

// DZ_Class.h

```
// Описания классов для варианта массив
#include "stdafx.h"
#include <iostream>
#include "DZ.h"  // Общие константы
using namespace std;
// Абстрактный класс для дома
class AbstrHome : public CObject {
public:
          virtual int classType() = 0;
          virtual char *className() = 0;
          virtual void printOn(ostream &) = 0;
          AbstrHome(){ name = (char *)NULL;
               no = NULL;};
          ~AbstrHome(){
          if( name != (char *)NULL)
         delete [] name;
          };
          char *name;     // ? Резерв Поисковое имя
          int no;              // ? РезервНомер в списке
};
// Абстрактный класс для улицы
class AbstrStreet {
public:
          virtual int classType() = 0;
          virtual char *className() = 0;
          virtual void printOn(ostream &) = 0;
          AbstrStreet(){ name = (char *)NULL;
               no = NULL;};
          ~AbstrStreet(){
          if( name != (char *)NULL)
         delete [] name;
          };
          char *name;     // ? Резерв Поисковое имя
```

```
        int no;             // ? РезервНомер в списке
};
//////////////////////
//// Класс домов
//////////////////////
class Home: public AbstrHome {
// Конструкторы
public:
        Home();
        Home( Home & H) ;
        Home( Home * pH);
        Home(const char *HomName, const char *Number);
        Home(const char *HomName, const char *Number, int  Numb) ;
        Home(const char *HomName, const char *Number, int  Numb,
                int Etag, int Men=0,HomeType Type = fast,int Apart=0);
// Деструктор
        ~Home();
 //Оператор присваивания для поддержки перегрузки "+"
        Home  operator =(Home &  H );
// Виртуальные методы
public:
        virtual int classType() {  return HomeClass; }
        virtual char *className() { return "Home"; }
        virtual void printOn(ostream & out);

// Методы класса Home
        void setName(const char *HomName , const char *Number=NULL);
        const char *getName() {          return (const char *)name; };
        const char *getNumb() {          return (const char *)Home_Number; };
        int  getNo() {    return no; };
        void getParam(int & iH, int & Etag ,int & Men ,HomeType & Type, int & Apart )
           {iH = iHome; Etag = EtagCount; Men = MenCount;
            Type = TypeHome; Apart = NumbApartament ; };
        void setParam(int  iH, int  Etag ,int  Men ,HomeType  Type, int  Apart )
           {  iHome= iH; EtagCount= Etag; MenCount= Men;
           TypeHome = Type; NumbApartament = Apart; };
        void setAllParam(const char *HomName , const char *Number, int  iH, int  Etag ,
       int  Men ,HomeType  Type, int  Apart , BOOL rem = false);
// Дружественная функция для перегрузки
   friend Home & operator +(Home & H1 , Home & H2);
// Свойства класса Home
public:
        char *Home_Number;   // Символьный номер дома
        int iHome;           // Номер дома числовой
        int EtagCount;       // Число этажей
        int MenCount ;       // Число жителей в доме
        HomeType TypeHome ;      // Тип дома
        BOOL HomeRemont ; // Требуется ли ремонт дома
        int NumbApartament;  // Число квартир// …
};
//////////////////////
// Класс улиц
//////////////////////

// Разница для массивов и списков только здесь !!! (нужное для массивов и списков
перекомментировать)
 class Street: public AbstrStreet , public CObArray {
//class Street: public AbstrStreet , public CArray <CObject *, CObject * >{
///////// Для списков кроме перекомментирования нужно изменить библиотеку DZ_LIB.cpp
// class Street:  public AbstrStreet , public CObList {
// class Street:  public AbstrStreet , public CList<CObject *, CObject * > {
public:
```

```
        Street();
            Street(const char *sName);
            Street(const char *sNumbSearch, const char *sName);
            Street(int Num);
            Street(const char *sName , int Num);
            Street(Street & S);
            ~Street() {    };
    // Перегрузка присваивания
    Street operator =(Street & S);
    // Виртуальные методы
    public:
            virtual int classType(){  return StreetClass;};
            virtual char *className() {return "Street";};
            virtual void printOn(ostream & out);
    // Свойства класса Street
    // Номер улицы – тип целый
    public:
            char *Name_Street;
            int Homes_num;
            int NumberStreet;
            BOOL Remont;      // для всех домов
            BOOL RemontStreet;
            StreetType StrType;
            Street * ListOfNear;   // соседние улицы   - резервировано
    // Методы класса Улиц - Street
            void add(Home *pH, TypeAddDel = tail , int Numb = 1 , TypeAddDel = createObj);
            void del( TypeAddDel = tail , int Numb = 1 , TypeAddDel = nodeleteObj);
     // Получить и установить параметры улицы name,, no, NumberStreet
            int GetNumberHome(){return  (int) GetCount()  ;};
            int GetNumberMens(); //
            char * GetNameStreet(){ return Name_Street;};
            char * GetKeyNameStreet(){ return name;};
            int GetNumbStreet(){ return NumberStreet;};
            int GetKeyNumbStreet(){ return no;};
            void SetNameStreet(const char * NameStr);
            void SetKeyNameStreet(const char * sName);
            void SetNumbStreet( int n ){ NumberStreet = n;  return;};
            void SetKeyNumbStreet( int k){ no = k; return;};
            BOOL GetRemont();
            BOOL GetRemontStr(){ return RemontStreet;};
            void SetRemontStr(BOOL rS){ RemontStreet = rS; return;};
            StreetType GetStreetType(){ return StrType;};
            void SetStreetType(StreetType t){ StrType= t; return;};
      // Дружественные функции
      friend Street & operator +( Street &  X ,  Street &  Y );
    };
```

Модуль - DZ_LIB.H (для массивов)

// DZ_LIB.cpp (для DZ_Array.cpp)

```
// Библиотека функций и методов для варианта с массивом (DZ_Array)
#include "stdafx.h"
#include <iostream>
using namespace std;
////////////////////////////////
#include "DZ_Class.h"
////////////////////////////////
class Home;
////////////////////////////
```

```
// Home -  конструкторы
//////////////////////////// Класс Home
// Конструктор без параметров
Home::Home(): AbstrHome() {
                    Home_Number = NULL;
                    iHome       = NULL;
                    no          = NULL;
                    EtagCount   = NULL;
                    MenCount  = NULL;
                    TypeHome  = fast;
                    NumbApartament  = NULL;
                    HomeRemont  = false;
          };
//////////////////////////// Класс Home
// Класс Home: Деструктор
////////////////////////////
Home::~Home( )
          {
           if ( Home_Number != (char *)NULL)  delete []Home_Number;
          };
//////////////////////////// Класс Home
// Конструтрор копирования
          Home::Home( Home & H): AbstrHome()  {
                    no = 0;
                    if ( H.name != (char *)NULL )
                   { name= new char[strlen(H.getName()) + 1];
                     strcpy_s(name , strlen(H.getName()) + 1 , H.getName());}
                    else
                     name = (char *)NULL;

        if ( H.Home_Number != (char *)NULL )
                     { Home_Number = new char[strlen(H.Home_Number) + 1];
                      strcpy_s(Home_Number , strlen(H.Home_Number) + 1, H.Home_Number); }
                    else
                    Home_Number = (char *)NULL;
        iHome = H.iHome;
                    EtagCount   = H.EtagCount;
                    MenCount  = H.MenCount;
                    TypeHome  = H.TypeHome;
                    NumbApartament  = H.NumbApartament;
                    HomeRemont  = H.HomeRemont;
          };
//////////////////////////// Класс Home
// Конструктор: номер дома и имя для поиска (name)
Home::Home(const char *HomName, const char *Number) {
// Имя
          if ( HomName != (char *)NULL )
                     {name= new char[strlen(HomName) + 1];
                     strcpy_s(name, strlen(HomName) + 1 , HomName);
                }
                    else
                     name = (char *)NULL;
// Номер дома
                    if ( Number != (char *)NULL )
                     {Home_Number = new char[strlen(Number) + 1];
                     strcpy_s(Home_Number , strlen(Number) +1 , Number); }
                    else
                      Home_Number = (char *)NULL;
                    iHome       = NULL;
                    EtagCount   = NULL;
                    MenCount  = NULL;
                    TypeHome  = fast;
```

```
                NumbApartament  = NULL;
                HomeRemont  = false;
        };
////////////////////////// Класс Home
// Конструктор: номер дома и имя/номер для поиска (name)
Home::Home(const char *HomName, const char *Number, int  Numb) {
// Имя
        if ( HomName != (char *)NULL )
          {name= new char[strlen(HomName) + 1];
          strcpy_s(name ,  strlen(HomName) + 1, HomName); }
        else
          name = (char *)NULL;
// Номер
        if ( Number != (char *)NULL )
          {Home_Number = new char[strlen(Number) + 1];
          strcpy_s(Home_Number ,  strlen(Number) + 1 , Number); }
        else
           Home_Number = (char *)NULL;
//
        iHome      = Numb;
        EtagCount   = NULL;
        MenCount  = NULL;
        TypeHome  = fast;
        NumbApartament  = NULL;
        HomeRemont  = false;
    }
////////////////////////// Класс Home
// Конструктор со всеми параметрами
Home::Home(const char *HomName, const char *Number, int  Numb,
        int Etag, int Men,HomeType Type,int Apart) {
//Имя
        if ( HomName != (char *)NULL )
          {name= new char[strlen(HomName) + 1];
          strcpy_s(name , strlen(HomName) + 1, HomName); }
        else
          name = (char *)NULL;
// Номер
        if ( Number != (char *)NULL )
          {Home_Number = new char[strlen(Number) + 1];
          strcpy_s(Home_Number , strlen(Number) +1 , Number); }
        else
           Home_Number = (char *)NULL;
//
        iHome      = Numb;
        EtagCount   = Etag;
        MenCount  = Men;
        TypeHome  = Type;
        NumbApartament  = Apart;
        HomeRemont  = false;
    }
////////////////////////// Класс Home
// Конструктор копирования
Home Home::operator =(Home &  H )  {
// Имя
        if ( H.name != (char *)NULL )
            { name= new char[strlen(H.getName()) + 1];
                  strcpy_s(name , strlen(H.getName()) + 1 , H.getName());}
        else
          name = (char *)NULL;
// Номер
        if ( H.Home_Number != (char *)NULL )
          { Home_Number = new char[strlen(H.Home_Number) + 1];
```

```
                    strcpy_s(Home_Number , strlen(H.Home_Number) + 1, H.Home_Number); }
                  else
                    Home_Number = (char *)NULL;
//
      iHome = H.iHome;
                  EtagCount   = H.EtagCount;
                  MenCount  = H.MenCount;
                  TypeHome  = H.TypeHome;
                  NumbApartament  = H.NumbApartament;
                  HomeRemont  = H.HomeRemont;
                  return *this;
};
//////////////////////
// Методы класса Home
//////////////////////
// Метод печати дома
//////////////////
void Home::printOn(ostream & out) {
//
        out << "Номер сп. -" <<no ;
                  if ( name !=(char *)NULL)
                  {out <<  "  Имя - " <<  name << endl; }
                  else
                  {out <<  "  Имя не задано " <<  endl; };
//
      if ( Home_Number !=(char *)NULL)
                  out <<          "Номер сим. -" <<  Home_Number << endl;
                  else
                  {out <<  "  Символьное имя не задано " <<  endl; };
//
                  out  << "  Номер -" << iHome << endl <<
                  "Этажей -" << EtagCount <<  "  Жителей -" << MenCount <<     endl;
                  if (TypeHome == fast)
                     cout << "Тип дома - простой ";
                  if (TypeHome == multiple)
                     cout << "Тип дома - много строений ";
                  if (TypeHome == complex)
                     cout << "Тип дома - сложный ";
                  if ( HomeRemont ) cout << "  Требуется ремонт! ";
                  else cout << "  Ремонт не нужен! ";
                  cout << "  Число квартир - " << NumbApartament <<  endl<< endl;
        }
//////////////////////////// Класс Home
// Метод - Установить имя
//////////////////
void Home::setName(const char *HomName , const char *Number){
 // Имя из базового класса - резерв для приска
            if ( name != NULL)
                        delete []name;
            if ( HomName != (char *)NULL )
                     {name= new char[strlen(HomName) + 1];
                       strcpy_s(name , strlen(HomName) + 1 , HomName); }
            else
                     name = (char *)NULL;
// Имя дома
            if ( Home_Number != NULL)
                        delete []Home_Number;

        if ( Number != NULL)
           {
                  Home_Number= new char[strlen(Number) + 1];
                  strcpy_s(Home_Number , strlen(Number) + 1 , Number);
```

```
                    }
                    else
                      Home_Number = (char *)NULL;
};
//////////////////////////// Класс Home
// Метод - Установить все параметры
//////////////////
void Home::setAllParam(const char *HomName , const char *Number, int  iH, int  Etag ,
     int  Men ,HomeType  Type, int  Apart , BOOL rem)
        {
 // Имя из базового класса - резерв для приска
            if ( name != NULL)
                    delete []name;
            if ( HomName != (char *)NULL )
                  {name= new char[strlen(HomName) + 1];
                  strcpy_s(name, strlen(HomName) + 1, HomName); }
            else
                  name = (char *)NULL;
 // Имя из базового класса - резерв для приска
            if ( Home_Number != NULL)
                    delete []Home_Number;

        if ( Number != NULL)
           {
                Home_Number= new char[strlen(Number) + 1];
                strcpy_s(Home_Number , strlen(Number) + 1, Number) ;
                }
                else
                Home_Number = (char *)NULL;
//
                 iHome= iH;
                 EtagCount= Etag;
                 MenCount= Men;
                 TypeHome = Type;
                 NumbApartament = Apart;
                 HomeRemont  = rem;
            };
// Методы дружественные - friend
//////////////////////////// Класс Home
// Перегрузка сложения домов
//////////////////
 Home & operator +(Home & H1 , Home & H2)
 {
 Home *pTemp = new Home;  // новый объект для сложения
//
 pTemp->no = H1.no;
//  Имя базового - сумма имен
 pTemp->name = new char[strlen(H1.name) + strlen(H2.name)+ 5 ]; // учтем пробелы!
 pTemp->name[0]='\0';
 if (H1.name != NULL)
   strcpy_s(pTemp->name, strlen(H1.name) + 1 , H1.name);
 if (H2.name != NULL)
   {
   strcat_s(pTemp->name, strlen(H1.name) + strlen(H2.name)+ 5 ," + ");
   strcat_s(pTemp->name, strlen(H1.name) + strlen(H2.name)+ 5 ,H2.name);
   };
//  Имя = сумма имен Home_Number
 pTemp->Home_Number = new char[strlen(H1.Home_Number) + strlen(H2.Home_Number)+ 5
];
 pTemp->Home_Number[0]='\0';
 if (H1.Home_Number != NULL)
  strcpy_s(pTemp->Home_Number, strlen( H1.Home_Number ) + 1  , H1.Home_Number);
```

```
        if (H2.Home_Number != NULL)
          {
          strcat_s(pTemp->Home_Number,strlen(H1.Home_Number) + strlen(H2.Home_Number)+ 5 ,"
        + ");
          strcat_s(pTemp->Home_Number,strlen(H1.Home_Number) + strlen(H2.Home_Number)+ 5
        ,H2.Home_Number);
          };
        // Параметры
            pTemp->   iHome      = H1.iHome;
            pTemp->   EtagCount   = H1.EtagCount;
            pTemp->   MenCount  = H1.MenCount + H2.MenCount;
            pTemp->   TypeHome  = complex;
            pTemp->   NumbApartament  = H1.NumbApartament + H2.NumbApartament;
            pTemp->HomeRemont  = (H1.HomeRemont || H2.HomeRemont ) ? true : false ;
         return *pTemp;
         };
        //////////////////////////
        // Класс улиц Street
        ////////////////////
        //// Конструкторы Street
        ////////////////// Класс Street
        //  Конструктор: - Пустые параметры
        ////////////////
        Street::Street():AbstrStreet() {
        // для базового класса
                        no = NULL;
                        name = (char *)NULL;
                        Name_Street = (char *)NULL;
                        Homes_num = NULL;
                        StrType = two;
                        Remont = false;
                        RemontStreet = false;
                        NumberStreet = NULL;
                        ListOfNear = (Street *)NULL;
                } ;
        ////////////////// Класс Street
        // Конструктор: Имя улицы (Name_Street) и имя для поиска (name) общие
        ////////////////
        Street::Street(const char *sName) :AbstrStreet() {
                        if ( sName != (char *) NULL )
                          { name= new char[strlen(sName) + 1];
                          strcpy_s(name , strlen(sName) + 1 , sName);
                          Name_Street= new char[strlen(sName) + 1];
                          strcpy_s(Name_Street ,  strlen(sName) + 1 , sName); }
                        else
                        {
                          name = (char *)NULL;
                          Name_Street = (char *)NULL;
                        };
                        no = NULL;
                        Homes_num = NULL;
                        StrType = two;
                        Remont = false;
                        RemontStreet = false;
                        NumberStreet = NULL;
                        ListOfNear = (Street *)NULL;
                };
        ////////////////// Класс Street
        // Конструктор: Имя улицы и имя для поиска разные
        ////////////////
        Street::Street(const char *sName , const char *sNumb)
                          :AbstrStreet(){
```

```
        // Имя базового для поиска
                        if ( sNumb != (char *) NULL )
                          { name= new char[strlen(sNumb) + 1];
                          strcpy_s(name , strlen(sNumb) + 1 , sNumb);
                          }
                        else
                          name = (char *)NULL;
        // Имя  улицы
                        if ( sName != (char *) NULL )
                          {
                          Name_Street= new char[strlen(sName) + 1];
                          strcpy_s(Name_Street, strlen(sName) + 1 , sName); }
                        else
                          Name_Street = (char *)NULL;
        //
                        no = NULL;
                        Homes_num = NULL;
                        StrType = two;
                        Remont = false;
                        RemontStreet = false;
                        NumberStreet = NULL;
                        ListOfNear = (Street *)NULL;
                };
        ////////////////// Класс Street
        // Конструктор: Номер для поиска
        //////////////////
        Street::Street(int Num) {
              name = (char *)NULL;
                        Name_Street = (char *)NULL;
                        no = Num; // Номер из базового для поиска
                        Homes_num = NULL;
                        StrType = two;
                        Remont = false;
                        RemontStreet = false;
                        NumberStreet = Num;
                        ListOfNear = (Street *)NULL;
                        };
        ////////////////// Класс Street
        // Конструктор: Имя улицы и номер для происка
        //////////////////
                Street::Street(const char *sName , int Num) :AbstrStreet() {
        // Имя улицы и для поиска
                        if ( sName != (char *) NULL )
                          { name= new char[strlen(sName) + 1];
                          strcpy_s(name , strlen(sName) + 1, sName);
                          Name_Street= new char[strlen(sName) + 1];
                          strcpy_s(Name_Street, strlen(sName) + 1 , sName); }
                        else
                        {
                          name = (char *)NULL;
                          Name_Street = (char *)NULL;
                        };
                        no = Num;  // Номер для поиска
                        Homes_num = NULL;
                        StrType = two;
                        Remont = false;
                        RemontStreet = false;
                        NumberStreet = Num;
                        ListOfNear = (Street *)NULL;
                };
        ////////////////// Класс Street*
        // Конструктор копирования
```

```
//////////////////
        Street::Street(Street & S) :AbstrStreet() {
// Имя для поиска
                if ( S.name != (char *) NULL )
                  { name= new char[strlen(S.name) + 1];
                  strcpy_s(name , strlen(S.name) + 1, S.name);
                  }
                else
                  name = (char *)NULL;
// Имя улицы
                if ( S.Name_Street != (char *) NULL )
                  {
                  Name_Street= new char[strlen(S.Name_Street) + 1];
                  strcpy_s(Name_Street , strlen(S.Name_Street) + 1 ,  S.Name_Street); }
                else
                  Name_Street = (char *)NULL;
//
                no = S.no;
                Homes_num = S.Homes_num;
                StrType = S.StrType;
                Remont = S.Remont;
                RemontStreet = S.RemontStreet;
                NumberStreet = S.NumberStreet;
                ListOfNear = S.ListOfNear;
                // Цикл формирования новой улицы
          int nRazm = 0;
                nRazm = (int ) S.GetCount() ;
   for ( int i = 0 ;  i < nRazm ; i++)
   {
  Add( ((Home * ) S.GetAt(i)));
          };
          };
//////////////////////
// Методы класса Street
//////////////////////
////////////////// Класс Street*
// Метод печати улицы
//////////////////
          void Street::printOn(ostream & out) {
      if ( Name_Street != (char *)NULL )
            { out << "{{{******* "<<endl<<"Улица -  "<< Name_Street ; }
            else
            {out << "Название не задано! "<<endl;};
     if ( name != (char *)NULL )
          { out << "  Ключ для поиска -  "<< name << endl ; }
      else
            {out << "Ключ не задан! "<<endl;};

       out << "Номер улицы -  "<< NumberStreet ;

                out << "  Номер для поиска -  "<< no <<  endl;
                out << "Число домов на улице -  "<< Homes_num ;
                if ( RemontStreet )
                    out << "  Нужен ремонт улицы."<<  endl;
                else
                    out << "  Улица отремонтирована."<<  endl;
                if ( GetRemont() )
                    out << "Нужен ремонт домов улицы."<<  endl;
                else
                    out << "Все эти дома отремонтированы."<<  endl;

                int nRazm = 1;
```

```
                nRazm = (int ) GetCount() ;
        out << "Число в домов на улице  = "<< GetCount()<<"}" << endl;
  for ( int i = 0 ;  i < nRazm ; i++)
  {
 ((Home * ) GetAt(i))->printOn(cout);
  }
                out << "*******}}}" << endl;
        }
////////////////// Класс Street*
/// Перегрузка присваивания для улицы
////////////////////
Street Street::operator =(Street & S) {
// Имя для поиска
                if ( S.name != (char *) NULL )
                  { name= new char[strlen(S.name) + 1];
                  strcpy_s(name , strlen(S.name) + 1, S.name);
                  }
                else
                  name = (char *)NULL;
// Имя улицы
                if ( S.Name_Street != (char *) NULL )
                  {
                  Name_Street= new char[strlen(S.Name_Street) + 1];
                  strcpy_s(Name_Street , strlen(S.Name_Street) + 1 ,  S.Name_Street); }
                else
                  Name_Street = (char *)NULL;
//
                no = S.no;
                Homes_num = S.Homes_num;
                StrType = S.StrType;
                Remont = S.Remont;
                RemontStreet = S.RemontStreet;
                NumberStreet = S.NumberStreet;
                ListOfNear = S.ListOfNear;
        int nRazm = 0;
                nRazm = (int ) S.GetCount() ;
// Цикл копирования домов
  for ( int i = 0 ;  i < nRazm ; i++)
  {
 Add( ((Home * ) S.GetAt(i)));
        };
                return *this;
        };
////////////////// Класс Street*
// Метод вычисления необходимости ремонта домов улицы
////////////////////
BOOL Street::GetRemont(){
  BOOL Flag = false;
/// Цикл проверки ремонта
        int nRazm = 0;
                nRazm = (int ) GetCount() ;
// Цикл копирования домов
  for ( int i = 0 ;  i < nRazm ; i++)
  {
if (((Home * ) GetAt(i))->HomeRemont  ) Flag = true ;
  };
return Flag;
};
////////////////// Класс Street*
// Метод  добавления дома на улицу (выбор варианта добавления)
////////////////////
void Street::add(Home *pH , TypeAddDel t , int Numb , TypeAddDel tcr ) {
```

```
            if ( t == tail)    Add((CObject *) pH); // в Хвост
            if ( t == head) InsertAt(0,(CObject *) pH); // в Голову
            if ( t == Number) InsertAt(Numb,(CObject *) pH); // По номеру
// Подсчитать число домов
            Homes_num =  (int) GetCount() ;
           } ;
////////////////// Класс Street*
// Метод  удаления дома с улицы (выбор варианта удаления)
////////////////////
void Street::del( TypeAddDel t , int Numb , TypeAddDel tdl) {

            if ( t == tail) RemoveAt(GetCount() - 1); // Из головы
            if ( t == head) RemoveAt(0); // Из хвоста
            if ( t == Number) RemoveAt(Numb); // По номеру
// Подсчитать число домов
            Homes_num =  (int) GetCount() ;
           } ;
////////////////// Класс Street*
// дружественная функция для операции сложения улиц
////////////////////
Street & operator +( Street &  X ,  Street &  Y )
{
 Street *pTemp = new Street;

 pTemp->name = new char[strlen(X.name) + strlen(Y.name) + 5 ];
 if (X.name != (char *)NULL )
 {  strcpy_s(pTemp->name, strlen(X.name) + 1, X.name);}
 else
   pTemp->name[0]='\0';
 if (Y.name != (char *)NULL )
 {  strcat_s(pTemp->name,  strlen(X.name) + strlen(Y.name) + 5 , " + ");
    strcat_s(pTemp->name, strlen(X.name) + strlen(Y.name) + 5  , Y.name); }
// Name_Street
 pTemp->Name_Street = new char[strlen(X.Name_Street) + strlen(Y.Name_Street) + 5];
 if (X.Name_Street != (char *)NULL )
 {  strcpy_s(pTemp->Name_Street, strlen(X.Name_Street) + 1 , X.Name_Street);}
 else
   pTemp->Name_Street[0]='\0';
 if (Y.Name_Street != (char *)NULL )
   {
   strcat_s(pTemp->Name_Street,strlen(X.Name_Street) + strlen(Y.Name_Street) + 5 ," + ");
   strcat_s(pTemp->Name_Street,strlen(X.Name_Street) + strlen(Y.Name_Street) + 5
,Y.Name_Street);
   };
 // Параметры Street
   pTemp->Homes_num = NULL;
   pTemp->NumberStreet = X.NumberStreet;
   pTemp->Remont = (X.GetRemont() || Y.GetRemont()) ? true : false;      // для домов
   pTemp->RemontStreet = (X.RemontStreet || Y.RemontStreet) ? true : false; ;
   pTemp->StrType = ( X.StrType == two || Y.StrType == two) ? two : one ;
   pTemp->ListOfNear = (Street * )NULL;      // резерв, нужно сложить списки соседних улиц
 // Массив
          int nRazm = 0;
   nRazm = (int ) X.GetCount() ;
   for ( int i = 0 ;  i < nRazm ; i++)
   {
   pTemp->Add( ((Home * ) X.GetAt(i)));
          };
  nRazm = (int ) Y.GetCount() ;
   for ( int i = 0 ;  i < nRazm ; i++)
   {
```

```
        pTemp->Add( ((Home * ) Y.GetAt(i)));
              };
               pTemp->Homes_num =  (int )(pTemp->GetCount()) ;

 return *pTemp;
} ;
////////////////// Класс Street*
// Метод подсчета числа жителей
////////////////////
int Street::GetNumberMens(){
  int Summ = 0;
 // Цикл подсчета
        int nRazm = 0;
        nRazm = (int ) GetCount() ;
  for ( int i = 0 ;  i < nRazm ; i++)
  {
  Summ = Summ +  ((Home *)GetAt( i))->MenCount ;
        };
        return Summ;
    };
////////////////// Класс Street*
// Метод подсчета числа квартир
////////////////////
int Street::GetNumberApart(){
   int Summ = 0;
        int nRazm = 0;
 // Цикл подсчета
        nRazm = (int ) GetCount() ;
  for ( int i = 0 ;  i < nRazm ; i++)
  {
  Summ = Summ +  ((Home *)GetAt( i))->NumbApartament ;
        };
        return Summ;
};
////////////////// Класс Street
// Метод установки названия улицы
////////////////////
void Street::SetNameStreet(const char * NameStr)
        {
        if (Name_Street != (char *) NULL)
          delete []Name_Street;
         if ( NameStr != (char *) NULL )
          {
          Name_Street= new char[strlen(NameStr) + 1];
          strcpy_s(Name_Street , strlen(NameStr) + 1, NameStr);
          }
         else
           Name_Street = (char *) NULL;

        return;};
////////////////// Класс Street
// Метод установки имени для поиска улицы
////////////////////
void Street::SetKeyNameStreet(const char * sName)
        {
          if (name != (char *) NULL)
            delete []name;
         if ( sName != (char *) NULL )
          {
          name= new char[strlen(sName) + 1];
          strcpy_s(name , strlen(sName) + 1 , sName);
          }
```

```
                else
                  Name_Street = (char *) NULL;
                return;};
        ////////////////////////
```

Модуль - DZ_Array.cpp (для массивов)

// DZ_Array.cpp

```
// Главный модуль проекта для отладки и сдачи системы классов (база списки)

#include "stdafx.h"
#include "DZ_Class.h"
#include <iostream>
using namespace std;
////////////////////////////// Для ПМИ
//int mainMETOD() // Эту строку нужно раскомментировать, а следующую закоментировать
// при подключении отладочного варианта текста программы (и наоборот для методики
испытаний)
int  main(void)
{
int iPunkt;
system (" chcp 1251>nul ");
     while ( true ) {
system (" CLS ");
        cout << endl << "Меню  тестового примера системы классов улиц." << endl << endl;

// Новое меню
        cout << endl << "1.  ТЗ - 5.1.1  Создание улиц с домами " <<  endl;
   cout <<  "2.  ТЗ - 5.1.2  Создание объектов для  домов улицы  " <<  endl;
   cout <<  "3.  ТЗ - 5.1.3  Создание объектов для  домов улицы на основе других " <<  endl;
   cout <<  "4.  ТЗ - 5.1.4  Учет свойств дома(см. ТЗ) " <<  endl;
   cout <<  "5.  ТЗ - 5.1.5  Задание и получение характеристик дома " <<  endl;
   cout <<  "6.  ТЗ - 5.1.6  Сложение двух домов " <<  endl;
   cout <<  "7.  ТЗ - 5.1.7  Перегрузить оператор присваивания для домов " <<  endl;
   cout <<  "8.  ТЗ - 5.1.8  Распечатка характеристик дома " <<  endl;
   cout <<  "9.  ТЗ - 5.1.9  Учет свойств улицы(см. ТЗ) " <<  endl;
   cout <<  "10. ТЗ - 5.1.10 Распечатка содержания улицы и ее свойств " <<  endl;
   cout <<  "11. ТЗ - 5.1.11 Задание характеристик улицы " <<  endl;
   cout <<  "12. ТЗ - 5.1.12 Получение характеристик улицы " <<  endl;
   cout <<  "13. ТЗ - 5.1.13 Сложение двух улиц " <<  endl;
   cout <<  "14. ТЗ - 5.1.14 Добавление дома на улицу " <<  endl;
   cout <<  "15. ТЗ - 5.1.15 Удаление дома с улицы " <<  endl;
   cout <<  "16. ТЗ - 5.1.16 Установка и снятие признака ремонта улицы " <<  endl;
   cout <<  "17. ТЗ - 5.1.17 Автоматическое получение признака ремонта домов улицы " <<  endl;
   cout <<  "18. ТЗ - 5.1.18 Перегрузка оператора присваивания для улиц " <<  endl;

        cout << endl << "0.Выход " <<  endl;

//      system (" pause ");
        cin >> iPunkt;
//      cout << "Ввели - " << iPunkt<< endl;
        switch(iPunkt )
        {
//////////////////
         case 1:
cout << endl << "ТЗ - 5.1.1  Создание улиц с домами " <<  endl;
   {
         Street S1("Ленинский проспект");
         S1.printOn(cout);
```

```
        Home H1("Жилой","д.2", 7,2,3, fast , 5);
        Home H2("Магазин", "д.3", 3);
        Home H3("ДЭЗ", "д.4а", 4,2);
        S1.add(&H1);
        S1.add(&H2);
        S1.add(&H3);
        S1.printOn(cout);

        }
   system (" pause ");
   system (" pause ");

        break;
///
        case 2:
cout << "5.1.2  Создание объектов для  домов улицы  " <<  endl;
    {
        Home H1;
        Home H2("Жилой","д.2");
        Home H3("Жилой", "д.3", 3);
        Home H4("Жилой", "д.4а", 4,2);
        Home H5("ДЭЗ", "д.5", 5,2,3);
        Home H6("Жилой", "д.6", 6,2,3, fast);
        Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
        H1.printOn(cout);
        H2.printOn(cout);
        system (" pause ");
        H3.printOn(cout);
        H4.printOn(cout);
        system (" pause ");
        H5.printOn(cout);
        H6.printOn(cout);
        system (" pause ");
        H7.printOn(cout);
        }
   system (" pause ");

        break;
///
        case 3:
cout << "5.1.3  Создание объектов для  домов улицы на основе других  " <<  endl;
    {
        Home H6("Жилой", "д.6", 6,2,3, fast, 100);
        Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
        Home Test(H6);
        H6.printOn(cout);
        Test.printOn(cout);
        system (" pause ");
        system (" pause ");


cout << "Указатель!!! " <<  endl;

        Home *pHome = new Home (H7);
        H7.printOn(cout);
        pHome->printOn(cout);
        delete pHome;
        }
   system (" pause ");

        break;
//
```

```
        case 4:
cout << "5.1.4  Учет свойств дома(см. ТЗ) " <<  endl;
    {
        Home H6("Жилой", "д.6", 6,2,3, fast, 100);
        H6.printOn(cout);
        Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
        H7.printOn(cout);
        }
   system (" pause ");
   system (" pause ");

        break;
//
        case 5:
cout << "5.1.5  Задание и получение характеристик дома " <<  endl;
    {
        Home H6("Жилой", "д.6", 6,2,3, fast, 100);
        H6.printOn(cout);
    int  iH,  Etag , Men ,  Apart ;
        HomeType Type;
        // Получение
    H6.getParam(iH , Etag, Men, Type , Apart);
    cout << "Номер -" << iH <<
          "  Этажей -" << Etag <<  "  Жителей -" << Men <<     endl;
              if (Type == fast)
                 cout << "Тип дома - простой ";
              if (Type == multiple)
                 cout << "Тип дома - много строений ";
              if (Type == complex)
                 cout << "Тип дома - сложный ";
              cout << "  Число квартир - " << Apart <<  endl;
        H6.setParam( 11,12,13,fast, 15 );
        H6.printOn(cout);

        system (" pause ");
    system (" pause ");
        // Задание

        cout << "Признак ремонта!!! " <<  endl;
        Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
        H7.printOn(cout);
        H7.setAllParam("Аптека", "10/8", 1,2,3,fast, 5 , true);
        H7.printOn(cout);

        }
    system (" pause ");

        break;
//

                case 6:
cout << "5.1.6  Сложение двух домов " <<  endl;
    {
  Home H1("Жилой", "д.6", 6,2,3, fast , 3);
  Home H2("Ашан", "д.7", 9,10,11, multiple , 5);
  Home Temp;
  Temp = H1 + H2;
  H1.printOn(cout);
  H2.printOn(cout);
  Temp.printOn(cout);
        }
   system (" pause ");
```

```
    system (" pause ");
//
          break;
//

                  case 7:
cout << "5.1.7  Перегрузить оператор присваивания для домов " <<  endl;
    {
   Home H1("Жилой", "д.6", 6,2,3, fast , 3);
   Home Temp;
   Temp = H1;
   H1.printOn(cout);
   Temp.printOn(cout);
   H1.setName("Перекресток");
   H1.printOn(cout);
   Temp.printOn(cout);

          }
    system (" pause ");
    system (" pause ");
//
          break;

//

                  case 8:
cout << "5.1.8  Распечатка характеристик дома " <<  endl;
    {
   Home H1("Жилой", "д.6", 6,2,3, fast , 3);
   H1.printOn(cout);
          }
    system (" pause ");
    system (" pause ");
//
                  break;
//

                  case 9:
cout << "5.1.9  Учет свойств улицы(см. ТЗ) " <<  endl;
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Home H4("ДЭЗ", "д.4а", 4,2);
          Street SNew("Улица" , 15);
          SNew.printOn(cout);
          SNew.add(&H2);
          SNew.add(&H3);
          SNew.add(&H4);
          SNew.printOn(cout);
          }
    system (" pause ");
    system (" pause ");
//
             break;
//

                  case 10:
cout << "5.1.10 Распечатка содержания улицы и ее свойств" <<  endl;
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Home H4("ДЭЗ", "д.4а", 4,2);
```

```
        Street SNew("Улица" , 15);
        SNew.printOn(cout);
        SNew.add(&H2);
        SNew.add(&H3);
        SNew.add(&H4);
        SNew.printOn(cout);

        }
   system (" pause ");
   system (" pause ");
//
                break;
//

                case 11:
cout << "5.1.11 Задание характеристик улицы  " <<  endl;
    {
        Home H2("Жилой","д.2", 7,2,3, fast , 5);
        Home H3("Магазин", "д.3", 3);
        Home H4("ДЭЗ", "д.4а", 4,2);
        Street Sumstreet("Улица с параметрами" , 15);
        Sumstreet.add(&H2);
        Sumstreet.add(&H3);
        Sumstreet.add(&H4);
        Sumstreet.printOn(cout);
    cout <<  "******* Изменения параметров *********" <<  endl;

        Sumstreet.SetNameStreet("Новая");
        Sumstreet.SetKeyNameStreet("Новая ключ");
        Sumstreet.SetNumbStreet( 33 );
        Sumstreet.SetKeyNumbStreet( 77 );
        Sumstreet.printOn(cout);

 cout <<  "******* Параметры *********" <<  endl;
 cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
 cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
 cout <<  "Название улицы для поиска->  " << Sumstreet.GetKeyNameStreet() << endl;
 cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() << endl;
 cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
 cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
 cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
  if ( Sumstreet.GetRemont() )
    cout <<  "На улице нужен ремонт домов!" << endl;
 else
    cout <<  "На улице не нужен ремонт домов!" << endl;
  if ( Sumstreet.GetStreetType() == one  )
   cout <<  "Тип улицы -> одностороннее движение" <<  endl;
     if ( Sumstreet.GetStreetType() == two  )
   cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;


        }
   system (" pause ");
   system (" pause ");
//
            break;
//
                case 12:
cout << "5.1.12 Получение характеристик улицы " <<  endl;
    {
                Home H2("Жилой","д.2", 7,2,3, fast , 5);
        Home H3("Магазин", "д.3", 3);
```

```
          Home H4("ДЭЗ", "д.4а", 4,2);
          Street Sumstreet("Улица с параметрами" , 15);
          Sumstreet.add(&H2);
          Sumstreet.add(&H3);
          Sumstreet.add(&H4);
          Sumstreet.printOn(cout);
 cout <<  "******* Параметры *********" <<  endl;
 cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
 cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
 cout <<  "Название улицы для поиска->  " << Sumstreet.GetKeyNameStreet() << endl;
 cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() << endl;
 cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
 cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
 cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
  if ( Sumstreet.GetRemont() )
     cout <<  "На улице нужен ремонт домов!" << endl;
 else
     cout <<  "На улице не нужен ремонт домов!" << endl;
   if ( Sumstreet.GetStreetType() == one  )
    cout <<  "Тип улицы -> одностороннее движение" <<  endl;
      if ( Sumstreet.GetStreetType() == two  )
    cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;

          }
    system (" pause ");
    system (" pause ");
//
              break;
//
                    case 13:
cout << "5.1.13 Сложение двух улиц" <<  endl;
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Home H4("ДЭЗ", "д.4а", 4,2);
         Street Astreet("Первая", 20);
          Astreet.add(&H2 , head);
          Astreet.add(&H3 , head);
          Astreet.add(&H4 , head);
         Astreet.printOn(cout);
          Home H5("Аптека","д.2", 7,2,3, fast , 5);
          Home H6("Перекресток", "д.3", 3);
          Home H7("Детский сад", "д.4а", 4,2);
         Street Dstreet("Вторая ", 20);
          Dstreet.add(&H5 , tail);
          Dstreet.add(&H6 , tail);
          Dstreet.add(&H7 , tail);
          Dstreet.printOn(cout);
      system (" pause ");
      system (" pause ");

 cout << endl << "******* Сложение *********" <<  endl;
           Street Sumstreet("  ", 20);
           Sumstreet.printOn(cout);
           Sumstreet = Astreet + Dstreet;
           Sumstreet.printOn(cout);
           system (" pause ");
 cout <<  "******* Параметры *********" <<  endl;
 cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
 cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
 cout <<  "Название улицы для поиска->  " << Sumstreet.GetKeyNameStreet() << endl;
 cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() << endl;
```

```
 cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
 cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
 cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
  if ( Sumstreet.GetRemont() )
     cout <<  "На улице нужен ремонт домов!" << endl;
 else
     cout <<  "На улице не нужен ремонт домов!" << endl;
   if ( Sumstreet.GetStreetType() == one  )
    cout <<  "Тип улицы -> одностороннее движение" <<  endl;
      if ( Sumstreet.GetStreetType() == two  )
    cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;
          }
    system (" pause ");
    system (" pause ");
//
             break;
//
                    case 14:
cout << "5.1.14 Добавление дома на улицу" <<  endl;
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Street S1("Улица с параметрами" , 15);
          S1.add(&H2);
          S1.add(&H3);
          S1.printOn(cout);
          system (" pause ");
     system (" pause ");
          Home H5("Аптека","д.2", 7,2,3, fast , 5);
          Home H6("Перекресток", "д.3", 3);
          S1.add(&H5 , head);
          S1.add(&H6 , head);
          S1.printOn(cout);
     system (" pause ");
          Home H7("Детский сад", "д.4а", 4,2);
          S1.add(&H7 , tail);
          S1.printOn(cout);
     system (" pause ");
          Home H8("Жилой 3","д.2", 70,20,30, fast , 50);
          S1.add(&H8 , Number, 2);
     S1.printOn(cout);
          }
    system (" pause ");
    system (" pause ");
//
             break;
//

                    case 15:
cout << "5.1.15 Удаление дома с улицы" <<  endl;
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Street S1("Улица с параметрами" , 15);
          S1.add(&H2);
          S1.add(&H3);
          Home H5("Аптека","д.2", 7,2,3, fast , 5);
          Home H6("Перекресток", "д.3", 3);
          S1.add(&H5 , head);
          S1.add(&H6 , head);
          Home H7("Детский сад", "д.4а", 4,2);
          S1.add(&H7 , tail);
```

```
        Home H8("Жилой 3","д.2", 70,20,30, fast , 50);
        S1.add(&H8 , Number, 2);
    S1.printOn(cout);
    system (" pause ");
    system (" pause ");
    cout << "Удаление дома с улицы конец !!!" <<  endl;
    Home Temp;
        S1.del(); //
        S1.printOn(cout);
        system (" pause ");
    cout << "Удаление дома с улицы начало!!!" <<  endl;
        S1.del( head);
        S1.printOn(cout);
    system (" pause ");
        cout << "Удаление дома с улицы второго!!!" <<  endl;

        S1.del( Number, 2);
        S1.printOn(cout);
        }
   system (" pause ");
//
            break;
//
                case 16:
cout << "5.1.16 Установка и снятие признака ремонта улицы " <<  endl;
    {
        Home H2("Жилой","д.2", 7,2,3, fast , 5);
        Home H3("Магазин", "д.3", 3);
        Street S1("Улица 1" , 15);
        S1.add(&H2);
        S1.add(&H3);
        S1.printOn(cout);
          if ( S1.GetRemontStr() )
    cout <<  "Улице нужен ремонт!" << endl;
      else
    cout <<  "Улице не нужен ремонт!" << endl;

        cout << "После установки!!!!" <<  endl;

    S1.SetRemontStr(true);
          if ( S1.GetRemontStr() )
    cout <<  "Улице нужен ремонт!" << endl;
      else
    cout <<  "Улице не нужен ремонт!" << endl;

        cout << "После снятия признака ремонта!!!!" <<  endl;

    S1.SetRemontStr(false);
          if ( S1.GetRemontStr() )
    cout <<  "Улице нужен ремонт!" << endl;
      else
    cout <<  "Улице не нужен ремонт!" << endl;

        }
   system (" pause ");
   system (" pause ");
//
            break;
//

                case 17:
cout << "5.1.17 Автоматическое получение признака ремонта домов улицы " <<  endl;
```

```
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Street S1("Улица 1" , 15);
          S1.add(&H2);
          S1.add(&H3);
          cout << "До установки признака ремонта дома и вычисления признака ремонта домов
улицы!!!!" <<  endl;
          S1.printOn(cout);
            if ( S1.GetRemont() )
    cout <<  "На улице нужен ремонт домов!" << endl;
      else
    cout <<  "На улице не нужен ремонт домов!" << endl;
    S1.del( tail);
          H3.setAllParam("Магазин", "д.3", 1,2,3,fast, 3 , true);
    H3.printOn(cout);
          S1.add(&H3);
          S1.GetRemont();
          cout << "После вычисления признака ремонта домов улицы!!!!" <<  endl;
          S1.printOn(cout);
            if ( S1.GetRemont() )
    cout <<  "На улице нужен ремонт домов!" << endl;
      else
    cout <<  "На улице не нужен ремонт домов!" << endl;
          }
   system (" pause ");
   system (" pause ");
//
             break;
//

                  case 18:
cout << "5.1.18 Перегрузка оператора присваивания для улиц " <<  endl;
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          H3.setAllParam("Магазин", "д.3", 1,2,3,fast, 3 , true);
          Street S1("Улица 1" , 15);
          Street SNew("Улица" , 15);
          S1.add(&H2);
          S1.add(&H3);
          S1.printOn(cout);
          SNew.printOn(cout);
    SNew = S1;
    SNew.printOn(cout);

    S1.del( head);

          system (" pause ");
    system (" pause ");
          S1.SetNameStreet("Новое название S1 ");
          cout << "После изменения S1 (название и удален первый)!!!!" <<  endl;
          cout << "S1!!!!" <<  endl;
          S1.printOn(cout);
          cout << "SNew!!!!" <<  endl;
    SNew.printOn(cout);

          }
   system (" pause ");
//
             break;
//
```

```
//////////////////

////////////////////////////////
          case 0:
          case 88-48:
          case 120-48:

cout << endl << "Выход " <<  endl;
system(" PAUSE");

          return 0;
          default:
cout << endl << "Выбор функции не верен! " <<  endl;
    break;
/// При  выходе
cout << endl << "Выход " <<  endl;
 system(" PAUSE");
          return 0;
         }
 system(" PAUSE");
};

 ///
 system(" PAUSE");
         return 0;
}


//////////////////////////////// Для отладки
int  mainDEBUG(void) // Эту строку нужно рзаскомментировать, а следующую зраскоментировать
// при подключении отладочного варианта текста программы (и наоборот для методики
испытаний)
//int  main(void)
{
system(" chcp 1251 > nul");

int iPunkt;

     while ( true ) {
system (" CLS ");
          cout << endl << "Меню  тестового примера системы классов улиц." << endl << endl;
          cout << endl << "1.Конструкторы Home " <<  endl;
          cout << endl << "2.Методы Home " <<  endl;
          cout << endl << "3.Конструкторы Street " <<  endl;
          cout << endl << "4.Методы Street " <<  endl;
          cout << endl << "0.Выход " <<  endl;

          cin >> iPunkt;

//        cout << "Ввели - " << iPunkt<< endl;
          switch(iPunkt)
          {
           case 1:
cout << endl << "1.Конструкторы Home " <<  endl;
////////////////////////////////
           {
           Home H1;
           Home H2("Жилой","д.2");
           Home H3("Жилой", "д.3", 3);
           Home H4("Жилой", "д.4а", 4,2);
```

```
            Home H5("ДЭЗ", "д.5", 5,2,3);
            Home H6("Жилой", "д.6", 6,2,3, fast);
            Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
            H1.printOn(cout);
            H2.printOn(cout);
            H3.printOn(cout);
            H4.printOn(cout);
            H5.printOn(cout);
            H6.printOn(cout);
            H7.printOn(cout);
// Конструктор копирования
            Home Test(H5);
            H5.printOn(cout);
            Test.printOn(cout);
            Home *pHome = new Home (H6);
            H6.printOn(cout);
            pHome->printOn(cout);
            Home Temp;
            Temp = H7;
    H7.printOn(cout);
            Temp.printOn(cout);
cout << endl << "Указатели!!!!!!!! " <<  endl;

            Home *pHome1 = new Home (* pHome);
     pHome->printOn(cout);
            pHome1->printOn(cout);
            delete pHome;
            delete pHome1;
            }
//////////////////////////////////////
//////////////////////////////////////
            break;
            case 2:
cout << endl << "2.Методы Home " <<  endl;
//////////////////////////////////////
            {
// + , SetName,printOn,classType,className, getName, getNumb,getno, getParam
// +
   Home H1("Жилой", "д.6", 6,2,3, fast , 3);
   Home H2("Жилой", "д.7", 9,10,11, fast , 5);
   Home Temp;
   Temp = H1 + H2;
   H1.printOn(cout);
   H2.printOn(cout);
   Temp.printOn(cout);
// SetName
   Temp.setName("Университет");
   Temp.printOn(cout);
   Temp.setName("Магазин" ,"д.11/12");
   Temp.printOn(cout);
//classType
  cout << "Тип класса = "<< Temp.classType() << endl;
//className
  cout << "Name класса = "<< Temp.className() << endl;
//getName
  cout << "Name дома = "<< Temp.getName() << endl;
//getNumb
  cout << "Номер дома = "<< Temp.getNumb() << endl;
//getno
  cout << "Номер в списке = "<< Temp.getNo() << endl;
//getParam
  cout << "Параметры: "<< endl;
```

```
  int  iH,  Etag , Men ,  Apart ;
  HomeType  Type;
  Temp.getParam(iH , Etag, Men, Type , Apart);
    cout << "Номер -" << iH <<
            "  Этажей -" << Etag <<  "  Жителей -" << Men <<     endl;
                  if (Type == fast)
                     cout << "Тип дома - простой ";
                  if (Type == multiple)
                     cout << "Тип дома - много строений ";
                  if (Type == complex)
                     cout << "Тип дома - сложный ";
                   cout << "  Число квартир - " << Apart <<  endl;
           Temp.printOn(cout);
           Temp.setParam(1,2,3,fast, 5 );
           Temp.printOn(cout);
           Temp.setAllParam("Аптека", "10/8 кв.3", 11,12,13,fast, 15 );
           Temp.printOn(cout);
           }
//////////////////////////////////////
           break;
           case 3:
cout << endl << "3.Конструкторы Street " <<  endl;
           {
           Street S1;
           Street S2("Лениский проспект");
           Street S3("Ленин","Лениский проспект");
           Street S4(5);
           Street S5("Горького ул." , 7);
           Home H5("ДЭЗ", "д.5", 5,2,3);
           Home H6("Жилой", "д.6", 6,2,3, fast);
           Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
           S5.Add( (CObject *)&H5);
           S5.Add( (CObject *)&H6);
           S5.Add( (CObject *)&H7);
           S5.printOn(cout);
           S1.printOn(cout);
           S2.printOn(cout);
           S3.printOn(cout);
           S4.printOn(cout);
           S5.printOn(cout);
           //
    S5.RemoveAt(1);
           S5.printOn(cout);
//////////////////////////////
           Street S6(S5);
           S6.printOn(cout);
           Home H2("Жилой","д.2", 7,2,3, fast , 5);
           Home H3("Магазин", "д.3", 3);
           Home H4("ДЭЗ", "д.4а", 4,2);

           Home HNew("Сарай","д.10/12", 7,2,3, fast , 5);
           Street SNew("Улица" , 15);
           SNew.printOn(cout);
           SNew.Add((CObject *)&HNew);
           SNew.Add((CObject *)&H2);
           SNew.Add((CObject *)&H3);
           SNew.Add((CObject *)&H4);
           SNew.printOn(cout);
           cout << endl << "******* Конструктор копии *********" <<  endl;
           Street SCopy(SNew);
           SCopy.printOn(cout);
           cout << endl << "******* Перегрузка присваивания *********" <<  endl;
```

```
    Street SEmpty("Пусто!!" , 15);
          SEmpty.printOn(cout);
    SEmpty = SCopy;
          SEmpty.printOn(cout);
          // МЕТОД add
Street Shead("Улица Head" , 15);
          Shead.add(&HNew , head);
          Shead.add(&H2 , head);
          Shead.add(&H3 , head);
          Shead.add(&H4 , head);
          Shead.printOn(cout);
          ///
          Shead.del( head);
          Shead.printOn(cout);
          Shead.del( tail);
          Shead.printOn(cout);
          Shead.del( Number , 1);
          Shead.printOn(cout);
          };
          break;


          case 4:
cout << endl << "4.Методы Street " <<  endl;
          {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Home H4("ДЭЗ", "д.4а", 4,2);
          Street Astreet("Добавление head", 20);
          Astreet.printOn(cout);
          cout << endl << "******* Добавление head *********" <<  endl;
          Astreet.add(&H2 , head);
          Astreet.add(&H3 , head);
          Astreet.add(&H4 , head);
          Astreet.printOn(cout);

          Home H5("Жилой","д.2", 7,2,3, fast , 5);
          Home H6("Магазин", "д.3", 3);
          Home H7("ДЭЗ", "д.4а", 4,2);
cout << endl << "******* Добавление tail err *********" <<  endl;
          Street Dstreet("Добавление tail ", 20);
          Dstreet.add(&H5 , tail);
          Dstreet.add(&H6 , tail);
          Dstreet.add(&H7 , tail);
          Dstreet.printOn(cout);
 cout << endl << "******* Добавление NUM 2 *********" <<  endl;
          Home H8("Жилой 3","д.2", 70,20,30, fast , 50);
          Home H9("Магазин 3", "д.3", 3);
          Home H10("ДЭЗ 3", "д.4а", 4,2);
          Dstreet.add(&H8 , Number, 2);
          Dstreet.add(&H9 , Number, 2);
          Dstreet.add(&H10 , Number, 2);
          Dstreet.printOn(cout);

          cout << endl << "******* Удаление *********" <<  endl;
          Dstreet.del(); //
          Dstreet.printOn(cout);
          Dstreet.del( head);
          Dstreet.printOn(cout);
          Dstreet.del( Number, 2);
          Dstreet.printOn(cout);
```

```
cout << endl << "******* Сложение *********" <<  endl;
            Street Sumstreet("  ", 20);
            Sumstreet.printOn(cout);
            Sumstreet = Astreet + Dstreet;
            Sumstreet.printOn(cout);



cout <<  "******* Параметры *********" <<  endl;
 cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
 cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
 cout <<  "Название улицы для поиска->  " << Sumstreet.GetKeyNameStreet() << endl;
 cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() << endl;
 cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
 cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
 cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
  if ( Sumstreet.GetRemont() )
     cout <<  "На улице нужен ремонт домов!" << endl;
 else
     cout <<  "На улице нужен ремонт домов!" << endl;
   if ( Sumstreet.GetStreetType() == one  )
    cout <<  "Тип улицы -> одностороннее движение" <<  endl;
      if ( Sumstreet.GetStreetType() == two  )
    cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;
 cout <<  "******* Изменения параметров *********" <<  endl;

          Sumstreet.SetNameStreet("Новая");
          Sumstreet.SetKeyNameStreet("Новая ключ");
          Sumstreet.SetNumbStreet( 33 );
          Sumstreet.SetKeyNumbStreet( 77 );
 cout <<  "Название улицы 2->  " << Sumstreet.GetNameStreet() << endl;
 cout <<  "Номер улицы 2->  " << Sumstreet.GetNumbStreet() << endl;
 cout <<  "Название улицы для поиска 2->  " << Sumstreet.GetKeyNameStreet() << endl;
 cout <<  "Номер улицы для поиска 2->  " << Sumstreet.GetKeyNumbStreet() << endl;

           };
           break;
//////////////////////////////////
           case 0:
           case 88-48:
           case 120-48:

cout << endl << "Выход " <<  endl;
system(" PAUSE");

           return 0;
//
           default:
cout << endl << "Выбор функции не верен! " <<  endl;
     break;
/// При  выходе
cout << endl << "Выход " <<  endl;
 system(" PAUSE");
           return 0;
          }
 system(" PAUSE");
};

 ///
 system(" PAUSE");
          return 0;
}
```

// Вариант для Списков

Модуль - DZ_LIB.H (для списков)

// DZ_LIB.cpp (для DZ_List.cpp)

```
/ Библиотека функций и методов для варианта с списком (DZ_List)
#include "stdafx.h"
#include <iostream>
using namespace std;
////////////////////////////////
#include "DZ_Class.h"
////////////////////////////////
class Home;
//////////////////////////
// Home -  конструкторы
////////////////////////// Класс Home
// Конструктор без параметров
Home::Home(): AbstrHome() {
                Home_Number = NULL;
                iHome       = NULL;
                no          = NULL;
                EtagCount   = NULL;
                MenCount  = NULL;
                TypeHome  = fast;
                NumbApartament  = NULL;
                HomeRemont  = false;
        };
////////////////////////// Класс Home
// Класс Home: Деструктор
//////////////////////////
Home::~Home( )
         {
          if ( Home_Number != (char *)NULL)  delete []Home_Number;
         };
////////////////////////// Класс Home
// Конструтрор копирования
        Home::Home( Home & H): AbstrHome()  {
                no = 0;
                if ( H.name != (char *)NULL )
               { name= new char[strlen(H.getName()) + 1];
                 strcpy_s(name , strlen(H.getName()) + 1 , H.getName());}
                else
                 name = (char *)NULL;

      if ( H.Home_Number != (char *)NULL )
                 { Home_Number = new char[strlen(H.Home_Number) + 1];
                  strcpy_s(Home_Number , strlen(H.Home_Number) + 1, H.Home_Number); }
                else
                Home_Number = (char *)NULL;
      iHome = H.iHome;
                EtagCount   = H.EtagCount;
                MenCount  = H.MenCount;
                TypeHome  = H.TypeHome;
                NumbApartament  = H.NumbApartament;
                HomeRemont  = H.HomeRemont;
        };
```

```
//////////////////////////// Класс Home
// Конструктор: номер дома и имя для поиска (name)
Home::Home(const char *HomName, const char *Number) {
// Имя
        if ( HomName != (char *)NULL )
                 {name= new char[strlen(HomName) + 1];
                  strcpy_s(name, strlen(HomName) + 1 , HomName);
             }
                else
                  name = (char *)NULL;
// Номер дома
                if ( Number != (char *)NULL )
                  {Home_Number = new char[strlen(Number) + 1];
                  strcpy_s(Home_Number , strlen(Number) +1 , Number); }
                else
                   Home_Number = (char *)NULL;
                iHome      = NULL;
                EtagCount   = NULL;
                MenCount  = NULL;
                TypeHome  = fast;
                NumbApartament  = NULL;
                HomeRemont  = false;
        };
//////////////////////////// Класс Home
// Конструктор: номер дома и имя/номер для поиска (name)
Home::Home(const char *HomName, const char *Number, int  Numb) {
// Имя
            if ( HomName != (char *)NULL )
                 {name= new char[strlen(HomName) + 1];
                  strcpy_s(name ,  strlen(HomName) + 1, HomName); }
                else
                  name = (char *)NULL;
// Номер
                if ( Number != (char *)NULL )
                  {Home_Number = new char[strlen(Number) + 1];
                  strcpy_s(Home_Number ,  strlen(Number) + 1 , Number); }
                else
                   Home_Number = (char *)NULL;
//
                iHome      = Numb;
                EtagCount   = NULL;
                MenCount  = NULL;
                TypeHome  = fast;
                NumbApartament  = NULL;
                HomeRemont  = false;
        }
//////////////////////////// Класс Home
// Конструктор со всеми параметрами
Home::Home(const char *HomName, const char *Number, int  Numb,
                int Etag, int Men,HomeType Type,int Apart) {
//Имя
                if ( HomName != (char *)NULL )
                  {name= new char[strlen(HomName) + 1];
                  strcpy_s(name , strlen(HomName) + 1, HomName); }
                else
                  name = (char *)NULL;
// Номер
                if ( Number != (char *)NULL )
                  {Home_Number = new char[strlen(Number) + 1];
                  strcpy_s(Home_Number , strlen(Number) +1 , Number); }
                else
                   Home_Number = (char *)NULL;
```

```
//
                iHome      = Numb;
                EtagCount   = Etag;
                MenCount  = Men;
                TypeHome  = Type;
                NumbApartament  = Apart;
                HomeRemont  = false;
        }
//////////////////////////// Класс Home
// Конструктор копирования
Home Home::operator =(Home &  H )  {
// Имя
            if ( H.name != (char *)NULL )
                { name= new char[strlen(H.getName()) + 1];
                        strcpy_s(name , strlen(H.getName()) + 1 , H.getName());}
                 else
                   name = (char *)NULL;
// Номер
                 if ( H.Home_Number != (char *)NULL )
                   { Home_Number = new char[strlen(H.Home_Number) + 1];
                     strcpy_s(Home_Number , strlen(H.Home_Number) + 1, H.Home_Number); }
                 else
                    Home_Number = (char *)NULL;
//
      iHome = H.iHome;
                EtagCount   = H.EtagCount;
                MenCount  = H.MenCount;
                TypeHome  = H.TypeHome;
                NumbApartament  = H.NumbApartament;
                HomeRemont  = H.HomeRemont;
                return *this;
};
//////////////////////
// Методы класса Home
//////////////////////
// Метод печати дома
//////////////////
void Home::printOn(ostream & out) {
//
        out << "Номер сп. -" <<no ;
                if ( name !=(char *)NULL)
                {out <<  "  Имя - " <<  name << endl; }
                else
                {out <<  "  Имя не задано " <<  endl; };
//
      if ( Home_Number !=(char *)NULL)
                out <<          "Номер сим. -" <<  Home_Number << endl;
                else
                {out <<  "  Символьное имя не задано " <<  endl; };
//
                out  << "  Номер -" << iHome << endl <<
                "Этажей -" << EtagCount <<  "  Жителей -" << MenCount <<     endl;
                if (TypeHome == fast)
                   cout << "Тип дома - простой ";
                if (TypeHome == multiple)
                   cout << "Тип дома - много строений ";
                if (TypeHome == complex)
                   cout << "Тип дома - сложный ";
                if ( HomeRemont ) cout << "  Требуется ремонт! ";
                else cout << "  Ремонт не нужен! ";
                cout << "  Число квартир - " << NumbApartament <<  endl<< endl;
        }
```

```
//////////////////////////// Класс Home
// Метод - Установить имя
//////////////////
void Home::setName(const char *HomName , const char *Number){
 // Имя из базового класса - резерв для приска
              if ( name != NULL)
                            delete []name;
              if ( HomName != (char *)NULL )
                      {name= new char[strlen(HomName) + 1];
                        strcpy_s(name , strlen(HomName) + 1 , HomName); }
              else
                      name = (char *)NULL;
// Имя дома
              if ( Home_Number != NULL)
                            delete []Home_Number;

         if ( Number != NULL)
            {
                  Home_Number= new char[strlen(Number) + 1];
                  strcpy_s(Home_Number , strlen(Number) + 1 , Number);
                  }
                  else
                    Home_Number = (char *)NULL;
};
//////////////////////////// Класс Home
// Метод - Установить все параметры
//////////////////
void Home::setAllParam(const char *HomName , const char *Number, int  iH, int  Etag ,
      int  Men ,HomeType  Type, int  Apart , BOOL rem)
          {
 // Имя из базового класса - резерв для приска
              if ( name != NULL)
                            delete []name;
              if ( HomName != (char *)NULL )
                     {name= new char[strlen(HomName) + 1];
                      strcpy_s(name, strlen(HomName) + 1, HomName); }
              else
                      name = (char *)NULL;
 // Имя из базового класса - резерв для приска
              if ( Home_Number != NULL)
                            delete []Home_Number;

         if ( Number != NULL)
            {
                  Home_Number= new char[strlen(Number) + 1];
                  strcpy_s(Home_Number , strlen(Number) + 1, Number) ;
                  }
                  else
                  Home_Number = (char *)NULL;
//
                   iHome= iH;
                   EtagCount= Etag;
                   MenCount= Men;
                   TypeHome = Type;
                   NumbApartament = Apart;
                   HomeRemont  = rem;
              };
// Методы дружественные - friend
//////////////////////////// Класс Home
// Перегрузка сложения домов
//////////////////
 Home & operator +(Home & H1 , Home & H2)
```

```
 {
 Home *pTemp = new Home;  // новый объект для сложения
//
 pTemp->no = H1.no;
//  Имя базового - сумма имен
 pTemp->name = new char[strlen(H1.name) + strlen(H2.name)+ 5 ]; // учтем пробелы!
 pTemp->name[0]='\0';
 if (H1.name != NULL)
    strcpy_s(pTemp->name, strlen(H1.name) + 1 , H1.name);
 if (H2.name != NULL)
    {
    strcat_s(pTemp->name, strlen(H1.name) + strlen(H2.name)+ 5 ," + ");
    strcat_s(pTemp->name, strlen(H1.name) + strlen(H2.name)+ 5 ,H2.name);
    };
//  Имя = сумма имен Home_Number
 pTemp->Home_Number = new char[strlen(H1.Home_Number) + strlen(H2.Home_Number)+ 5 ];
 pTemp->Home_Number[0]='\0';
 if (H1.Home_Number != NULL)
   strcpy_s(pTemp->Home_Number, strlen( H1.Home_Number ) + 1  , H1.Home_Number);
 if (H2.Home_Number != NULL)
    {
    strcat_s(pTemp->Home_Number,strlen(H1.Home_Number) + strlen(H2.Home_Number)+ 5 ," + ");
    strcat_s(pTemp->Home_Number,strlen(H1.Home_Number) + strlen(H2.Home_Number)+ 5
,H2.Home_Number);
    };
// Параметры
      pTemp->   iHome      = H1.iHome;
      pTemp->   EtagCount   = H1.EtagCount;
      pTemp->   MenCount  = H1.MenCount + H2.MenCount;
      pTemp->   TypeHome  = complex;
      pTemp->   NumbApartament  = H1.NumbApartament + H2.NumbApartament;
      pTemp->HomeRemont  = (H1.HomeRemont || H2.HomeRemont ) ? true : false ;
 return *pTemp;
 };
//////////////////////////
// Класс улиц Street
//////////////////////
//// Конструкторы Street
//////////////////////
//// Конструкторы Street
////////////////// Класс Street
//  Конструктор: - Пустые параметры
//////////////////
Street::Street():AbstrStreet() {
// для базового класса
                    no = NULL;
                    name = (char *)NULL;
                    Name_Street = (char *)NULL;
                    Homes_num = NULL;
                    StrType = two;
                    Remont = false;
                    RemontStreet = false;
                    NumberStreet = NULL;
                    ListOfNear = (Street *)NULL;
          } ;
////////////////// Класс Street
// Конструктор: Имя улицы (Name_Street) и имя для поиска (name) общие
//////////////////
Street::Street(const char *sName) :AbstrStreet() {
                    if ( sName != (char *) NULL )
                      { name= new char[strlen(sName) + 1];
                      strcpy_s(name , strlen(sName) + 1 , sName);
```

```
                  Name_Street= new char[strlen(sName) + 1];
                  strcpy_s(Name_Street ,  strlen(sName) + 1 , sName); }
                else
                {
                  name = (char *)NULL;
                  Name_Street = (char *)NULL;
                };
                no = NULL;
                Homes_num = NULL;
                StrType = two;
                Remont = false;
                RemontStreet = false;
                NumberStreet = NULL;
                ListOfNear = (Street *)NULL;
        };
////////////////// Класс Street
// Конструктор: Имя улицы и имя для поиска разные
//////////////////
Street::Street(const char *sName , const char *sNumb)
                  :AbstrStreet(){
// Имя базового для поиска
                if ( sNumb != (char *) NULL )
                  { name= new char[strlen(sNumb) + 1];
                  strcpy_s(name , strlen(sNumb) + 1 , sNumb);
                  }
                else
                  name = (char *)NULL;
// Имя  улицы
                if ( sName != (char *) NULL )
                  {
                  Name_Street= new char[strlen(sName) + 1];
                  strcpy_s(Name_Street, strlen(sName) + 1 , sName); }
                else
                  Name_Street = (char *)NULL;
//
                no = NULL;
                Homes_num = NULL;
                StrType = two;
                Remont = false;
                RemontStreet = false;
                NumberStreet = NULL;
                ListOfNear = (Street *)NULL;
        };
////////////////// Класс Street
// Конструктор: Номер для поиска
//////////////////
Street::Street(int Num) {
      name = (char *)NULL;
                Name_Street = (char *)NULL;
                no = Num; // Номер из базового для поиска
                Homes_num = NULL;
                StrType = two;
                Remont = false;
                RemontStreet = false;
                NumberStreet = Num;
                ListOfNear = (Street *)NULL;
                };
////////////////// Класс Street
// Конструктор: Имя улицы и номер для поиска
//////////////////
        Street::Street(const char *sName , int Num) :AbstrStreet() {
// Имя улицы и для поиска
```

```
                if ( sName != (char *) NULL )
                  { name= new char[strlen(sName) + 1];
                  strcpy_s(name , strlen(sName) + 1, sName);
                  Name_Street= new char[strlen(sName) + 1];
                  strcpy_s(Name_Street, strlen(sName) + 1 , sName); }
                else
                {
                  name = (char *)NULL;
                  Name_Street = (char *)NULL;
                };
                no = Num;  // Номер для поиска
                Homes_num = NULL;
                StrType = two;
                Remont = false;
                RemontStreet = false;
                NumberStreet = Num;
                ListOfNear = (Street *)NULL;
        };
////////////////// Класс Street*!!!!!!!!!!!!!!!**
// Конструктор копирования
//////////////////
        Street::Street(Street & S) :AbstrStreet() {

                if ( S.name != (char *) NULL )
                  { name= new char[strlen(S.name) + 1];
                  strcpy_s(name , strlen(S.name) + 1, S.name);
                  }
                else
                  name = (char *)NULL;

                if ( S.Name_Street != (char *) NULL )
                  {
                  Name_Street= new char[strlen(S.Name_Street) + 1];
                  strcpy_s(Name_Street , strlen(S.Name_Street) + 1 ,  S.Name_Street); }
                else
                  Name_Street = (char *)NULL;

                no = S.no;
                Homes_num = S.Homes_num;
                StrType = S.StrType;
                Remont = S.Remont;
                RemontStreet = S.RemontStreet;
                NumberStreet = S.NumberStreet;
                ListOfNear = S.ListOfNear;
                // Цикл формирования новой улицы
        int nRazm = 0;
        nRazm = (int ) S.GetCount() ;
// !!! Для LIST!!!
POSITION pos = S.GetHeadPosition();
  for ( int i = 0 ;  i < nRazm ; i++)
  {
// !!! Для LIST!!!
AddTail( ((Home * ) S.GetAt(pos)));
S.GetNext( pos );
        };
///
        };
//////////////////////
// Методы класса Street
//////////////////////
////////////////// Класс Street*!!!!!!!!!!!!!!!**
// Метод печати улицы
```

```
//////////////////////
          void Street::printOn(ostream & out) {
      if ( Name_Street != (char *)NULL )
            { out << "{{{******* "<<endl<<"Улица -  "<< Name_Street ; }
            else
            {out << "Название не задано! "<<endl;};
    if ( name != (char *)NULL )
          { out << "  Ключ для поиска -  "<< name << endl ; }
      else
            {out << "Ключ не задан! "<<endl;};

       out << "Номер улицы -  "<< NumberStreet ;

                    out << "  Номер для поиска -  "<< no <<  endl;
                    out << "Число домов на улице -  "<< Homes_num ;
                    if ( RemontStreet )
                        out << "  Нужен ремонт улицы."<<  endl;
                    else
                        out << "  Улица отремонтирована."<<  endl;
                    if ( GetRemont() )
                        out << "Нужен ремонт домов улицы."<<  endl;
                    else
                        out << "Все эти дома отремонтированы."<<  endl;

                    int nRazm = 1;
                    nRazm = (int ) GetCount() ;
                    out << "Число в списке  = "<< GetCount()<<"}" << endl;

// Цикл просмотра списка
POSITION pos = GetHeadPosition();
//
for ( int i = 0 ;  i < nRazm ; i++)
  {
((Home *)(GetAt( pos )))->printOn(out);
GetNext( pos );
  }
                    out << "*******}}}" << endl;
          } ;

////////////////// Класс Street*!!!!!!!!!!!!!!!!!!**
/// Перегрузка присваивания для улицы
//////////////////////
Street Street::operator =(Street & S) {

                    if ( S.name != (char *) NULL )
                      { name= new char[strlen(S.name) + 1];
                      strcpy_s(name , strlen(S.name) + 1, S.name);
                      }
                    else
                      name = (char *)NULL;

                    if ( S.Name_Street != (char *) NULL )
                      {
                      Name_Street= new char[strlen(S.Name_Street) + 1];
                      strcpy_s(Name_Street , strlen(S.Name_Street) + 1 ,  S.Name_Street); }
                    else
                      Name_Street = (char *)NULL;

                    no = S.no;
                    Homes_num = S.Homes_num;
                    StrType = S.StrType;
                    Remont = S.Remont;
```

```
                RemontStreet = S.RemontStreet;
                NumberStreet = S.NumberStreet;
                ListOfNear = S.ListOfNear;
                // Цикл формирования новой улицы
        int nRazm = 0;
        nRazm = (int ) S.GetCount() ;
// Цикл добавления в списке!
POSITION pos = S.GetHeadPosition();
  for ( int i = 0 ;  i < nRazm ; i++)
  {
AddTail( ((Home * ) S.GetAt(pos)));
S.GetNext( pos );
        };
                return *this;
        };
////////////////// Класс Street*
// Метод вычисления необходимости ремонта домов улицы
////////////////////
//      HomeRemont
BOOL Street::GetRemont(){
          BOOL Flag = false;
                int nRazm = 0;
        nRazm = (int ) GetCount() ;
          POSITION pos = GetHeadPosition();
  for ( int i = 0 ;  i < nRazm ; i++)
  {
if (((Home * ) GetAt(pos))->HomeRemont  ) Flag = true ; // Хотя бы одна требует ремонта
GetNext( pos );
        };
return Flag;
};

////////////////// Класс Street*
// Метод  добавления дома на улицу (выбор варианта добавления)
////////////////////
        void Street::add(Home *pH , TypeAddDel t , int Numb , TypeAddDel tcr ) {
//
         if ( t == tail)    AddTail((CObject *) pH);
         if ( t == head) AddHead((CObject *) pH);
// Добавление по номеру
    if ( t == Number)
         {
           if( GetCount() < Numb )
                { cout << endl << "Добавление невозможно, проверьте номер!" <<  endl;
             return;  };
  int nRazm = (int ) GetCount() ;
  POSITION pos = GetHeadPosition();
  for ( int i = 0 ;  i < Numb ; i++)
  {
 // Навигация по списку
  GetNext( pos );
        };
  InsertAfter (pos , (CObject *) pH);
         };
// Запомним число домов на улице
         Homes_num =  (int) GetCount() ;
        } ;
////////////////// Класс Street*!!!!!!!!!!!!!!!!!
// Метод  удаления дома с улицы (выбор варианта удаления)
////////////////////
        void Street::del( TypeAddDel t , int Numb , TypeAddDel tdl) {
         if ( t == head) RemoveHead();
```

```
          if ( t == tail) RemoveTail();
// Удаление по номеру
     if ( t == Number)
          {
            if( GetCount() < Numb )
                 { cout << endl << "Удаление невозможно, проверьте номер!" <<  endl;
              return;  };
   int nRazm = (int ) GetCount() ;
   POSITION pos = GetHeadPosition();
   for ( int i = 0 ;  i < Numb ; i++)
   {
  // Навигация по списку
   GetNext( pos );
         };
   RemoveAt (pos );
          };
          Homes_num =  (int) GetCount() ;
         } ;
////////////////// Класс Street*!!!!!!!!!!!!!!!!!
// дружественная функция для операции сложения улиц
////////////////////
Street & operator +( Street &  X ,  Street &  Y )
{
 Street *pTemp = new Street;

 pTemp->name = new char[strlen(X.name) + strlen(Y.name) + 5 ];
 if (X.name != (char *)NULL )
 {  strcpy_s(pTemp->name, strlen(X.name) + 1, X.name);}
 else
   pTemp->name[0]='\0';
 if (Y.name != (char *)NULL )
 {  strcat_s(pTemp->name,  strlen(X.name) + strlen(Y.name) + 5 , " + ");
    strcat_s(pTemp->name, strlen(X.name) + strlen(Y.name) + 5  , Y.name); }
// Name_Street
 pTemp->Name_Street = new char[strlen(X.Name_Street) + strlen(Y.Name_Street) + 5];
 if (X.Name_Street != (char *)NULL )
 {  strcpy_s(pTemp->Name_Street, strlen(X.Name_Street) + 1 , X.Name_Street);}
 else
   pTemp->Name_Street[0]='\0';
 if (Y.Name_Street != (char *)NULL )
   {
   strcat_s(pTemp->Name_Street,strlen(X.Name_Street) + strlen(Y.Name_Street) + 5 ," + ");
   strcat_s(pTemp->Name_Street,strlen(X.Name_Street) + strlen(Y.Name_Street) + 5 ,Y.Name_Street);
   };
 // Параметры Street
   pTemp->Homes_num = NULL;
   pTemp->NumberStreet = X.NumberStreet;
   pTemp->Remont = (X.GetRemont() || Y.GetRemont()) ? true : false;      // для домов
   pTemp->RemontStreet = (X.RemontStreet || Y.RemontStreet) ? true : false; ;
   pTemp->StrType = ( X.StrType == two || Y.StrType == two) ? two : one ;
   pTemp->ListOfNear = (Street * )NULL;      // резерв, нужно сложить списки соседних улиц
 // Первый список
          int nRazm = 0;
   nRazm = (int ) X.GetCount() ;
   POSITION pos =  X.GetHeadPosition();
   for ( int i = 0 ;  i < nRazm ; i++)
   {
// Добавить на улицу
pTemp->AddTail( ((Home * ) X.GetAt(pos)));
 X.GetNext( pos );
         };
 // Второй список
```

```
  nRazm = (int ) Y.GetCount() ;
  pos =  Y.GetHeadPosition();
   for ( int i = 0 ;  i < nRazm ; i++)
   {
// Добавить на улицу
pTemp->AddTail( ((Home * )Y.GetAt(pos)));
 Y.GetNext( pos );        };
          pTemp->Homes_num =  (int )(pTemp->GetCount()) ;
 return *pTemp;
} ;
////////////////// Класс Street*!!!!!!!!!!!!!!!!!
// Метод подсчета числа жителей
////////////////////
int Street::GetNumberMens(){
   int Summ = 0;
  // Цикл
   int nRazm = 0;
            nRazm = (int ) GetCount() ;
            POSITION pos = GetHeadPosition();
            for ( int i = 0 ;  i < nRazm ; i++)
  {
  Summ = Summ +  ((Home *)GetAt( pos))->MenCount ;
  GetNext( pos );
         };
         return Summ;
     };
////////////////// Класс Street*!!!!!!!!!!!!!!
// Метод подсчета числа квартир
////////////////////
int Street::GetNumberApart(){
    int Summ = 0;
         int nRazm = 0;
         nRazm = (int ) GetCount() ;
           POSITION pos = GetHeadPosition();
           for ( int i = 0 ;  i < nRazm ; i++)
  {
  Summ = Summ +  ((Home *)GetAt( pos))->NumbApartament ;
  GetNext( pos );
         };
         return Summ;
   };
////////////////// Класс Street
// Метод установки названия улицы
////////////////////
void Street::SetNameStreet(const char * NameStr)
         {
         if (Name_Street != (char *) NULL)
           delete []Name_Street;
          if ( NameStr != (char *) NULL )
           {
           Name_Street= new char[strlen(NameStr) + 1];
           strcpy_s(Name_Street , strlen(NameStr) + 1, NameStr);
           }
          else
            Name_Street = (char *) NULL;

         return;};
////////////////// Класс Street
// Метод установки имени для поиска улицы
////////////////////
void Street::SetKeyNameStreet(const char * sName)
         {
```

```
            if (name != (char *) NULL)
              delete []name;
           if ( sName != (char *) NULL )
             {
             name= new char[strlen(sName) + 1];
             strcpy_s(name , strlen(sName) + 1 , sName);
             }
           else
              Name_Street = (char *) NULL;
          return;};
//////////////////////
```

Модуль - DZ_List.cpp (для списков)

// DZ_List.cpp

```
// Главный модуль проекта для отладки и сдачи системы классов (вариант списки)

#include "stdafx.h"
#include "DZ_Class.h"
#include <iostream>
using namespace std;

int  main(void)
//int mainMETOD()
{
int iPunkt;
system (" chcp 1251>nul ");
     while ( true ) {
system (" CLS ");
         cout << endl << "Меню  тестового примера системы классов улиц." << endl << endl;

// Новое меню
         cout << endl << "1.  ТЗ - 5.1.1  Создание улиц с домами " <<  endl;
   cout <<  "2.  ТЗ - 5.1.2  Создание объектов для  домов улицы  " <<  endl;
   cout <<  "3.  ТЗ - 5.1.3  Создание объектов для  домов улицы на основе других " <<  endl;
   cout <<  "4.  ТЗ - 5.1.4  Учет свойств дома(см. ТЗ) " <<  endl;
   cout <<  "5.  ТЗ - 5.1.5  Задание и получение характеристик дома " <<  endl;
   cout <<  "6.  ТЗ - 5.1.6  Сложение двух домов " <<  endl;
   cout <<  "7.  ТЗ - 5.1.7  Перегрузить оператор присваивания для домов " <<  endl;
   cout <<  "8.  ТЗ - 5.1.8  Распечатка характеристик дома " <<  endl;
   cout <<  "9.  ТЗ - 5.1.9  Учет свойств улицы(см. ТЗ) " <<  endl;
   cout <<  "10. ТЗ - 5.1.10 Распечатка содержания улицы и ее свойств " <<  endl;
   cout <<  "11. ТЗ - 5.1.11 Задание характеристик улицы " <<  endl;
   cout <<  "12. ТЗ - 5.1.12 Получение характеристик улицы " <<  endl;
   cout <<  "13. ТЗ - 5.1.13 Сложение двух улиц " <<  endl;
   cout <<  "14. ТЗ - 5.1.14 Добавление дома на улицу " <<  endl;
   cout <<  "15. ТЗ - 5.1.15 Удаление дома с улицы " <<  endl;
   cout <<  "16. ТЗ - 5.1.16 Установка и снятие признака ремонта улицы " <<  endl;
   cout <<  "17. ТЗ - 5.1.17 Автоматическое получение признака ремонта домов улицы " <<  endl;
   cout <<  "18. ТЗ - 5.1.18 Перегрузка оператора присваивания для улиц " <<  endl;

         cout << endl << "0.Выход " <<  endl;

//       system (" pause ");
         cin >> iPunkt;
//       cout << "Ввели - " << iPunkt<< endl;
         switch(iPunkt )
         {
//////////////////
```

```
        case 1:
cout << endl << "ТЗ - 5.1.1  Создание улиц с домами " <<  endl;
    {
        Street S1("Ленинский проспект");
        S1.printOn(cout);
        Home H1("Жилой","д.2", 7,2,3, fast , 5);
        Home H2("Магазин", "д.3", 3);
        Home H3("ДЭЗ", "д.4а", 4,2);
        S1.add(&H1);
        S1.add(&H2);
        S1.add(&H3);
        S1.printOn(cout);

        }
   system (" pause ");
   system (" pause ");

        break;
///
        case 2:
cout << "5.1.2  Создание объектов для  домов улицы  " <<  endl;
    {
        Home H1;
        Home H2("Жилой","д.2");
        Home H3("Жилой", "д.3", 3);
        Home H4("Жилой", "д.4а", 4,2);
        Home H5("ДЭЗ", "д.5", 5,2,3);
        Home H6("Жилой", "д.6", 6,2,3, fast);
        Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
        H1.printOn(cout);
        H2.printOn(cout);
        system (" pause ");
        H3.printOn(cout);
        H4.printOn(cout);
        system (" pause ");
        H5.printOn(cout);
        H6.printOn(cout);
        system (" pause ");
        H7.printOn(cout);
        }
   system (" pause ");

        break;
///
        case 3:
cout << "5.1.3  Создание объектов для  домов улицы на основе других  " <<  endl;
    {
        Home H6("Жилой", "д.6", 6,2,3, fast, 100);
        Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
        Home Test(H6);
        H6.printOn(cout);
        Test.printOn(cout);
        system (" pause ");
        system (" pause ");


cout << "Указатель!!! " <<  endl;

        Home *pHome = new Home (H7);
        H7.printOn(cout);
        pHome->printOn(cout);
        delete pHome;
```

```
        }
   system (" pause ");

        break;
//
        case 4:
cout << "5.1.4  Учет свойств дома(см. ТЗ) " <<  endl;
    {
        Home H6("Жилой", "д.6", 6,2,3, fast, 100);
        H6.printOn(cout);
        Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
        H7.printOn(cout);
        }
   system (" pause ");
   system (" pause ");

        break;
//
        case 5:
cout << "5.1.5  Задание и получение характеристик дома " <<  endl;
    {
        Home H6("Жилой", "д.6", 6,2,3, fast, 100);
        H6.printOn(cout);
    int  iH,  Etag , Men ,  Apart ;
        HomeType Type;
        // Получение
    H6.getParam(iH , Etag, Men, Type , Apart);
    cout << "Номер -" << iH <<
          "  Этажей -" << Etag <<  "  Жителей -" << Men <<     endl;
                if (Type == fast)
                   cout << "Тип дома - простой ";
                if (Type == multiple)
                   cout << "Тип дома - много строений ";
                if (Type == complex)
                   cout << "Тип дома - сложный ";
                cout << "  Число квартир - " << Apart <<  endl;
        H6.setParam( 11,12,13,fast, 15 );
        H6.printOn(cout);

        system (" pause ");
    system (" pause ");
        // Задание

        cout << "Признак ремонта!!! " <<  endl;
        Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
        H7.printOn(cout);
        H7.setAllParam("Аптека", "10/8", 1,2,3,fast, 5 , true);
        H7.printOn(cout);

        }
    system (" pause ");

        break;
//

                case 6:
cout << "5.1.6  Сложение двух домов " <<  endl;
    {
  Home H1("Жилой", "д.6", 6,2,3, fast , 3);
  Home H2("Ашан", "д.7", 9,10,11, multiple , 5);
  Home Temp;
  Temp = H1 + H2;
```

```
  H1.printOn(cout);
  H2.printOn(cout);
  Temp.printOn(cout);
        }
   system (" pause ");
   system (" pause ");
//
        break;
//

                case 7:
cout << "5.1.7  Перегрузить оператор присваивания для домов " <<  endl;
    {
  Home H1("Жилой", "д.6", 6,2,3, fast , 3);
  Home Temp;
  Temp = H1;
  H1.printOn(cout);
  Temp.printOn(cout);
  H1.setName("Перекресток");
  H1.printOn(cout);
  Temp.printOn(cout);

        }
   system (" pause ");
   system (" pause ");
//
        break;

//

                case 8:
cout << "5.1.8  Распечатка характеристик дома " <<  endl;
    {
  Home H1("Жилой", "д.6", 6,2,3, fast , 3);
  H1.printOn(cout);
        }
   system (" pause ");
   system (" pause ");
//
                break;
//

                case 9:
cout << "5.1.9  Учет свойств улицы(см. ТЗ) " <<  endl;
    {
        Home H2("Жилой","д.2", 7,2,3, fast , 5);
        Home H3("Магазин", "д.3", 3);
        Home H4("ДЭЗ", "д.4а", 4,2);
        Street SNew("Улица" , 15);
        SNew.printOn(cout);
        SNew.add(&H2);
        SNew.add(&H3);
        SNew.add(&H4);
        SNew.printOn(cout);
        }
   system (" pause ");
   system (" pause ");
//
           break;
//

                case 10:
```

```
cout << "5.1.10 Распечатка содержания улицы и ее свойств" <<  endl;
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Home H4("ДЭЗ", "д.4а", 4,2);
          Street SNew("Улица" , 15);
          SNew.printOn(cout);
          SNew.add(&H2);
          SNew.add(&H3);
          SNew.add(&H4);
          SNew.printOn(cout);

          }
   system (" pause ");
   system (" pause ");
//
                  break;
//

                  case 11:
cout << "5.1.11 Задание характеристик улицы  " <<  endl;
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Home H4("ДЭЗ", "д.4а", 4,2);
          Street Sumstreet("Улица с параметрами" , 15);
          Sumstreet.add(&H2);
          Sumstreet.add(&H3);
          Sumstreet.add(&H4);
          Sumstreet.printOn(cout);
    cout <<  "******* Изменения параметров *********" <<  endl;

          Sumstreet.SetNameStreet("Новая");
          Sumstreet.SetKeyNameStreet("Новая ключ");
          Sumstreet.SetNumbStreet( 33 );
          Sumstreet.SetKeyNumbStreet( 77 );
          Sumstreet.printOn(cout);

 cout <<  "******* Параметры *********" <<  endl;
 cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
 cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
 cout <<  "Название улицы для поиска->  " << Sumstreet.GetKeyNameStreet() << endl;
 cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() << endl;
 cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
 cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
 cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
  if ( Sumstreet.GetRemont() )
    cout <<  "На улице нужен ремонт домов!" << endl;
 else
    cout <<  "На улице не нужен ремонт домов!" << endl;
  if ( Sumstreet.GetStreetType() == one  )
   cout <<  "Тип улицы -> одностороннее движение" <<  endl;
     if ( Sumstreet.GetStreetType() == two  )
   cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;


          }
   system (" pause ");
   system (" pause ");
//
             break;
//
```

```
                case 12:
cout << "5.1.12 Получение характеристик улицы " <<  endl;
    {
                Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Home H4("ДЭЗ", "д.4а", 4,2);
          Street Sumstreet("Улица с параметрами" , 15);
          Sumstreet.add(&H2);
          Sumstreet.add(&H3);
          Sumstreet.add(&H4);
          Sumstreet.printOn(cout);
 cout <<  "******* Параметры *********" <<  endl;
 cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
 cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
 cout <<  "Название улицы для поиска->  " << Sumstreet.GetKeyNameStreet() << endl;
 cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() << endl;
 cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
 cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
 cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
  if ( Sumstreet.GetRemont() )
     cout <<  "На улице нужен ремонт домов!" << endl;
 else
     cout <<  "На улице не нужен ремонт домов!" << endl;
   if ( Sumstreet.GetStreetType() == one  )
    cout <<  "Тип улицы -> одностороннее движение" <<  endl;
      if ( Sumstreet.GetStreetType() == two  )
    cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;

          }
    system (" pause ");
    system (" pause ");
//
             break;
//
                case 13:
cout << "5.1.13 Сложение двух улиц" <<  endl;
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Home H4("ДЭЗ", "д.4а", 4,2);
         Street Astreet("Первая", 20);
          Astreet.add(&H2 , head);
          Astreet.add(&H3 , head);
          Astreet.add(&H4 , head);
         Astreet.printOn(cout);
          Home H5("Аптека","д.2", 7,2,3, fast , 5);
          Home H6("Перекресток", "д.3", 3);
          Home H7("Детский сад", "д.4а", 4,2);
         Street Dstreet("Вторая ", 20);
          Dstreet.add(&H5 , tail);
          Dstreet.add(&H6 , tail);
          Dstreet.add(&H7 , tail);
          Dstreet.printOn(cout);
      system (" pause ");
      system (" pause ");

 cout << endl << "******* Сложение *********" <<  endl;
           Street Sumstreet("  ", 20);
           Sumstreet.printOn(cout);
           Sumstreet = Astreet + Dstreet;
           Sumstreet.printOn(cout);
           system (" pause ");
```

```
 cout <<  "******* Параметры *********" <<  endl;
 cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
 cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
 cout <<  "Название улицы для поиска->  " << Sumstreet.GetKeyNameStreet() << endl;
 cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() << endl;
 cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
 cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
 cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
  if ( Sumstreet.GetRemont() )
     cout <<  "На улице нужен ремонт домов!" << endl;
 else
     cout <<  "На улице не нужен ремонт домов!" << endl;
   if ( Sumstreet.GetStreetType() == one  )
    cout <<  "Тип улицы -> одностороннее движение" <<  endl;
      if ( Sumstreet.GetStreetType() == two  )
    cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;
          }
    system (" pause ");
    system (" pause ");
//
             break;
//
                    case 14:
cout << "5.1.14 Добавление дома на улицу" <<  endl;
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Street S1("Улица с параметрами" , 15);
          S1.add(&H2);
          S1.add(&H3);
          S1.printOn(cout);
          system (" pause ");
    system (" pause ");
          Home H5("Аптека","д.2", 7,2,3, fast , 5);
          Home H6("Перекресток", "д.3", 3);
          S1.add(&H5 , head);
          S1.add(&H6 , head);
          S1.printOn(cout);
    system (" pause ");
          Home H7("Детский сад", "д.4а", 4,2);
          S1.add(&H7 , tail);
          S1.printOn(cout);
    system (" pause ");
          Home H8("Жилой 3","д.2", 70,20,30, fast , 50);
          S1.add(&H8 , Number, 2);
    S1.printOn(cout);
          }
    system (" pause ");
    system (" pause ");
//
             break;
//

                    case 15:
cout << "5.1.15 Удаление дома с улицы" <<  endl;
    {
          Home H2("Жилой","д.2", 7,2,3, fast , 5);
          Home H3("Магазин", "д.3", 3);
          Street S1("Улица с параметрами" , 15);
          S1.add(&H2);
          S1.add(&H3);
          Home H5("Аптека","д.2", 7,2,3, fast , 5);
```

```
        Home H6("Перекресток", "д.3", 3);
        S1.add(&H5 , head);
        S1.add(&H6 , head);
        Home H7("Детский сад", "д.4а", 4,2);
        S1.add(&H7 , tail);
        Home H8("Жилой 3","д.2", 70,20,30, fast , 50);
        S1.add(&H8 , Number, 2);
    S1.printOn(cout);
    system (" pause ");
    system (" pause ");
    cout << "Удаление дома с улицы конец !!!" <<  endl;
    Home Temp;
        S1.del(); //
        S1.printOn(cout);
        system (" pause ");
    cout << "Удаление дома с улицы начало!!!" <<  endl;
        S1.del( head);
        S1.printOn(cout);
    system (" pause ");
        cout << "Удаление дома с улицы второго!!!" <<  endl;

        S1.del( Number, 2);
        S1.printOn(cout);
        }
   system (" pause ");
//
            break;
//
                case 16:
cout << "5.1.16 Установка и снятие признака ремонта улицы " <<  endl;
    {
        Home H2("Жилой","д.2", 7,2,3, fast , 5);
        Home H3("Магазин", "д.3", 3);
        Street S1("Улица 1" , 15);
        S1.add(&H2);
        S1.add(&H3);
        S1.printOn(cout);
          if ( S1.GetRemontStr() )
    cout <<  "Улице нужен ремонт!" << endl;
      else
    cout <<  "Улице не нужен ремонт!" << endl;

        cout << "После установки!!!!" <<  endl;

    S1.SetRemontStr(true);
          if ( S1.GetRemontStr() )
    cout <<  "Улице нужен ремонт!" << endl;
      else
    cout <<  "Улице не нужен ремонт!" << endl;

        cout << "После снятия признака ремонта!!!!" <<  endl;

    S1.SetRemontStr(false);
          if ( S1.GetRemontStr() )
    cout <<  "Улице нужен ремонт!" << endl;
      else
    cout <<  "Улице не нужен ремонт!" << endl;

        }
   system (" pause ");
   system (" pause ");
//
```

```
            break;
//

                case 17:
cout << "5.1.17 Автоматическое получение признака ремонта домов улицы " <<  endl;
    {
        Home H2("Жилой","д.2", 7,2,3, fast , 5);
        Home H3("Магазин", "д.3", 3);
        Street S1("Улица 1" , 15);
        S1.add(&H2);
        S1.add(&H3);
        cout << "До установки признака ремонта дома и вычисления признака ремонта домов
улицы!!!!" <<  endl;
        S1.printOn(cout);
          if ( S1.GetRemont() )
    cout <<  "На улице нужен ремонт домов!" << endl;
      else
    cout <<  "На улице не нужен ремонт домов!" << endl;
    S1.del( tail);
        H3.setAllParam("Магазин", "д.3", 1,2,3,fast, 3 , true);
    H3.printOn(cout);
        S1.add(&H3);
        S1.GetRemont();
        cout << "После вычисления признака ремонта домов улицы!!!!" <<  endl;
        S1.printOn(cout);
          if ( S1.GetRemont() )
    cout <<  "На улице нужен ремонт домов!" << endl;
      else
    cout <<  "На улице не нужен ремонт домов!" << endl;
        }
   system (" pause ");
   system (" pause ");
//
            break;
//

                case 18:
cout << "5.1.18 Перегрузка оператора присваивания для улиц " <<  endl;
    {
        Home H2("Жилой","д.2", 7,2,3, fast , 5);
        Home H3("Магазин", "д.3", 3);
        H3.setAllParam("Магазин", "д.3", 1,2,3,fast, 3 , true);
        Street S1("Улица 1" , 15);
        Street SNew("Улица" , 15);
        S1.add(&H2);
        S1.add(&H3);
        S1.printOn(cout);
        SNew.printOn(cout);
    SNew = S1;
    SNew.printOn(cout);

    S1.del( head);

        system (" pause ");
    system (" pause ");
        S1.SetNameStreet("Новое название S1 ");
        cout << "После изменения S1 (название и удален первый)!!!!" <<  endl;
        cout << "S1!!!!" <<  endl;
        S1.printOn(cout);
        cout << "SNew!!!!" <<  endl;
    SNew.printOn(cout);
```

```
            }
    system (" pause ");
//
                break;
//


//////////////////

////////////////////////////////
            case 0:
            case 88-48:
            case 120-48:

cout << endl << "Выход " <<  endl;
system(" PAUSE");

            return 0;
            default:
cout << endl << "Выбор функции не верен! " <<  endl;
     break;
/// При  выходе
cout << endl << "Выход " <<  endl;
 system(" PAUSE");
            return 0;
           }
// system(" PAUSE");
};

 ///
 system(" PAUSE");
           return 0;
}


//////////////////////////////// Для отладки
//int  main(void)
int  mainDEBUG(void)
{
system(" chcp 1251 > nul");

int iPunkt;

      while ( true ) {
system (" CLS ");

           cout << endl << "Меню  тестового примера системы классов улиц." << endl << endl;
           cout << endl << "1.Конструкторы Home " <<  endl;
           cout << endl << "2.Методы Home " <<  endl;
           cout << endl << "3.Конструкторы Street " <<  endl;
           cout << endl << "4.Методы Street " <<  endl;
           cout << endl << "0.Выход " <<  endl;

           cin >> iPunkt;

//         cout << "Ввели - " << iPunkt<< endl;
           switch(iPunkt)
           {
            case 1:
cout << endl << "1.Конструкторы Home " <<  endl;
////////////////////////////////
            {
```

```
            Home H1;
            Home H2("Жилой","д.2");
            Home H3("Жилой", "д.3", 3);
            Home H4("Жилой", "д.4а", 4,2);
            Home H5("ДЭЗ", "д.5", 5,2,3);
            Home H6("Жилой", "д.6", 6,2,3, fast);
            Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
            H1.printOn(cout);
            H2.printOn(cout);
            H3.printOn(cout);
            H4.printOn(cout);
            H5.printOn(cout);
            H6.printOn(cout);
            H7.printOn(cout);
// Конструктор копирования
            Home Test(H5);
            H5.printOn(cout);
            Test.printOn(cout);
            Home *pHome = new Home (H6);
            H6.printOn(cout);
            pHome->printOn(cout);
            Home Temp;
            Temp = H7;
    H7.printOn(cout);
            Temp.printOn(cout);
cout << endl << "Указатели!!!!!!!!! " <<  endl;

            Home *pHome1 = new Home (* pHome);
     pHome->printOn(cout);
            pHome1->printOn(cout);
            delete pHome;
            delete pHome1;
            }
//////////////////////////////////////
//////////////////////////////////////
            break;
            case 2:
cout << endl << "2.Методы Home " <<  endl;
//////////////////////////////////////
            {
// + , SetName,printOn,classType,className, getName, getNumb,getno, getParam
// +
   Home H1("Жилой", "д.6", 6,2,3, fast , 3);
   Home H2("Жилой", "д.7", 9,10,11, fast , 5);
   Home Temp;
   Temp = H1 + H2;
   H1.printOn(cout);
   H2.printOn(cout);
   Temp.printOn(cout);
// SetName
   Temp.setName("Университет");
   Temp.printOn(cout);
   Temp.setName("Магазин" ,"д.11/12");
   Temp.printOn(cout);
//classType
  cout << "Тип класса = "<< Temp.classType() << endl;
//className
  cout << "Name класса = "<< Temp.className() << endl;
//getName
  cout << "Name дома = "<< Temp.getName() << endl;
//getNumb
  cout << "Номер дома = "<< Temp.getNumb() << endl;
```

```
//getno
  cout << "Номер в списке = "<< Temp.getNo() << endl;
//getParam
  cout << "Параметры: "<< endl;
  int  iH,  Etag , Men ,  Apart ;
  HomeType  Type;
  Temp.getParam(iH , Etag, Men, Type , Apart);
    cout << "Номер -" << iH <<
            "  Этажей -" << Etag <<  "  Жителей -" << Men <<     endl;
                  if (Type == fast)
                     cout << "Тип дома - простой ";
                  if (Type == multiple)
                     cout << "Тип дома - много строений ";
                  if (Type == complex)
                     cout << "Тип дома - сложный ";
                  cout << "  Число квартир - " << Apart <<  endl;
           Temp.printOn(cout);
           Temp.setParam(1,2,3,fast, 5 );
           Temp.printOn(cout);
           Temp.setAllParam("Аптека", "10/8 кв.3", 11,12,13,fast, 15 );
           Temp.printOn(cout);
           }
//////////////////////////////////////
           break;
           case 3:
cout << endl << "3.Конструкторы Street " <<  endl;
           {
           Street S1;
           Street S2("Лениский проспект");
           Street S3("Ленин","Лениский проспект");
           Street S4(5);
           Street S5("Горького ул." , 7);
// Пучтые улицы
           S1.printOn(cout);
           S2.printOn(cout);
           S3.printOn(cout);
           S4.printOn(cout);
    Home H5("ДЭЗ", "д.5", 5,2,3);
           Home H6("Жилой", "д.6", 6,2,3, fast);
           Home H7("Магазин", "д.7", 7,2,3, multiple , 5);
           S5.AddTail( (CObject *)&H5);
           S5.AddTail( (CObject *)&H6);
           S5.AddTail( (CObject *)&H7);
           S5.printOn(cout);

//////////////////////////////
           Street S6(S5);
           S6.printOn(cout);
           Home H2("Жилой","д.2", 7,2,3, fast , 5);
           Home H3("Магазин", "д.3", 3);
           Home H4("ДЭЗ", "д.4а", 4,2);

           Home HNew("Сарай","д.10/12", 7,2,3, fast , 5);
           Street SNew("Улица" , 15);
           SNew.printOn(cout);
           SNew.AddHead((CObject *)&HNew);
           SNew.AddHead((CObject *)&H2);
           SNew.AddHead((CObject *)&H3);
           SNew.AddHead((CObject *)&H4);
           SNew.printOn(cout);
           cout << endl << "******* Конструктор копии *********" <<  endl;
           Street SCopy(SNew);
```

```
            SCopy.printOn(cout);
            cout << endl << "******* Перегрузка присваивания *********" <<  endl;
    Street SEmpty("Пусто!!" , 15);
            SEmpty.printOn(cout);
    SEmpty = SCopy;
            SEmpty.printOn(cout);

            // МЕТОД add
Street Shead("Улица Head" , 15);
            Shead.add(&HNew , head);
            Shead.add(&H2 , head);
            Shead.add(&H3 , head);
            Shead.add(&H4 , head);
            Shead.printOn(cout);
           ///
            Shead.del( head);
            Shead.printOn(cout);
            Shead.del( tail);
            Shead.printOn(cout);
//          Shead.del( Number , 1);
//          Shead.printOn(cout);
            };
            break;


            case 4:
cout << endl << "4.Методы Street " <<  endl;
            {
            Home H2("Жилой","д.2", 7,2,3, fast , 5);
            Home H3("Магазин", "д.3", 3);
            Home H4("ДЭЗ", "д.4а", 4,2);
           Street Astreet("Добавление head", 20);
           Astreet.printOn(cout);
           cout << endl << "******* Добавление head *********" <<  endl;
            Astreet.add(&H2 , head);
            Astreet.add(&H3 , head);
            Astreet.add(&H4 , head);
            Astreet.printOn(cout);

            Home H5("Жилой 2","д.2", 7,2,3, fast , 5);
            Home H6("Магазин 2", "д.3", 3);
            Home H7("ДЭЗ 2", "д.4а", 4,2);
cout << endl << "******* Добавление tail err *********" <<  endl;
           Street Dstreet("Добавление tail ", 20);
            Dstreet.add(&H5 , tail);
            Dstreet.add(&H6 , tail);
            Dstreet.add(&H7 , tail);
            Dstreet.printOn(cout);
 cout << endl << "******* Добавление после NUM 1 (0-первый) *********" <<  endl;
            Home H8("Жилой 3","д.2", 70,20,30, fast , 50);
            Home H9("Магазин 3", "д.3", 3);
            Home H10("ДЭЗ 3", "д.4а", 4,2);
            Dstreet.add(&H8 , Number, 1);
            Dstreet.add(&H9 , Number, 1);
            Dstreet.add(&H10 , Number, 1);
            Dstreet.printOn(cout);

            cout << endl << "******* Удаление *********" <<  endl;
            Dstreet.del(); //
            Dstreet.printOn(cout);
            Dstreet.del( head);
            Dstreet.printOn(cout);
```

```
            Dstreet.del( Number, 2);
            Dstreet.printOn(cout);
//          Dstreet.del( tail);
//          Dstreet.printOn(cout);

cout << endl << "******* Сложение *********" <<  endl;
             Street Sumstreet("  ", 20);
             Sumstreet.printOn(cout);
             Sumstreet = Astreet + Dstreet;
             Sumstreet.printOn(cout);



cout <<  "******* Параметры *********" <<  endl;
 cout <<  "Название улицы ->  " << Sumstreet.GetNameStreet() << endl;
 cout <<  "Номер улицы ->  " << Sumstreet.GetNumbStreet() << endl;
 cout <<  "Название улицы для поиска->  " << Sumstreet.GetKeyNameStreet() << endl;
 cout <<  "Номер улицы для поиска->  " << Sumstreet.GetKeyNumbStreet() << endl;
 cout <<  "Число домов на улице = " << Sumstreet.GetNumberHome() << endl;
 cout <<  "Число жителей на улице = " << Sumstreet.GetNumberMens()  << endl;
 cout <<  "Число квартир на улице = " << Sumstreet.GetNumberApart() << endl;
  if ( Sumstreet.GetRemont() )
     cout <<  "На улице нужен ремонт домов!" << endl;
 else
     cout <<  "На улице нужен ремонт домов!" << endl;
   if ( Sumstreet.GetStreetType() == one  )
    cout <<  "Тип улицы -> одностороннее движение" <<  endl;
      if ( Sumstreet.GetStreetType() == two  )
    cout <<  "Тип улицы -> двухсторонне  движение" <<  endl;
 cout <<  "******* Изменения параметров *********" <<  endl;

          Sumstreet.SetNameStreet("Новая");
          Sumstreet.SetKeyNameStreet("Новая ключ");
          Sumstreet.SetNumbStreet( 33 );
          Sumstreet.SetKeyNumbStreet( 77 );
 cout <<  "Название улицы 2->  " << Sumstreet.GetNameStreet() << endl;
 cout <<  "Номер улицы 2->  " << Sumstreet.GetNumbStreet() << endl;
 cout <<  "Название улицы для поиска 2->  " << Sumstreet.GetKeyNameStreet() << endl;
 cout <<  "Номер улицы для поиска 2->  " << Sumstreet.GetKeyNumbStreet() << endl;

           };
           break;
//////////////////////////////////
           case 0:
           case 88-48:
           case 120-48:

cout << endl << "Выход " <<  endl;
system(" PAUSE");

           return 0;
//
           default:
cout << endl << "Выбор функции не верен! " <<  endl;
     break;
/// При  выходе
cout << endl << "Выход " <<  endl;
 system(" PAUSE");
           return 0;
          }
 system(" PAUSE");
};
```

```
 ///
 system(" PAUSE");
        return 0;
}
```