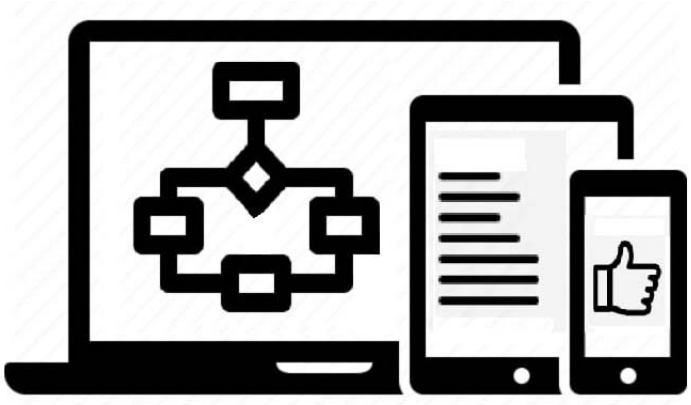
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические рекомендации к лабораторным работам для студентов направлений подготовки 09.03.04 «Программная инженерия» и 09.03.01 «Информатика и вычислительная техника» дневной формы обучения

Часть 1



Могилев 2023

УДК 621.01
ББК 36.4
О87

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «28» марта 2023 г., протокол № 9

Составители: ст. преподаватель О. В. Сергиенко;
канд. техн. наук, доц. Ю. В. Вайнилович

Рецензент Ю. С. Романович

Методические рекомендации к лабораторным работам предназначены для студентов направлений подготовки 09.03.04 «Программная инженерия» и 09.03.01 «Информатика и вычислительная техника» дневной формы обучения.

Учебное издание

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Часть 1

Ответственный за выпуск	В. В. Кутузов
Корректор	А. А. Подошевко
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

При изучении дисциплины «Объектно-ориентированное программирование» студенты выполняют лабораторные работы, варианты которых приведены в данных методических рекомендациях.

Каждая лабораторная работа соответствует темам лекций и содержит в себе десять вариантов индивидуальных заданий.

Варианты заданий выдаются студентам заранее с тем, чтобы они имели возможность подготовиться к выполнению лабораторной работы: просмотреть теоретический материал по теме работы и продумать алгоритмы решения задач.

Программы пишутся на языке C#. Каждую программу в работающем виде (после отладки и тестирования) студент показывает преподавателю, после чего лабораторная работа подлежит защите.

К защите работы студент подготавливает отчет, включающий в себя титульный лист, формулировку задания, описание исходных, результирующих данных и вспомогательных переменных, алгоритм решения задачи, текст программы и результаты ее тестирования.

Защита лабораторной работы состоит из двух частей: практической и теоретической. В практической части студент объясняет принципы работы представленной им программы, в теоретической – отвечает на вопросы по теме лабораторной работы.

1 Лабораторная работа № 1. Классы и объекты

Цель работы – усвоение понятий класса, объекта, определения членов класса, овладение приемами разработки простейших программ в объектно-ориентированном стиле программирования.

Теоретические сведения

Классы: основные понятия.

Класс является типом данных, определяемым пользователем; он описывает признаки состояния и поведение множества схожих объектов окружающей действительности.

Элементами класса являются данные и методы, предназначенные для их обработки.

Описание класса имеет вид:

[атрибуты][спецификаторы] class <имя_класса> [: предки]
{ тело_класса }

Данные: поля и константы.

Данные, содержащиеся в классе, могут быть переменными или константами. Переменные, описанные в классе, называются полями класса. Синтаксис описания элемента данных:

[атрибуты][спецификаторы][const] <тип> <имя> [= <начальное значение>]

По умолчанию элементы класса считаются закрытыми (private). Все методы класса имеют непосредственный доступ к его закрытым полям.

Все поля сначала автоматически инициализируются нулем соответствующего типа. После этого полю присваивается значение, заданное при его явной инициализации. Задание начальных значений для статических полей выполняется при инициализации класса, а обычных – при создании экземпляра.

Поля, описанные со спецификатором static, а также константы существуют в единственном экземпляре для всех объектов класса, поэтому к ним обращаются через имя класса. Обращение к полю класса выполняется с помощью операции доступа – точки:

- для обычных полей: <имя экземпляра>.<имя поля>;
- для статических: <имя класса>.<имя поля> .

Методы.

Методы класса имеют непосредственный доступ к его полям. Метод, описанный со спецификатором static, должен обращаться только к статическим полям класса. Статический метод вызывается через имя класса, а обычный – через имя экземпляра.

Например, для статического метода класса Class1 с сигнатурой

static void Poisk(int a, ref int b, out int c);

вызов может быть следующим: Poisk(1, ref x, out y).
Для метода с сигнатурой void Out()необходимо создать объект:

Class1 myob = new Class1();

а затем вызвать метод myob.Out().

Конструкторы.

Конструктор предназначен для инициализации полей объекта. Он вызывается автоматически при создании объекта класса с помощью операции new.
Формат записи конструктора:

[спецификатор] <имя_класса>()
{тело конструктора}

Обычно в качестве элемента «спецификатор» используется модификатор доступа public, поскольку конструкторы, как правило, вызываются вне их класса.
Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации. Конструктор, вызываемый без параметров, называется конструктором по умолчанию. Если в классе не указан ни один конструктор или какие-то поля не были инициализированы, полям значимых типов присваивается нуль соответствующего типа, полям ссылочных типов – значение null.
Создание объекта выполняется операцией

<имя_переменной_типа_класса> = new <имя_класса>();

Свойства.

Свойство – это специальный тип членов класса, который предназначен для организации доступа к закрытым полям класса и определяет методы его получения и установки.
Формат записи свойства:

[атрибуты][спецификаторы] <тип> <имя_свойства>
{[get {<код аксессора чтения поля> //код доступа}]
[set {<код аксессора записи поля> // код доступа}]}

Значения спецификаторов для свойств и методов аналогичны. Чаще всего свойства объявляются как открытые (со спецификатором public). После определения свойства любое использование его имени означает вызов соответствующего аксессора. Код доступа представляет собой блоки операторов, которые выполняются при получении (get) или установке (set) свойства. Может отсутствовать

либо часть get, либо set, но не обе одновременно. Если отсутствует часть set, свойство доступно только для чтения, если отсутствует часть get, свойство доступно только для записи.

Задания для самостоятельного выполнения

Создайте проект, в котором опишите класс для решения задачи вашего варианта.

Указания. Разрабатываемый класс должен содержать следующие элементы: скрытые поля, конструкторы без параметров и с параметрами (имена некоторых полей должны совпадать с идентификаторами параметров), свойства, метод вывода полей и указанный в таблице метод.

Составьте тестирующую программу с выдачей результатов. В программе должна выполняться проверка всех разработанных элементов класса, вывод состояния объекта (таблица 1.1).

Таблица 1.1 – Варианты заданий

Номер варианта	Класс	Метод
1, 13	Сотрудник (поля – имя, p – минимальная зарплата)	Доход: $k*p$, где k – повышающий коэффициент
2, 14	Квартира (поля – номер, стоимость 1 м^2, площадь)	Стоимость квартиры
3, 15	Агрополе (поля – название, вес r посеянных семян на единицу площади, S – площадь, га)	Урожай: $k*r$, где k – коэффициент, зависящий от культуры
4, 16	Стол (поля – название, площадь S, см)	Стоимость: $S^2/3+500000$
5, 17	Автобус (поля – количество пассажиров, стоимость билета)	Выручка
6, 18	Транспортное средство (поля – название, расстояние, цена за 1 км)	Стоимость проезда
7, 19	Пособие (поля – повышающий коэффициент k, минимальное пособие r)	Размер пособия: $k*r$
8, 20	Телефон (поля – марка, количество функций – k)	Стоимость: $40*ln(k)$
9	Автомобиль (поля – марка, расход горючего на 100 км N, расстояние R, км)	Объем горючего: $N*R$
10	Одежда (поля – модель, ширина ткани H, м, норма расхода – L, м)	Расход ткани: $(2 + H)*L$
11	Постройка (поля – название, высота здания V, м)	Высота фундамента: $0,03*V$
12	Аудитория (поля – номер, площадь S, м^2)	Количество мест: S/1,2
13	Геометрическая фигура прямоугольник (поля – высота h, мм, ширина w, мм)	Площадь прямоугольника $A = h*w$

Контрольные вопросы

1 Дайте определение терминам класс и объект. Как соотносятся эти понятия между собой?

2 Приведите синтаксис создания объекта в общем виде. Проиллюстрируйте его фрагментом программы.

3 Какие члены класса содержат код?

4 Какие члены класса содержат данные?

5 В чём состоит назначение конструктора?

2 Лабораторная работа № 2. Статические члены класса

Цель работы – усвоение основных приемов создания классов с полями типа массив; формирование знаний о возможности использования индексаторов для доступа к элементам закрытых массивов.

Теоретические сведения

Цикл foreach.

Цикл foreach используется для опроса элементов коллекции. Формат записи цикла имеет вид:

foreach (<тип> <имя_переменной> in <коллекция>)
<тело цикла;>

Здесь элементы тип и имя_переменной задают тип и имя итерационной переменной, которая при выполнении цикла foreach будет последовательно получать значения элементов из коллекции.

Элемент коллекция служит для указания опрашиваемой коллекции. Таким образом, элемент тип должен совпадать (или быть совместимым) с базовым типом массива. С помощью foreach невозможно изменить содержимое коллекции.

Цикл foreach работает и с многомерными массивами. В этом случае он возвращает элементы в порядке следования строк: от первой до последней.

Индексаторы.

Индексатор предназначен для обращения к скрытому полю класса, представляющему собой массив, используя имя объекта и номер элемента массива в квадратных скобках.

Синтаксис индексатора:

```
<атрибуты><спецификаторы> <тип> this[<список_параметров>]
{
  get код_доступа
  set код_доступа
}
```

Спецификаторы аналогичны спецификаторам свойств и методов. Здесь тип – базовый тип индексатора. Он соответствует базовому типу массива. Код доступа представляет собой блоки операторов, которые выполняются при получении (get) или установке значения (set) элемента массива. Может отсутствовать либо часть get, либо set. Если отсутствует часть set, индексатор доступен только для чтения, если отсутствует часть get, индексатор доступен только для записи. Список параметров содержит одно или несколько описаний индексов, по которым выполняется доступ к элементу.

Язык C# допускает использование многомерных индексаторов. Они описываются аналогично обычным и применяются в основном для контроля за занесением данных в многомерные массивы и выборке данных из многомерных массивов, оформленных в виде классов.

Например, если внутри класса объявлен двумерный массив int[,] a;, то заголовок индексатора должен иметь вид:

```
public int this[int i, int j]
```

Задания для самостоятельного выполнения

Создайте проект, в котором опишите класс для решения задачи вашего варианта.

Класс должен содержать закрытое поле двумерного массива, конструктор без параметров, конструктор с параметрами, свойства, индексатор, методы (ввода, вывода, обработки массива). Обработку массива в соответствии с заданием варианта осуществлять в одном методе, исходные данные и результаты работы метода передавать параметрами. В программе должны проверятся все элементы разработанного класса (таблица 2.1).

Таблица 2.1 – Варианты заданий

Номер варианта	Тип массива	Размерность	Метод
1	Вещественный	N × N	Найти произведение элементов массива, расположенных между максимальным и минимальным элементами
2	Целочисленный	N × N	Найти произведение элементов массива с четными номерами
3	Целочисленный	N × N	Найти сумму элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами
4	Вещественный	N × N	Найти сумму положительных элементов
5	Целочисленный	N × N	Найти сумму элементов массива с нечетными номерами
6	Вещественный	N × N	Найти сумму элементов массива, расположенных между первым и вторым нулевым элементами
7	Вещественный	N × N	Найти произведение элементов массива, расположенных после максимального элемента

Окончание таблицы 2.1

Номер варианта	Тип массива	Размерность	Метод
8	Целочисленный	N × N	Найти произведение отрицательных элементов массива
9	Целочисленный	N × N	Найти сумму модулей элементов массива, расположенных после первого элемента, равного нулю
10	Вещественный	N × N	Найти сумму элементов массива, расположенных после первого положительного элемента

Контрольные вопросы

1 Какие члены класса содержат код?
2 Какие члены класса содержат данные?
3 В чём состоит назначение конструктора?
4 Чем конструктор отличается от обычного метода?
5 Перечислите типы конструкторов класса.
6 Сколько конструкторов может содержать класс?

3 Лабораторная работа № 3. Наследование классов

Цель работы **–** изучение приемов создания иерархии классов, выделение общих признаков объектов в базовый класс, организации доступа к элементам базового и производных классов.

Теоретические сведения

Наследование.

Класс в C# может иметь произвольное количество потомков и только одного предка. При описании класса имя его предка записывается в заголовке класса после двоеточия:

[атрибуты][спецификаторы] class <имя_класса> [:предки] <тело класса>

Если имя предка не указано, предком считается базовый класс всей иерархии System.Object.

Элементы базового класса, определенные как private, в производном классе недоступны. Поля, определенные со спецификатором protected, будут доступны методам всех классов, производных от базового. Этот модификатор остается со своим членом независимо от реализуемого количества уровней наследования.

Пример **–** Рассмотрим в качестве базового класс, в котором описывается печатная продукция (журналы, газеты и др.)

```
class Press{
  //Закрытые поля
  string name; //Название
  int copies;    //Тираж
  double price;//Цена

  //Конструктор с параметрами
  public Press(string name, int copies, double price){
    this.name = name;
    this.copies = copies;
    this.price = price;
  }

  //Свойства
  public string Name {get {return name;}}
  public int Copies {get {return copies;} set {copies = value;}}
  public double Price {get {return price;}}

  //Метод вычисления стоимости тиража
  public double Cost(Press pr){return pr.copies*pr.price;}
  //Метод вывода состояния объекта
  public void Output(Press pr){
    Console.WriteLine("Название: {0} \t Тираж: {1} \t Цена:
    {2}", pr.name, pr.copies, pr.price);
}}
```

Вызов конструкторов базового класса.

Конструкторы не наследуются, поэтому производный класс должен иметь собственные конструкторы. Порядок вызова конструкторов определяется приведенными далее правилами:

Если в конструкторе производного класса явный вызов конструктора базового класса отсутствует, автоматически вызывается конструктор базового класса без параметров.

Для иерархии, состоящей из нескольких уровней, конструкторы базовых классов вызываются, начиная с самого верхнего уровня. После этого выполняются конструкторы тех элементов класса, которые являются объектами, в порядке их объявления в классе, а затем исполняется конструктор класса. Таким образом, каждый конструктор инициализирует свою часть объекта.

Если конструктор базового класса требует указания параметров, он должен быть явным образом вызван в конструкторе производного класса в списке инициализации. Вызов выполняется с помощью ключевого слова base.

Формат расширенного объявления таков:

```
имя_производного_класса(список_параметров):base (список_аргументов)
{// тело конструктора}
```

Здесь с помощью элемента список_аргументов задаются аргументы, необходимые конструктору в базовом классе. При отсутствии ключевого слова base автоматически вызывается конструктор базового класса, действующий по умолчанию.

Ключевое слово base всегда отсылает к базовому классу, стоящему в иерархии классов непосредственно над вызывающим классом. В примере выше это класс Press и его конструктор.

Наследование и сокрытие имен.

Новым членам (полям, методам и свойствам) производного класса можно давать имена, совпадающие с именами членов базового класса. В этом случае перед членом производного класса необходимо поставить ключевое слово new. При этом, хотя соответствующие члены базового класса наследуются, они становятся скрытыми в производном классе.

Для обращения к скрытому члену применяется ссылка base, которая указывает на базовый класс производного класса. Формат ее записи такой:

base.<член базового класса>

Объекту базового класса можно присваивать объект производного класса, но вызываются для него только методы и свойства, определенные в базовом классе. Иными словами, возможность доступа к элементам класса определяется типом ссылки, а не типом объекта, на который она указывает. Это и понятно: ведь компилятор еще до выполнения программы определяет, какой метод вызывать, и вставляет в код фрагмент, передающий управление на этот метод. Этот процесс называется ранним связыванием.

Поскольку в производном классе определяется собственный метод с именем Cost, он скрывает одноименный метод, определенный в базовом классе. Следовательно, при вызове этого метода для объекта типа Magazine, вычисления выполняются так, как предусмотрено в этом классе, а не в классе Press. Ссылка base позволяет получить доступ к методу в базовом классе. Объект informatica типа Press создается с помощью конструктора класса Magazine. Но т. к. разрешение, какой метод вызвать, в данном случае осуществляется в момент компиляции (раннее связывание), то обращение происходит к методу базового класса.

Задания для самостоятельного выполнения.

Создайте проект, в котором опишите иерархию классов для решения задачи вашего варианта. Дополните базовый класс, который был создан в лабораторной работе № 1, классами-потомками. Производные классы должны содержать необходимые дополнения и изменения, указанные в варианте задания.

Составить тестирующую программу с выдачей результатов. В программе должна выполняться проверка разработанных элементов всей иерархии классов, вывод состояния различных объектов (таблица 3.1).

Таблица 3.1 – Варианты заданий

Номер варианта	Потомки	Дополнение и изменение в потомках
1	2	3
1, 13	Менеджер (поле – объем продаж, т)	Доход менеджера изменить в зависимости от объема продаж (если больше H, то увеличить на 1 % от H)
	Инженер (поле – количество разработанных проектов n)	Доход инженера увеличить на $4{,}8*n$
2, 14	Квартира в центре (поля – номер этажа, этажность дома)	Увеличить стоимость с учетом надбавки за расположение в центре на 1 %, и уменьшить на 1000 $ для квартир на первом и последнем этажах
	Квартира в пригороде (поле – расстояние от центра r)	Изменить стоимость в зависимости от расстояния: если меньше 10 км увеличить на 3 %, в противном случае уменьшить на $0{,}01*r$
3, 15	Фермерское (поле – количество внесенных удобрений m)	Найти урожай с учетом увеличения на $0{,}001*m$ на единицу площади
	Приусадебное (поле – сроки посева – ранний, средний, поздний)	Изменить урожай с учетом сроков (+10 % для раннего, –5 % для позднего) и площади посевов (менее 0,01 увеличить на 50 % от S)
4, 16	Письменный (поле – используемый материал, стоимость отделки)	Увеличить стоимость на стоимость отделки
	Обеденный (поле – форма)	Изменить стоимость в зависимости от формы: увеличить для прямоугольной формы на 10 %, овальной – на 20 % при $S < 0{,}5$ м^2 и $S > 2$ м^2
5, 17	Экспресс (поля – средняя скорость v, марка автобуса)	Изменить выручку с учетом увеличения на $0{,}05*v$ к цене билета
	Пригородный (поле – расстояние r)	Изменить выручку с учетом уменьшения на $0{,}01*r$ к цене билета
6, 18	Самолет (поля – высота h, км, скорость – v, км/ч)	Увеличить стоимость проезда на $100*h*v$
	Корабль (поля – количество палуб k, номер палубы n)	Увеличить стоимость проезда на палубах № 3–4 на k^2 %
7, 19	Инвалид (поле – номер группы)	Увеличить пособие для инвалидов 1-й группы на 30 %, 2-й – на 20 %
	Многодетные семьи (поле – количество детей)	Увеличить пособие от трех до пяти детей на 10 %, больше 5 – на 20 %
8, 20	Сотовый (поля – модель, год производства)	Изменить стоимость для телефонов, выпущенных: менее 1 года назад – на +20 %, более 3 лет – на 60 %
	Стационарный (поле – мобильность: переносной, непереносной)	Увеличить стоимость переносного телефона на 5,7, уменьшить стоимость непереносного телефона на 3,2
9	Грузовой (поле – грузоподъемность p, т)	Увеличить объем горючего на величину $M = p*R$

Окончание таблицы 3.1

1	2	3
	Легковой (поле – объем двигателя *V*, л)	Увеличить объем горючего для объема больше 3 на величину *M* = 0,005**V***R*
10	Пальто (поле – размер *V*)	Увеличить расход ткани: для пальто на (*V*/6,5 + 0,5)/10
	Костюм (поле – рост *H*)	для костюма на (2**H* + 0,3)/8
11	Офис (поле – количество этажей *N*)	Изменить высоту фундамента: для офиса с *N* > 10 на + 0,005*N
	Завод (поле -– вес *G*)	для завода на +0,000002**G*
12	Лекционная (поле – количество ярусов *K*)	Увеличить количество мест в лекционной аудитории на 2**K*
	Компьютерная (поле – количество компьютеров *P*)	Заменить количество мест в компьютерной аудитории на *P* – 1

Контрольные вопросы

1 Дайте определение терминам класс и объект. Как соотносятся эти понятия между собой?

2 Приведите синтаксис создания объекта в общем виде. Проиллюстрируйте его фрагментом программы.

3 Какие члены класса содержат код?

4 Какие члены класса содержат данные?

5 В чём состоит назначение конструктора?

4 Лабораторная работа № 4. Полиморфизм

Цель работы – освоить применение полиморфизма путем создания операций класса и их использования.

Перегрузка методов.

Использование нескольких методов с одним и тем же именем, но различными типами параметров и их способами передачи называется перегрузкой методов.

Компилятор определяет, какой именно метод требуется вызвать, по типу фактических параметров. Этот процесс называется разрешением перегрузки. Тип возвращаемого методом значения в разрешении не участвует.

Перегрузка методов является проявлением полиморфизма, одного из основных свойств ООП.

Перегрузка операций.

C# позволяет переопределить действие большинства операций так, чтобы при использовании с объектами конкретного класса они выполняли заданные

функции. Определение собственных операций класса называют перегрузкой операций.

При перегрузке операции ни одно из его исходных значений не теряется. Перегрузку операции можно расценивать как введение новой операции для класса.

Операции класса описываются с помощью методов специального вида (функций-операций).

Синтаксис операции:

<спецификаторы> <тип_возврата> operator
op(тип_параметра операнд1[,тип_параметра операнд2])
{<тело операции>}

В качестве спецификаторов одновременно используются ключевые слова public и static. Элемент op – это оператор (например " + " или " / "), который перегружается. Тело операции определяет действия, которые выполняются при использовании операции в выражении.

При описании операций необходимо соблюдать следующие правила:

– операция должна быть описана как открытый статический метод класса (спецификаторы public static);

– параметры в операцию должны передаваться по значению (т. е. не должны предваряться ключевыми словами refили out);

– сигнатуры всех операций класса должны различаться; типы, используемые в операции, должны иметь не меньшие права доступа, чем сама операция (т. е. должны быть доступны при использовании операции).

В C# существуют три вида операций класса: унарные, бинарные и операции преобразования типа.

Унарные операции.

Можно определять в классе следующие унарные операции:

+, - , ! , ~ , ++ , -- , true, false.

Примеры заголовков унарных операций:

public static int operator +(MyObject m)
public static MyObject operator --(MyObject m)
public static bool operator true(MyObject m)

Параметр, передаваемый в операцию, должен иметь тип класса, для которого она определяется.

Операция должна возвращать:

– для операций +, –, ! и ~ величину любого типа;

– для операций ++ и -- величину типа класса, для которого она опре-

деляется;

– для операций true и false величину типа bool.

Операции не должны изменять значение передаваемого им операнда. Операция, возвращающая величину типа класса, для которого она определяется, должна создать новый объект этого класса, выполнить с ним необходимые действия и передать его в качестве результата.

Бинарные операции.

Можно определять в классе следующие бинарные операции:

+ , -, * , / , |, &, | | , && , == , =! , > , < , >=, <=, %

Примеры заголовков бинарных операций:

publicstaticMyObjectoperator+ (MyObjectml, MyObject m2)
public static bool operator == (MyObject ml, MyObject m2)

Хотя бы один параметр, передаваемый в операцию, должен иметь тип класса, для которого она определяется. Операция может возвращать величину любого типа.

Операции == и !=, > и <, >= и <= определяются только парами и обычно возвращают логическое значение.

Пример – Создать класс с закрытыми полями *a* и *b*, строковой переменной, означающей операцию арифметического выражения и свойством C. Свойство – значение выражения над полями *a* и *b*. Типы полей *a* и *b* – *float*; операции арифметического выражения +=,--; равенство объектов – по *a* свойству *C* для --. Выполнить переопределение метода Equals() для сравнения объектов и метода ToString() для вывода состояния объекта.

```
class MyClass{
  public float A { set; get; }
  public float B { set; get; }
  public string Operation { set; get; }
  public MyClass(int a, int b, string operation){
    A = a;
    B = b;
    Operation = operation;
  }
  public float C{
    get{
      if (Operation == "+="){
        return A + B;
      }
      else if (Operation == "--")
```

```
        return A--;
      else
        return 0;
    }
  }
  public override bool Equals(object obj){
    if (this.A.ToString() == obj.ToString())
      return true;
    else
      return false;
  }
  public override string ToString(){
    return string.Format(" A = {0}  B = {1} Operation = {2}", A,
                         B, Operation);
  }
}
static void Main(string[] args){
  MyClass mc = new MyClass(10, 20, "--");
  Console.WriteLine(mc.C);
  bool b = mc.Equals(mc.C);
  Console.WriteLine(b);
  Console.WriteLine(mc.ToString());
  Console.ReadLine();
}
```

Индивидуальные задания

Создайте проект, в котором опишите класс для решения задачи Вашего варианта. Разрабатываемый класс должен содержать следующие элементы: скрытые поля, свойства, индексатор, конструктор, перегруженная операция. В программе должна выполняться проверка всех разработанных элементов класса.

Варианты заданий

1 Описать класс для работы с двумерными массивами вещественных чисел. Реализовать возможность выполнения для согласованных массивов комбинированной операции +=.

2 Описать класс для работы с двумерными массивами вещественных чисел. Реализовать возможность выполнения для согласованных массивов комбинированной операции -=.

3 Описать класс для работы с двумерным массивом целых чисел. Обеспечить сравнение массивов на равенство (перегрузку операции == для поэлементного сравнения).

4 Описать класс для работы с двумерным массивом целых чисел. Реализовать возможность выполнение операции умножения всех элементов массива на заданное число.

5 Описать класс – матрица размера 2 × 2. Элементы матрицы числа типа double. Перегрузить оператор + так, чтобы он выполнял операцию сложения матриц размера 2×2.

6 Описать класс – матрица размера 2 × 2. Элементы матрицы числа типа double. Перегрузить оператор – так, чтобы он выполнял операцию вычитания матриц размера 2×2.

7 Описать класс – матрица размера 2 × 2. Элементы матрицы числа типа double. Перегрузить оператор * так, чтобы он выполнял операцию умножения матриц размера 2×2 на действительное число типа double.

8 Описать класс – матрица размера 2 × 2. Элементы матрицы числа типа double. Перегрузить оператор > так, чтобы он выполнял операцию поэлементого сравнения элементов матриц A и B размера 2 × 2.

9 Описать класс – матрица размера 2 × 2. Элементы матрицы числа типа double. Перегрузить оператор < так, чтобы он выполнял операцию поэлементого сравнения элементов матриц A и B размера 2 × 2.

10 Описать класс для работы с двумерным массивом строк. Обеспечить перегрузку операции + для построчного соединения элементов.

Контрольные вопросы

1 Дайте определение понятию перегрузка.
2 Приведите синтаксис перегрузки арифметического оператора.
3 Какие члены класса содержат код?
4 Какие члены класса содержат данные?
5 В чём состоит назначение пергрузки?

5 Лабораторная работа № 5. Отношения между классами

Цель работы – приобретение навыков проектирования классов с использованием отношений агрегации и композиции.

Теоретические сведения

Отношение агрегации (агрегатное отношение) в C# означает, что один класс является частью другого класса, но при этом может существовать и независимо от него. Класс, который содержит другой класс, называется агрегатом, а класс, который содержится внутри другого класса, называется компонентом.

В C# для реализации отношения агрегации используются ссылки на объекты других классов. Например, класс Команда может содержать ссылки на объекты класса Игрок, которые могут существовать и без Команды. Это означает, что отношение агрегации является слабым.

Пример реализации отношения агрегации:

```
public class Team
{
   private List<Player> players = new List<Player>();

   public void AddPlayer(Player player)
   {
      players.Add(player);
   }
}
public class Player
{
   // Свойства игрока
}
```

В данном примере класс Team содержит ссылки на объекты класса Player в виде списка. При этом объекты класса Player могут существовать и независимо от класса Team.

Отношение композиции (компонентное отношение) в C# означает, что один класс является частью другого класса, и его существование зависит от существования этого другого класса. Класс, который содержит другой класс, называется контейнером, а класс, который содержится внутри другого класса, называется компонентом.

В C# для реализации отношения композиции используются объекты классов, созданные в конструкторе контейнера. Например, класс Автомобиль может содержать объект класса Двигатель, который не может существовать без Автомобиля. В случае отношения композиции объекты класса Двигатель уничтожаются вместе с объектами класса Автомобиль.

Пример реализации отношения композиции:

```
public class Car
{
   private Engine engine;

   public Car()
   {
      engine = new Engine();
   }

   // Методы автомобиля
}

public class Engine
{
```

```
    // Свойства двигателя
}
```

В данном примере класс Car содержит объект класса Engine, который создается в конструкторе класса Car. При этом объект класса Engine не может существовать без объекта класса Car.

Задания для самостоятельного выполнения

Написать программы решения задач с использованием классов, находящихся в отношении агрегации и композиции. Каждый из создаваемых классов должен иметь не менее трёх методов, свойств и конструкторов.

Вариант 1

1 Создать объект класса Завод, используя класс Склад деталей. Методы: изготовить деталь, доставить деталь на склад, получить деталь из склада, подсчитать количество деталей на складе, использовать деталь при сборке изделия, вывести на консоль наименование детали.

2 Создать объект класса Здание, используя класс Отопительная система. Методы: включить отопительную систему, выключить отопительную систему, определить температуру в здании, заполнить отопительную систему водой, слить воду из отопительной системы.

Вариант 2

1 Создать объект класса Больница, используя класс Приемное отделение. Методы: регистрация больного, выписка больного, установить диагноз болезни, назначить курс лечения, направить на обследование.

2 Создать объект Аэропорт, используя класс Взлётная полоса. Методы: взлёт самолёта, посадка самолёта, вывести на консоль длину взлетной полосы, количество взлетных полос.

Вариант 3

1 Создать объект класса Ресторан, используя класс Кухня. Методы: приготовить суп, салат, кофе, составить меню, принять заказ клиента, выполнить заказ, принести приготовленную еду посетителю ресторана.

2 Создать объект Библиотека, используя класс Книгохранилище. Методы: задать название книги, дополнить книгохранилище книгой, вывести на консоль количество книг, выдать книгу читателю.

Вариант 4

1 Создать объект класса Текст, используя класс Абзац. Методы: дополнить текст, вывести на консоль текст, заголовок текста.

2 Создать объект класса Наседка, используя классы Птица, Кукушка. Методы: летать, петь, нести яйца, высиживать птенцов.

Вариант 5

1 Создать объект класса Автомобиль, используя класс Колесо. Методы: ехать, заправляться, менять колесо, вывести на консоль марку автомобиля.

2 Создать объект класса Текстовый файл, используя класс Файл. Методы: создать, переименовать, вывести на консоль содержимое, дополнить, удалить.

Вариант 6

1 Создать объект класса Самолет, используя класс Крыло. Методы: летать, задавать маршрут, вывести на консоль маршрут.

2 Создать объект класса Одномерный массив, используя класс Массив. Методы: создать, вывести на консоль, выполнить операции (сложить, вычесть, перемножить).

Вариант 7

1 Создать объект класса Беларусь, используя класс Область. Методы: вывести на консоль столицу, количество областей, площадь, областные центры.

2 Создать объект класса Простая дробь, используя класс Число. Методы: вывод на экран, сложение, вычитание, умножение, деление.

Вариант 8

1 Создать объект класса Планета, используя класс Материк. Методы: вывести на консоль название материка, планеты, количество материков.

2 Создать объект класса Дом, используя классы Окно, Дверь. Методы: закрыть на ключ, вывести на консоль количество окон, дверей.

Вариант 9

1 Создать объект класса Звездная система, используя классы Планета, Звезда, Луна. Методы: вывести на консоль количество планет в звездной системе, название звезды, добавление планеты в систему.

2 Создать объект класса Роза, используя классы Лепесток, Бутон. Методы: расцвести, увянуть, вывести на консоль цвет бутона.

Вариант 10

1 Создать объект класса Компьютер, используя классы Винчестер, Дисковод, ОЗУ. Методы: включить, выключить, проверить на вирусы, вывести на консоль размер винчестера.

2 Создать объект класса Дерево, используя классы Лист. Методы: зацвести, опасть листьям, покрыться инеем, пожелтеть листьям.

Контрольные вопросы

1 Перечислите типы конструкторов класса.

2 Сколько конструкторов может содержать класс?

3 Объясните механизм вызова перегружаемого конструктора с помощью ключевого слова this.

4 Для чего используется наследование классов?

5 Опишите синтаксис производного класса. Какие модификаторы доступа применяются в иерархиях классов?

6 Лабораторная работа № 6. Обработка исключений

Цель работы – получение навыков использования механизма обработки исключительных ситуаций в прикладных программах.

Теоретические сведения

Исключительные ситуации.

Исключение представляет собой ошибку, происходящую во время выполнения программы. С помощью подсистемы обработки исключений можно обрабатывать эти ошибки, не вызывая аварийного завершения программы. Обработка исключений в C# выполняется с применением следующих ключевых слов: try, catch, throw и finally. Данные ключевые слова взаимосвязаны, т. к. использование одного из них влечет за собой использование других. При обработке исключений используются блоки try и catch.

Синтаксис:

```
try
{
//Блок кода_в котором контролируется появление ошибок.
}
catch (ExcepType1 ex)
{
//Обработчик исключений ExcepType1.
}
catch (ExcepType2 ex)
{
//Обработчик исключений ExcepType2.
}
```

Основные системные исключения приведены в таблице 6.1.

Тип исключения в обработчике catch должен соответствовать типу перехватываемого исключения. Не перехваченное исключение приводит к аварийному завершению программы. Для перехвата исключений вне зависимости от их типа (перехват всех исключений) используется оператор catch без параметров. Если исключение не произошло, то блоки catch для данного блока try при выполнении пропускаются.

Таблице 6.1 – Основные системные исключения

Исключение	Значение
FormatException	Некорректный формат операнда или аргумента (при передаче в метод)
DivideByZeroException	Предпринята попытка деления на ноль
IndexOutOfRangeException	Индекс массива выходит за пределы диапазона массива
InvalidCastException	Некорректное преобразование в процессе выполнения
OutOfMemoryException	Вызов new был неудачным из-за недостатка памяти
OverflowException	Переполнение при выполнении арифметической операции

Возврат из исключения.

После выполнения блока catch управление не передается обратно оператору программы, в котором возникло исключение. Выполнение программы продолжается с операторов, следующих после блока catch. Можно указать блок кода, который вызывается после выхода из обработчика catch. Для этого необходимо реализовать блок finally в конце последовательности блоков try/catch.

Общая форма блоков try/catch, включающей finally следующая:

```
try
{
// Блок_кода__контролирующий появление _ошибок .
}
catch (ExcepType1 ex)
{
// Обработка исключения ExcepType1.
}
catch (ExcepType2 ex)
{
// Обработка исключения ExcepType2.
}
finally
{
// Код блока finally.
}
```

Блок finally выполняется всегда независимо от того, возникло исключение или нет, и независимо от причины возникновения исключения.

Пример – Написать программу, в которой обрабатываются ошибка деления на ноль и ошибка некорректного ввода числа в консоль.

```
class Program{
  static int MyDel(int x, int y){
```

```
  return x / y;
 }

 static void Main(){
  try{
   Console.Write("Введите x: ");
   int x = int.Parse(Console.ReadLine());
   Console.Write("Введите y: ");
   int y = int.Parse(Console.ReadLine());
   int result = MyDel(x, y);
   Console.WriteLine("Результат: " + result);
  }
  // Обрабатываем исключение, возникающее при делении на ноль
  catch (DivideByZeroException ex){
   Console.WriteLine("Деление на 0 detected!!!\n");
   Console.WriteLine("ОШИБКА: " + ex.Message + "\n\n");
   Main();
  }
  //Обрабатываем исключение при некорректном вводе числа в
  //консоль
  catch (FormatException ex){
   Console.WriteLine("Это НЕ число!!!\n");
   Console.WriteLine("ОШИБКА: " + ex.Message + "\n\n");
   Main();
  }
  Console.ReadLine();
}}
```

Генерация исключений пользователем.

Как правило, система генерирует исключения автоматически при определенных условиях. Также исключение может быть создано пользователем и вызвано на выполнение с помощью оператора throw. Синтаксис оператора:

```
throw exceptOb;
```

где в качестве exceptOb должен быть обозначен объект класса исключений, производного от класса Exception.

После оператора throw указывается объект исключения, через конструктор которого передается сообщение об ошибке. Вместо типа Exception можно использовать объект любого другого типа исключений. Затем в блоке catch сгенерированное исключение обрабатывается.

Исключение, обработанное одним оператором catch, может быть сгенерировано повторно, и оно может перехватываться другим внешним оператором catch. Для этого используется оператор throw без типа исключения.

Пользовательские классы исключений.

Можно создавать свои собственные классы исключений, выполняющие обработку нестандартных ошибок в программе. Для этого необходимо объявить производный класс, который наследует класс Exception. Создаваемые пользователем классы исключений автоматически получают доступ к свойствам и методам, определенным в классе Exception.

Пример

```
//Класс, для описания пользовательского типа ошибок
class PersonException : Exception //Используем наследование
{
   //Принимает сообщение с описание ошибки, и код ошибки
   public PersonException(string aMessage, int aCode)
      : base(aMessage) //Вызываем конструктор базового класса
   {
      errorCode = aCode;
   }

   //Возвращает код ошибки
   public int ErrorCode { get { return errorCode; } }

   //Код ошибки
   private int errorCode;
}
```

Задания для самостоятельного выполнения

Во всех заданиях реализуемые функции или методы должны генерировать подходящие исключения. Обработку исключений нужно выполнять главной функцией, которая должна демонстрировать обработку всех перехватываемых исключений.

Функции, реализуемые в заданиях, обязаны выполнять проверку передаваемых параметров и генерировать исключение в случае ошибочных. Все функции реализуются в четырех вариантах:

1) без спецификации исключений;
2) со спецификацией throw();
3) с конкретной спецификацией с подходящим стандартным исключением;
4) спецификация с собственным реализованным исключением.

Собственное исключение должно быть реализовано в трех вариантах: как пустой класс; как независимый класс с полями-параметрами функции; как наследник от стандартного исключения с полями. Перехват и обработку исключений должна выполнять главная функция.

Варианты заданий

1 Функция вычисляет площадь треугольника по трем сторонам, используя формулу Герона.

2 Функция вычисляет корень линейного уравнения $ax + b = 0$.

3 Функция вычисляет периметр треугольника.

4 Функция переводит часы и минуты в секунды.

5 Функция вычисляет корень квадратного уравнения $ax + bx + c = 0$.

6 Функция вычисляет сумму бесконечной геометрической прогрессии.

7 Функция вычисляет целую часть неправильной дроби, представленной числителем и знаменателем – целыми числами.

8 Функция вычисляет разность между двумя датами в днях. Даты представлены структурой с тремя полями: год, месяц, день.

9 Функция вычисляет продолжительность телефонного разговора в минутах, принимая время начала и окончания. Время представлено классом с тремя полями: час, минута, секунда. Неполная минута считается за полную.

10 Функция вычисляет день недели по дате. Даты представлены структурой с тремя полями: год, месяц, день. Первое января считается понедельником.

Контрольные вопросы

1 Что такое исключение?
2 Какие виды ошибок существуют?
3 Какая конструкция обрабатывает возникающее исключение?
4 Какие способы обработки исключений вы знаете?
5 Перечислите типы классов исключений.

7 Лабораторная работа № 7. Обобщения

Цель работы – научиться разрабатывать обобщённые классы и применять их в программах.

Теоретические сведения

Обобщенный класс в C# – это класс, который может работать со множеством типов данных, а не только с одним. Для создания обобщенного класса в C# используется ключевое слово class, за которым следует имя класса, затем угловые скобки, в которых указываются параметры типа, и тело класса, в котором параметры типа могут использоваться в качестве типов данных для полей, свойств, методов и других элементов класса.

Например, обобщенный класс Stack<T> может работать со стеком объектов любого типа T:

```
public class Stack<T>
{
   private T[] elements;
   private int top = -1;
```

```
    public Stack(int size)
    {
        elements = new T[size];
    }

    public void Push(T item)
    {
        elements[++top] = item;
    }

    public T Pop()
    {
        return elements[top--];
    }
}
```

В данном примере класс Stack<T> имеет один параметр типа T, который используется в качестве типа элементов стека. Конструктор класса принимает размер стека, создает массив типа T и сохраняет его в поле класса elements. Методы Push и Pop добавляют и удаляют элементы из стека соответственно.

При использовании обобщенного класса Stack<T> параметр типа T может быть заменен на любой тип данных, например:

```
Stack<int> intStack = new Stack<int>(10);
intStack.Push(1);
intStack.Push(2);
intStack.Push(3);
int x = intStack.Pop(); // x = 3

Stack<string> stringStack = new Stack<string>(10);
stringStack.Push("Hello");
stringStack.Push("World");
string s = stringStack.Pop(); // s = "World"
```

В данном примере создаются два экземпляра класса Stack<T>, один для хранения целочисленных значений, другой – для хранения строковых значений. Параметр типа T заменяется на тип int и string соответственно. Методы Push и Pop работают с элементами соответствующего типа.

В C# обобщенные классы предоставляют более гибкое и безопасное решение, чем классы, которые работают только с определенным типом данных. Они обеспечивают возможность создания классов и методов, которые могут работать с различными типами данных, без необходимости создания отдельных классов или методов для каждого типа данных.

Кроме того, обобщенные классы позволяют избежать проблем, связанных с неявными приведениями типов, которые могут привести к ошибкам во время

выполнения программы. Вместо этого параметры типа задаются явно при создании экземпляра класса, что обеспечивает безопасность типов во время компиляции.

Обобщенные классы в C# могут иметь несколько параметров типа, например:

```
public class Dictionary<TKey, TValue>
{
   private List<KeyValuePair<TKey, TValue>> items = new List<KeyValue-
Pair<TKey, TValue>>();

   public void Add(TKey key, TValue value)
   {
      items.Add(new KeyValuePair<TKey, TValue>(key, value));
   }
}
```

Обобщенные классы в C# – это мощный инструмент, который позволяет создавать более универсальные и гибкие классы и методы. Они могут быть использованы для реализации широкого спектра алгоритмов и структур данных, и помогают сделать код более читабельным, безопасным и эффективным.

Задания для самостоятельного выполнения

Для заданной предметной области создать программу, состоящую из трехпяти классов. Каждый из создаваемых классов должен иметь не менее трёх методов, свойств, конструкторов. Предусмотреть использование типа данных – перечисление, коллекций List<T> (варианты заданий с нечётными номерами), Dictionary<TKey, TValue> (варианты заданий с чётными номерами). Ввод/вывод данных должен быть реализован вне классов.

Вариант 1

Предметная область: автоматическая телефонная станция (АТС). На АТС хранится информация о всех клиентах станции. АТС имеет список тарифов на междугородние разговоры. Клиент АТС может совершать множество звонков в различные города.

Система должна:

- позволять вводить информацию о тарифах;
- вводить информацию о клиентах и регистрировать звонки;
- по введенной фамилии о клиенте определять стоимость всех сделанных им звонков в соответствии с действующими тарифами;
- вычислять общую стоимость всех выполненных на АТС звонков.

Вариант 2

Предметная область: вокзал. Касса вокзала имеет список тарифов на различные направления. При покупке билета регистрируются паспортные данные пассажира. Пассажир покупает билеты на различные направления.

Система должна:

– позволять вводить данные о тарифах;
– позволять вводить паспортные данные пассажира и регистрировать покупку билета;
– рассчитывать стоимость купленных пассажиром билетов;
– после ввода наименования направления выводить список всех пассажиров, купивших на него билет.

Вариант 3

Предметная область: жилищно-эксплуатационная служба (ЖЭС). В ЖЭС хранятся тарифы на коммунальные услуги. ЖЭС имеет информацию о всех жильцах. При потреблении жильцами коммунальных услуг информация регистрируется в системе.

Система должна позволять выполнять следующие задачи:
– ввод тарифов;
– ввод информации о жильцах и потребленных услугах;
– после ввода фамилии выводить сумму всех потребленных услуг;
– выводить стоимость всех оказанных услуг.

Вариант 4

Предметная область: аэропорт. Касса аэропорта имеет список тарифов на различные направления. При покупке билета регистрируются паспортные данные.

Система должна:
– позволять вводить данные о тарифах;
– позволять вводить паспортные данные пассажира и регистрировать покупку билета;
– рассчитывать стоимость купленных пассажиром билетов;
– рассчитывать стоимость всех проданных билетов.

Вариант 5

Предметная область: банк. Информационная система банка хранит описание процентов по различным вкладам. Система хранит информацию о вкладчиках и сделанных ими вкладах. Каждый клиент может поместить в банк только один вклад.

Система должна позволять выполнять следующие задачи:
– хранить информацию о процентах по вкладам;
– хранить информацию о клиентах;
– пополнять клиенту величину вклада;
– вычислять общую сумму выплат по процентам для всех вкладов.

Вариант 6

Предметная область: отдел расчета зарплаты. Информационная система отдела расчета зарплаты на предприятии хранит данные о величине оплаты за различные виды работ. Система хранит информацию о работниках предприятия.

Система должна позволять выполнять следующие задачи:

– вводить информацию о различных видах работ;
– вводить информацию о работниках и выполненных ими работах;
– после ввода фамилии выводить для работника зарплату;
– выводить сумму выплат всем работникам.

Вариант 7

Предметная область: фирма грузоперевозок. Фирма имеет список тарифов по перевозке грузов. Клиент регистрируется в системе, после чего может заказать перевозку определенного объема груза.

Система должна позволять выполнять следующие задачи:

– ввод тарифов;
– регистрация клиента и заказ на перевозку грузов;
– вывод суммы заказа для определенного клиента;
– подсчет суммарной стоимости всех заказов.

Вариант 8

Предметная область: гостиница. Информационная система гостиницы хранит информацию о всех номерах и их стоимости. Система регистрирует клиентов. Каждый клиент может заказать один номер. При попытке заказа номера, который занят, выводится предупреждение.

Система должна позволять выполнять следующие задачи:

– ввод информации о номерах и их стоимости;
– регистрация клиента и заказ номера;
– вывод списка незанятых номеров;
– после ввода фамилии клиента вывод стоимости проживания.

Вариант 9

Предметная область: интернет-оператор. Провайдер имеет различные тарифы доступа в сети Интернет за 1 Мбайт в зависимости от величины абонентской платы. Информационная система провайдера хранит данные о клиентах.

Система должна позволять выполнять следующие задачи:

– ввод тарифов;
– регистрация пользователя;
– ввод данных о потребленном трафике для конкретного пользователя;
– подсчет общей стоимости реализованного трафика;
– поиск клиента, заплатившего наибольшую стоимость за услуги.

Вариант 10

Предметная область: интернет-магазин. В информационной системе хранятся данные о товарах. Клиент звонит в магазин и оставляет заказ на товар.

Система должна позволять выполнять следующие задачи:

– ввод информации о товарах;
– регистрация заказа клиента на покупку определенного товара;

– после ввода фамилии покупателя вывод списка заказанных им товаров;
– после ввода фамилии покупателя вывод суммы заказа.

Контрольные вопросы

1 Что такое сокрытие при наследовании классов?
2 Какова последовательность выполнения конструкторов при наследовании?
3 Каков механизм раннего связывания?
4 Что такое многоуровневая иерархия классов?
5 Назовите альтернативы наследованию классов.

8 Лабораторная работа № 8. Коллекции

Цель работы **–** научиться разрабатывать обобщённые классы, коллекции, итераторы и применять их в программах.

Теоретические сведения

Списки определены в пространстве имен System.Collections.Generic.

Как и массивы, списки поддерживают индексы, с помощью которых можно обратиться к определенным элементам.

C# позволяет осуществить перебор списка с помощью стандартного цикла foreach и других циклов.

Методы класса List:

– **void Add(T item)**: добавление нового элемента в список;
– **void AddRange(IEnumerable<T> collection)**: добавление в список коллекции или массива;
– **int BinarySearch(T item)**: бинарный поиск элемента в списке. Если элемент найден, то метод возвращает индекс этого элемента в коллекции. При этом список должен быть отсортирован;
– **void CopyTo(T[] array)**: копирует список в массив array;
– **void CopyTo(int index, T[] array, int arrayIndex, int count)**: копирует из списка, начиная с индекса index элементы, количество которых равно count, и вставляет их в массив array начиная с индекса arrayIndex;
– **bool Contains(T item)**: возвращает true, если элемент item есть в списке;
– **void Clear()**: удаляет из списка все элементы;
– **bool Exists(Predicate<T> match)**: возвращает true, если в списке есть элемент, который соответствует делегату match;
– **T? Find(Predicate<T> match)**: возвращает первый элемент, который соответствует делегату match. Если элемент не найден, возвращается null;
– **T? FindLast(Predicate<T> match)**: возвращает последний элемент, который соответствует делегату match. Если элемент не найден, возвращается null;

– **List<T> FindAll(Predicate<T> match)**: возвращает список элементов, которые соответствуют делегату match;

– **int IndexOf(T item)**: возвращает индекс первого вхождения элемента в списке;

– **int LastIndexOf(T item)**: возвращает индекс последнего вхождения элемента в списке;

– **List<T> GetRange(int index, int count)**: возвращает список элементов, количество которых равно count, начиная с индекса index;

– **void Insert(int index, T item)**: вставляет элемент item в список по индексу index. Если такого индекса в списке нет, то генерируется исключение;

– **void InsertRange(int index, collection)**: вставляет коллекцию элементов collection в текущий список начиная с индекса index. Если такого индекса в списке нет, то генерируется исключение;

– **bool Remove(T item)**: удаляет элемент item из списка, и если удаление прошло успешно, то возвращает true. Если в списке несколько одинаковых элементов, то удаляется только первый из них;

– **void RemoveAt(int index)**: удаление элемента по указанному индексу index. Если такого индекса в списке нет, то генерируется исключение;

– **void RemoveRange(int index, int count)**: параметр index задает индекс, с которого надо удалить элементы, а параметр count задает количество удаляемых элементов;

– **int RemoveAll((Predicate<T> match))**: удаляет все элементы, которые соответствуют делегату match. Возвращает количество удаленных элементов;

– **void Reverse()**: изменяет порядок элементов;

– **void Reverse(int index, int count)**: изменяет порядок на обратный для элементов, количество которых равно count, начиная с индекса index;

– **void Sort()**: сортировка списка;

– **void Sort(IComparer<T>? comparer)**: сортировка списка с помощью объекта comparer, который передается в качестве параметра.

Индивидуальные задания

Разработать новую версию программы решения задания лабораторной работы № 3. В новой версии программы сохранить все классы из предыдущей версии программы и определить новый класс Collection, содержащий следующее.

1 Закрытое поле типа List<T> – список (варианты заданий с чётными номерами) или Dictionary<TKey, TValue> – словарь (варианты заданий с нечётными номерами). В качестве параметра типа <T> для List<T> использовать базовый тип в созданной иерархии классов. В коллекции Dictionary<TKey, TValue> в качестве параметра типа <TKey> использовать наименование производного класса, а в качестве <TValue> – значение одного из полей производного класса.

2 Конструктор с параметрами и конструктор по умолчанию для инициализации поля, объявленного в п. 1.

3 Метод для добавления элементов в коллекцию.

4 Метод для удаления элементов из коллекции.

5 Метод для просмотра элементов коллекции.

6 Перегруженную версию виртуального метода ToString() для формирования строки с информацией об элементах коллекции.

7 Метод, выполняющий сортировку коллекции по заданному условию.

8 Итератор для поиска элементов коллекции, удовлетворяющих заданному условию.

9 Обобщённый метод с типом возврата List<T> (здесь <T> базовый тип в созданной иерархии классов), который возвращает из коллекции лишь те объекты, тип которых указан в качестве обобщенного параметра метода, т. е. в качестве обобщённого параметра метода указывается тип дочернего класса, а из коллекции возвращаются только объекты указанного типа.

В методе Main() создать экземпляр класса Collection и продемонстрировать всю функциональность этого класса.

Контрольные вопросы

1 Что такое обобщённые классы?
2 Что такое коллекции?
3 Что такое итераторы?
4 Перечислите основные методы списка.
5 Приведите пример обработки списка.

9 Лабораторная работа № 9. Потоки ввода-вывода. Работа с файлами

Цель работы **–** освоить способы работы с файловыми потоками через основные классы для работы с файлами.

Теоретические сведения

Файловый ввод-вывод.

Основными классами для работы с файлами и потоками в C# являются File, FileStream и StreamReader StreamWriter. Класс File предназначен для создания, открытия, удаления, изменения атрибутов файла.

Выполнять обмен с внешними устройствами можно на уровне:

– двоичного представления данных (BinaryReader, BinaryWriter);
– байтов (FileStream);
– текста, т. е. символов (StreamWriter, StreamReader).

Рассмотрим простейшие приёмы работы с файловыми потоками.

Класс FileStream – представляет байтовый поток, разработанный для выполнения операции ввода/вывода в файл.

Использование классов потоков в программе предполагает следующие операции:

– создание потока и связывание его с физическим файлом;

– обмен (ввод/вывод);

– закрытие файла.

Каждый класс файловых потоков содержит несколько вариантов конструкторов, с помощью которых можно создавать объекты этих классов различными способами и в различных режимах. Например, файлы можно открывать только для чтения, только для записи или для чтения и записи. Эти режимы доступа к файлу содержатся в перечислениях FileMode и FileAccess, определенном в пространстве имен System.I0.

Пример инициализации объекта FileStream:

FileStream myStream. = File.Open("C:\MyFile.rxt", FileMode.Open, File-Access.Read);

В данном случае режим открытия установлен как FileMode.Open, что означает открыть файл, если он существует; права доступа установлены FileAccess.Read, что означает возможность только читать файл. Функция Open возвращает объект типа FileStream, посредством которого в дальнейшем происходят чтение или запись в файл. Иногда удобнее работать с файлом с помощью класса StreamReader. StreamReader и StreamWriter связываются с потоком при помощи конструктора инициализации.

Для StreamWriter используйте один из следующих конструкторов:

StreamWriter(string filename) StreamWriter(string filename, bool appendFlag)

Здесь элемент filename означает имя открываемого файла, причем имя может включать полный путь к файлу. Во второй форме используется параметр appendFlag типа bool: если appendFlag равен значению true, выводимые данные добавляются в конец существующего файла. В противном случае заданный файл перезаписывается.

В обоих случаях, если файл не существует, он создается, а при возникновении ошибки ввода-вывода генерируется исключение типа IOException (также возможны и другие исключения).

Для StreamReader чаще всего используется следующий конструктор:

StreamReader (string filename);

Здесь элемент stream означает имя открытого потока. Этот конструктор генерирует исключение типа ArgumentNullException, если поток stream имеет null-значение, и исключение типа ArgumentException, если поток stream не открыт для ввода. После создания объект класса StreamReader автоматически преобразует байты в символы.

Можно указывать кодировку, в которой будут считываться/записываться данные при создании StreamReader/StreamWriter:

```
static void Main(string[] args){
  FileStream file1=new FileStream("d:\\test.txt",FileMode.Open);
  StreamReader reader = new StreamReader(file1, Encoding.Unicode);
  StreamWriter writer = new StreamWriter(file1, Encoding.UTF8);
}
```

Пример **–** Записать в текстовый файл таблицу из двух столбцов: в первом столбце цифры от 1 до 5 с шагом 1, во втором – квадраты этих чисел. Считать файл и найти сумму квадратов чисел.

```
class Program{
  static void Main(string[] args){
    const string testFile = @"dat.data";
    FileStream fs = File.Create(testFile);
    using (var bw = new StreamWriter(fs)){//Запись данных в файл
      bw.WriteLine("{0, 10}  {1, 10}", 1, 1);
      bw.WriteLine("{0, 10}  {1, 10}", 2, 4);
      bw.WriteLine("{0, 10}  {1, 10}", 3, 9);
      bw.WriteLine("{0, 10}  {1, 10}", 4, 16);
      bw.WriteLine("{0, 10}  {1, 10}", 5, 25);
    }
    Console.WriteLine();
    Console.WriteLine();
    string[] massivStr;
    double summa = 0;
    using (var sr = new StreamReader(testFile)){
      while (true){
        String str = sr.ReadLine();//Чтение одной строки из файла
        if (str == null)
          break;
        //Разделяем строку на элементы, используя разделитель
        //пробел. Лишние пробелы удаляем
        massivStr = str.Split(new char[] {' '},
                StringSplitOptions.RemoveEmptyEntries);
        Console.WriteLine("{0} {1}",
                    Convert.ToDouble(massivStr[0]),
                    Convert.ToDouble(massivStr[1]));
        summa += Convert.ToDouble(massivStr[1]);//Сумма чисел
}}}}
```

Задания для самостоятельного выполнения

1 Записать в текстовый файл результат расчета функции f(y). Результат должен быть записан в виде двух столбцов – аргумента и значения функции от данного аргумента. Начало и конец диапазона, имя файла, а также шаг расчета вводить с клавиатуры (таблица 9.1).

Таблица 9.1 – Варианты заданий

Вариант	Функция
1	f(y) = y*y
2	f(y) = y*y*y
3	f(y) = cos(y)
4	f(y) = sin(y)
5	f(y) = sin(y)*cos(y)
6	f(y) = sin(y)*y
7	f(y) = cos(y)*y
8	f(y) = sin(y)*cos(y)*y
9	f(y) = sin(y)*y*y
10	f(y) = cos(y)*y*y

2 Считать файл, вывести на экран среднее арифметическое.

Контрольные вопросы

1 Что такое файловый поток?
2 Какие параметры может иметь конструктор потока?
3 Какие классы позволяют работать с файлами?

10 Лабораторная работа № 10. Работа со строками

Цель работы – изучение строковых типов и использование строковых типов.

Теоретические сведения

Класс string.

Основным типом при работе со строками является тип string, задающий строки переменной длины. Объекты класса string объявляются как все прочие объекты простых типов – с явной или отложенной инициализацией, с явным или неявным вызовом конструктора класса. У класса string достаточно много конструкторов.

Они позволяют сконструировать строку из:

- символа, повторенного заданное число раз;
- массива символов char[];
- части массива символов.

Операции над строками.

Над строками определены следующие операции:

- присваивание (=);
- две операции проверки эквивалентности (= =) и (!=);
- конкатенация или сцепление строк (+);

– взятие индекса ([]).

Бинарная операция + сцепляет две строки, приписывая вторую строку к хвосту первой.

Взятие индекса позволяет рассматривать строку как массив и получать без труда каждый ее символ. Каждый символ строки имеет тип char, доступный только для чтения, но не для записи (таблица 10.1).

Таблица 10.1 – Статические методы и свойства класса string

Метод	Описание
Empty	Возвращается пустая строка. Свойство со статусом read only
Compare	Сравнение двух строк. Реализации метода позволяют сравнивать как строки, так и подстроки. При этом можно учитывать или не учитывать регистр, особенности национального форматирования дат, чисел и т. д.
CompareOrdinal	Сравнение двух строк. Метод перегружен. Реализации метода позволяют сравнивать как строки, так и подстроки. Сравниваются коды символов
Concat	Конкатенация строк, допускает сцепление произвольного числа строк
Copy	Создается копия строки
Format	Выполняет форматирование в соответствии с заданными спецификациями формата. Ниже приведено более полное описание метода

Методы Join и Split.

Методы *Join* и *Split* выполняют над строкой текста взаимно обратные преобразования. *Экземплярный метод Split* позволяет осуществить разбор текста на элементы. *Статический метод Join* выполняет обратную операцию, собирая строку из элементов.

Экземплярные методы класса String.

Рассмотрим наиболее характерные методы при работе со строками (таблица 10.3). Следует помнить, что *класс string* является *неизменяемым*. Поэтому Replace, Insert и другие методы представляют собой функции, возвращающие новую строку в качестве результата и не изменяющие строку, вызвавшую метод, таблица 10.2.

Класс StringBuilder.

Хотя класс System.String предоставляет широкую функциональность по работе со строками, все таки он имеет свои недостатки. Прежде всего объект String представляет собой неизменяемую строку. Когда мы выполняем какой-нибудь метод класса String, система создает новый объект в памяти с выделением ему достаточного места. Удаление первого символа – не самая затратная операция. Однако когда подобных операций множество, а объем текста, для которого надо выполнить данные операции, также не самый маленький, то издержки при потере производительности становятся более существенными.

Чтобы выйти из этой ситуации во фреймворк .NET был добавлен новый класс StringBuilder, который находится в пространстве имен System.Text. Этот класс представляет динамическую строку.

Таблица 10.2 – Динамические методы и свойства класса String

Метод	Описание
Insert	Вставляет подстроку в заданную позицию
Remove	Удаляет подстроку в заданной позиции
Replace	Заменяет подстроку в заданной позиции на новую подстроку
Substring	Выделяет подстроку в заданной позиции
IndexOf, IndexOfAny, LastIndexOf,LastIndexOfAny	Определяются индексы первого и последнего вхождения заданной подстроки или любого символа из заданного набора
StartsWith, EndsWith	Возвращается true или false, в зависимости от того, начинается или заканчивается строка заданной подстрокой
PadLeft, PadRight	Выполняет набивку нужным числом пробелов в начале и в конце строки
Trim, TrimStart, TrimEnd	Обратные операции к методам Pad. Удаляются пробелы в начале и в конце строки, или только с одного ее конца
ToCharArray	Преобразование строки в массив символов

Объекты этого класса всегда объявляются с явным вызовом конструктора класса (через операцию new) (таблица 10.3).

Таблица 10.3 – Методы класса StringBuilder

Название	Вид	Описание
Append	Экземплярный метод	Добавление данных в конец строки. Разные варианты метода позволяют добавлять в строку величины любых встроенных типов, массивы символов, строки и подстроки string
Insert	Экземплярный метод	Вставка подстроки в заданную позицию
Length	Изменяемое свойство	Возвращает длину строки. Присвоение ему значения 0 сбрасывает содержимое и очищает строку
Remove	Экземплярный метод	Удаление подстроки из заданной позиции
Replace	Экземплярный метод	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом
ToString	Экземплярный метод	Преобразование в строку типа string
Chars	Изменяемое свойство	Возвращает из массива или устанавливает в массиве символ с заданным индексом. Вместо него можно пользоваться квадратными скобками []
Equals	Экземплярный метод	Возвращает true, только если объекты имеют одну и ту же длину и состоят из одних и тех же символов
CopyTo	Экземплярный метод	Копирует подмножество символов строки в массив char

Задания для самостоятельного выполнения

Задание 1

1 Для заданной строки символов проверить, является ли она симметричной или нет. (Симметричной считается строка, которая одинаково читается слева направо и справа налево.)

2 Для заданной строки символов определить сумму всех входящих в неё цифр.

3 Для заданной строки определить все входящие в неё символы. Например: строка "abccbbbabba" состоит из символов "a","b" и "c".

4 Задана строка символов. Определить, какой символ встречается в этой строке подряд наибольшее число раз. В ответе указать символ, образующий самую длинную последовательность, длину последовательности.

5 Для заданной строки символов, состоящей из строчных букв и пробелов, определить слово наибольшей длины, которое начинается и заканчивается на одну и ту же букву.

6 Задана строка символов, содержащая два или более слов, разделенных пробелами. Написать программу, меняющую местами все четные и нечетные слова в строке.

7 Подсчитать, сколько раз в данной строке встречается некоторая буква, вводимая с клавиатуры.

8 Из строки удалить среднюю букву, если длина строки нечетная, если четная – удалить две средние буквы.

9 Заменить все вхождения в текст некоторой буквы на другую букву (их значения вводить с клавиатуры).

10 Заменить все вхождения подстроки Str1 на подстроку Str2 (подстроки вводятся с клавиатуры)

Задание 2

1 Дан символ C. Вывести его код (т. е. номер в кодовой таблице).

2 Дано целое число N ($32 \le N \le 126$). Вывести символ с кодом, равным N.

3 Дан символ C. Вывести два символа, первый из которых предшествует символу C в кодовой таблице, а второй следует за символом C.

4 Дано целое число N ($1 \le N \le 26$). Вывести N первых прописных (т. е. заглавных) букв латинского алфавита.

5 Дано целое число N ($1 \le N \le 26$). Вывести N последних строчных (т. е. маленьких) букв латинского алфавита в обратном порядке (начиная с буквы «z»).

6 Дан символ C, изображающий цифру или букву (латинскую или русскую). Если C изображает цифру, то вывести строку «digit», если латинскую букву – вывести строку «lat», если русскую – вывести строку «rus».

7 Дана непустая строка. Вывести коды ее первого и последнего символа.

8 Дано целое число $N > 0$ и символ C. Вывести строку длины N, которая состоит из символов C.

9 Дано четное число $N > 0$ и символы C1 и C2. Вывести строку длины N, которая состоит из чередующихся символов C1 и C2, начиная с C1.

10 Дана строка. Вывести строку, содержащую те же символы, но расположенные в обратном порядке.

Контрольные вопросы

1 Какие особенности у типа данных String?
2 Что такое StringBuilder?
3 Перечислите методы класса String.
4 Перечислите методы класса StringBuilder.
5 Как преобразовать строку в массив символьного типа?

11 Лабораторная работа № 11. Лямбда-выражения

Цель работы **–** научиться разрабатывать программы с использованием делегатов, событий и лямбда-выражений.

Теоретические сведения

В C# лямбда-выражения представляют собой краткую форму записи анонимных методов, которые могут быть переданы в качестве параметров в другие методы или использованы внутри других выражений. Они позволяют определять функции на лету, что упрощает написание кода и повышает его читабельность.

Синтаксис лямбда-выражений в C# выглядит следующим образом:

```
(parameters) => expression
```

где parameters – список параметров, разделенных запятой, которые принимает выражение;

expression – выражение, которое должно быть выполнено.

Например, следующее лямбда-выражение принимает два целочисленных аргумента и возвращает их сумму:

```
(int x, int y) => x + y
```

Лямбда-выражения могут быть использованы вместе с делегатами, например, чтобы определить обработчик события:

```
button.Click += (sender, e) => MessageBox.Show("Button clicked!");
```

Они также могут использоваться для выполнения операций на коллекциях, например, для фильтрации или сортировки:

```
var evenNumbers = numbers.Where(x => x  % 2 == 0);
var sortedNumbers = numbers.OrderBy(x => x);
```

Таким образом, лямбда-выражения в C# предоставляют удобный и гибкий способ определения анонимных методов, которые можно использовать для решения различных задач.

Задания для самостоятельного выполнения

Разработать программу с использованием делегатов. В программе следует:

1) определить делегат, принимающий несколько параметров различных типов и возвращающий значение некоторого типа;
2) написать не менее двух методов, соответствующих данному делегату;
3) написать метод, принимающий разработанный делегат, в качестве одного из входных параметров. Осуществить вызов метода, передавая ему в качестве параметра-делегата:
    а) один из методов, разработанных в п. 2;
    б) лямбда-выражение;
4) повторить п. 3, используя вместо разработанного делегата обобщенный делегат Func< > или Action< >, соответствующий сигнатуре разработанного делегата;
5) используя многоадресный делегат организовать цепочку вызовов методов, разработанных в п. 2;
6) создать блочное лямбда-выражение, соответствующее делегату, описанному в п. 1, и продемонстрировать его применение.

Вариант 1

Разработать программу «Охота на волков». В лесу размером N × N хаотически движутся с некоторой скоростью четыре волка и два охотника. При совпадении координат волка и охотника волк исчезает, и охотник оповещает другого охотника и лесничество о своей добыче. Если волк достигнет края леса, то он спасается. Процесс движения волков и охотников визуализировать на дисплее с помощью символов псевдографики.

Вариант 2

Разработать программу «Авиаразведка». Создать условную карту местности размером N × N . В ячейках карты расположить некоторое количество различных целей (например, танки, пушки, склады с горюче-смазочными материалами и др.). Из произвольной точки, находящейся на границе карты, стартует самолёт-разведчик. Достигнув противоположной границы карты, он меняет курс и летит в обратном направлении. Так продолжается до тех пор, пока не будет покрыта вся карта местности. Самолёт-разведчик фиксирует цели, чьи координаты совпадают с его координатами и по достижении границ карты сообщает в штаб информацию о типе и количестве обнаруженных целей. Процесс полёта самолета и расположение целей визуализировать с помощью символов псевдографики.

Вариант 3

Создать программу, в которой реализуется следующая ситуация. В лесу размером N × N потерялась девочка. Она хаотично ходит по лесу со скоростью V,

и при этом кричит «ау» в надежде, что кто-нибудь её услышит. По лесу также хаотично двигаются со скоростью 2V мама, папа, дедушка и бабушка девочки. Если кто-нибудь из них услышит девочку, а услышать её могут, если между девочкой и слушателем не более четырёх квадратов леса, то девочка будет спасена. Если она самостоятельно достигнет края леса, то также будет спасена. Процесс поиска визуализировать, используя символы псевдографики.

Вариант 4

Создать программу «Сапёр». В программе реализовать класс Bomb и класс TimerBomb.

Функциональность класса Bomb: конструктор, передаваемый параметр – время, через которое бомба взорвется; метод Code – принимает код обезвреживания бомбы, если принятый код совпал с кодом, сгенерированным бомбой, – бомба считается обезвреженной. При создании экземпляра бомбы генерируется случайный код в заданном диапазоне и задается время срабатывания бомбы (например, код от 1 до 10, время срабатывания – 5 с).

Функциональность класса TimerBomb – генерирует событие OnTick через заданный промежуток времени. Для генерации события OnTick можно использовать класс Timer из пространства имён System.Timers.

12 Лабораторная работа № 12. Основы многопоточного программирования

Цель работы – ознакомиться с организацией многопоточной обработки данных, получить основные навыки программирования с использованием потоков.

Теоретические сведения

Многопоточное программирование в C# позволяет выполнять несколько задач одновременно в рамках одного приложения. Это позволяет увеличить производительность и отзывчивость приложения, особенно при работе с большими объемами данных или при выполнении длительных операций.

Основы многопоточного программирования в C# включают в себя следующие концепции.

Потоки – это базовые единицы работы в многопоточном приложении. В C# потоки представлены классом Thread и позволяют выполнять код в отдельном потоке.

Синхронизация – это механизмы, которые позволяют управлять доступом к общим ресурсам из нескольких потоков, чтобы избежать ошибок и гонок данных. В C# для синхронизации используются мьютексы, семафоры, блокировки и другие механизмы.

Асинхронное программирование – это способ выполнения задач без блокировки главного потока приложения. В C# для асинхронного программирования

используется ключевое слово async и методы, возвращающие объекты Task или Task<T>.

Пример многопоточного приложения в C# может выглядеть следующим образом:

```
using System;
using System.Threading;

class Program
{
   static void Main(string[] args)
   {
      // Создаем новый поток и запускаем его на выполнение
      Thread thread = new Thread(new ThreadStart(DoWork));
      thread.Start();

      // Выполняем работу в главном потоке
      for (int i = 0; i < 10; i++)
      {
         Console.WriteLine("Main thread: " + i);
         Thread.Sleep(100);
      }
   }

   static void DoWork()
   {
      // Выполняем работу в отдельном потоке
      for (int i = 0; i < 10; i++)
      {
         Console.WriteLine("Worker thread: " + i);
         Thread.Sleep(100);
      }
   }
}
```

В этом примере мы создаем новый поток и запускаем на выполнение метод DoWork, который выполняет работу в отдельном потоке. В главном потоке мы также выполняем работу, но используем метод Thread.Sleep, чтобы освободить процессорный ресурс для работы второго потока. Конечно, на практике многопоточное программирование может быть гораздо более сложным, но эти основы помогут начать работу с многопоточностью в C#.

Задания для самостоятельного выполнения

Разработать консольное приложение согласно варианту. Требования к программе: реализовать возможность задавать приоритет каждого из порожденных потоков; использовать символы псевдографики для визуализации потоков на

экране дисплея.

1 Умножение матрицы на вектор. Обработку одной строки матрицы производить в порожденном потоке.

2 Поиск всех простых чисел (простым называется число, которое является своим наибольшим делителем) в указанном интервале чисел, разделенном на несколько диапазонов. Обработка каждого диапазона производится в порожденном потоке.

Классический алгоритм Евклида определения наибольшего общего делителя двух целых чисел (x, y) может применяться при следующих условиях:

– оба числа x и y неотрицательные;
– оба числа x и y отличны от нуля.

На каждом шаге алгоритма выполняются сравнения:

– если x == y, то ответ найден;
– если x < y, то y заменяется значением y – x;
– если x > y, то x заменяется значением x – y.

3 Винни-Пух и пчелы. Заданное количество пчел добывают мед равными порциями, задерживаясь в пути на случайное время. Винни-Пух потребляет мед порциями заданной величины за заданное время и столько же времени может прожить без питания. Работа каждой пчелы реализуется в порожденном потоке.

4 Шарики. Координаты заданного количества шариков изменяются на случайную величину по вертикали и горизонтали. При выпадении шарика за нижнюю границу допустимой области шарик исчезает. Изменение координат каждого шарика в отдельном потоке.

5 Противостояние нескольких команд. Каждая команда увеличивается на случайное количество бойцов и убивает случайное количество бойцов участника. Борьба каждой команды реализуется в отдельном потоке.

6 Контрольная сумма. Для нескольких файлов (разного размера) требуется вычислить контрольную сумму (сумму кодов всех символов файла). Обработка каждого файла выполняется в отдельном потоке.

7 Бег с препятствиями. Создается условная карта трассы в виде матрицы, ширина которой соответствует количеству бегунов, а высота фиксирована, которая содержит произвольное количество единиц (препятствий) в произвольных ячейках. Стартующие бегуны (потоки) перемещаются по трассе и при встрече с препятствием задерживаются на фиксированное время. По достижении финиша бегуны сообщают свой номер.

8 Создать два потока. Первый ищет числа Фибоначчи (каждое последующее число равно сумме двух предыдущих чисел), второй – простые числа. Результат работы каждого потока сохраняется в отдельный файл. После остановки потока программа производит анализ файлов, выводит их на экран, а также показывает количество найденных чисел Фибоначчи и простых чисел.

9 Создать два потока. Первый поток производит запись в файл случайных данных, второй производит чтение данных из этого файла и вывод их на экран.

10 Создать приложение, выполняющее вычисление значений функции y = 23*x^2 – 33, с шагом x = 0.01. Первый поток выполняет расчёт функции и

добавляет результаты расчёта в конец массива. Второй поток извлекает из массива значения x и y и выводит их на экран.

Контрольные вопросы

1 Что понимается в ООП под термином «полиморфизм»? Объясните основной принцип полиморфизма.

2 Когда осуществляется выбор версии виртуального метода?

3 Какое ключевое слово используется при реализации виртуального метода в производном классе?

Список литературы

1 **Павловская, Т. А.** C#. Программирование на языке высокого уровня: учебник для вузов / Т. А. Павловская. – Санкт-Петербург: Питер, 2019. – 432 с.

2 **Васильев**, **А. В.** Программирование на C# для начинающих. Основные сведения / А. В. Васильев. – Москва: Эксмо, 2018. – 592 с.

3 **Шилд, Г.** C# 4.0: полное руководство: пер. с англ. / Г. Шилд. – Москва: Вильямс, 2011. – 1056 с.

4 **Подбельский, В. В.** Язык Си# Базовый курс: учебное пособие / В. В. Подбельский. – Москва: Финансы и статистика, 2011. – 384 с.

5 **Троелсен, Э.** Язык программирования C# и платформа .NET 4 / Э. Троелсен. – Москва: Вильямс, 2007. – 1392 с.