Министерство образования, науки и молодежной политики
Краснодарского края
Государственное бюджетное профессиональное образовательное учреждение
Краснодарского края «Ейский полипрофильный колледж»

ЛАБОРАТОРНАЯ РАБОТА №3

по теме:

Перегрузка методов и операций

Выполнил:
студент ЕПК, группа
ФИО

Проверил:
преподаватель (*указать предмет*)
Фомин А. Т.

Ейск 2023

Постановка задачи

1. Разработать класс, инкапсулирующий дату (день, месяц, год). Перегрузить операции +(количество дней), -(количество дней), ==(объект даты), !=(объект даты). Реализовать методы вывода даты в разных форматах.
2. Разработать класс, инкапсулирующий время (часы, минуты, секунды). Перегрузить операции +(количество секунд), +(объект время), -(количество дней), -(объект время), ==(объект время), !=(объект время). Реализовать методы вывода времени в разных форматах.
3. Разработать класс, инкапсулирующий простые дроби. Перегрузить операции +, -, *, /, ==, !=. Реализовать метод упрощения дроби.
4. Разработать классы Point (точка на плоскости) и Points (множество точек на плоскости). Перегрузить операции +(точка) — добавляет точку в множество, -(точка) — убирает точку из множества, если она в нем присутствует, +(множество точек) — объединение множеств, -(множество точек) — вычитание множеств, == - для точки и для множества, != - для точки и для множества.
5. Разработать классы Point (точка в пространстве) и Points (множество точек в пространстве). Перегрузить операции +(точка) — добавляет точку в множество, -(точка) — убирает точку из множества, если она в нем присутствует, +(множество точек) — объединение множеств, -(множество точек) — вычитание множеств, == - для точки и для множества, != - для точки и для множества.
6. Разработать классы Point (точка на плоскости) и Points (множество точек на плоскости). Перегрузить операции +(точка), -(точка) – смещение множества на заданный вектор, *(скаляр) — для точки и для множества, == - для точки и для множества, != - для точки и для множества.
7. Разработать классы Point (точка в пространстве) и Points (множество точек в пространстве). Перегрузить операции +(точка), -(точка) – смещение множества на заданный вектор, *(скаляр) — для точки и для множества, == - для точки и для множества, != - для точки и для множества.
8. Разработать класс, инкапсулирующий матрицу произвольной размерности. Перегрузить операции +, *(скаляр), *(матрица), ==, !=.
9. Разработать класс строк на основе массива символов. Перегрузить операции +, ==, !=.

10. Разработать класс, инкапсулирующий двумерный вектор. Перегрузить операции +, -, *(скаляр), ==, !=.
11. Разработать класс, инкапсулирующий трехмерный вектор. Перегрузить операции +, -, *(скаляр), ==, !=.
12. Разработать класс, инкапсулирующий вектор произвольной размерности. Перегрузить операции +, *(скаляр), *(вектор), ==, !=.
13. Разработать класс, инкапсулирующий количество (вещественное число) с единицей измерения. Перегрузить операции +, -, ==, !=. Разработать статический класс для конвертации единиц измерения.
14. Разработать классы Student (ФИО, группы) и Students (список студентов). В Students перегрузить операции +(Student), -(Student). В Student перегрузить операции == и !=.
15. Разработать классы Book (автор, название, isbn) и Books (список книг). В Books перегрузить операции +(Book), -(Book). В Book перегрузить операции == и *!=*.

Содержание отчета

- Постановка задачи
- Описание классов, полей и методов (оформить в виде таблицы)
- Кодирование (классы должны быть выведены в отдельный файл). Необходимо представить исходный код файла с классами и файла Program.cs
- Демонстрация ввода и вывода данных в консоли
- Описание алгоритма
- Выводы. Рекомендации по усовершенствованию алгоритма программы

Теоретические сведения

Иногда возникает необходимость создать один и тот же метод, но с разным набором параметров. И в зависимости от имеющихся параметров применять определенную версию метода. Такая возможность еще называется перегрузкой методов (method overloading). В языке C# мы можем создавать в классе несколько методов с одним и тем же именем. Но при этом мы должны учитывать, что методы с одним и тем же именем должны иметь либо разное количество параметров, либо параметры разных типов. Например, пусть у нас есть следующий класс

```
Calculator:class Calculator
{
    public void Add(int a, int b)
    {
        int result = a + b;
        Console.WriteLine($"Result is {result}");
    }
    public void Add(int a, int b, int c)
    {
        int result = a + b + c;
        Console.WriteLine($"Result is {result}");
    }
    public int Add(int a, int b, int c, int d)
    {
        int result = a + b + c + d;
        Console.WriteLine($"Result is {result}");
        return result;
    }
    public void Add(double a, double b)
    {
        double result = a + b;
        Console.WriteLine($"Result is {result}");
    }
}
```

Здесь представлены четыре разных версии метода Add, то есть определены четыре перегрузки данного метода. Первые три версии метода отличаются по количеству параметров. Четвертая версия совпадает с первой по количеству

параметров, но отличается по их типу. При этом достаточно, чтобы хотя бы один параметр отличался по типу. Поэтому это тоже допустимая перегрузка метода Add.

Стоит отметить, что разные версии метода могут иметь разные возвращаемые значения, как в данном случае третья версия возвращает объект типа int. Однако простое изменение возвращаемого типа у метода еще не является основанием для создания новой версии метода. Опять же новая версия должна отличаться от других по количеству параметров или по их типу. После определения перегруженных версий мы можем использовать их в программе:

```
class Program
{
    static void Main(string[] args)
    {
        Calculator calc = new Calculator();
        calc.Add(1, 2); // 3
        calc.Add(1, 2, 3); // 6
        calc.Add(1, 2, 3, 4); // 10
        calc.Add(1.4, 2.5); // 3.9
        Console.ReadKey();
    }
}
```

```
Консольный вывод:
Result is 3
Result is 6
Result is 10
Result is 3.9
```

Наряду с методами мы можем также перегружать операции.
Например, пусть у нас есть следующий класс Counter:

```
class Counter
{
    public int Value {get; set;}
}
```

Данный класс представляет некоторый счетчик, значение которого хранится в свойстве Value. И допустим, у нас есть два объекта класса Counter — два счетчика, которые мы хотим сравнивать или складывать на основании их свойства Value, используя стандартные операции сравнения и сложения:

```
Counter c1 = new Counter {Value = 23};
Counter c2 = new Counter {Value = 45};
bool result = c1 > c2;
Counter c3 = c1 + c2;
```

Но на данный момент ни операция сравнения, ни операция сложения для объектов Counter не доступны. Эти операции могут использоваться для ряда примитивных типов. Например, по умолчанию мы можем складывать числовые значения, но как складывать объекты абстрактных типов (классов и структур) компилятор не знает. И для этого нам надо выполнить перегрузку нужных нам операций.

Перегрузка операций заключается в определении специального метода класса, для объектов которого мы хотим определить операции:

```
public static возвращаемый_тип operator операция(параметры)
{ }
```

Этот метод должен иметь модификаторы public static, так как перегружаемый операция будет использоваться для всех объектов данного класса. Далее идет название возвращаемого типа. Возвращаемый тип представляет тот тип, объекты которого мы хотим получить. К примеру, в результате сложения двух объектов Counter мы ожидаем получить новый объект Counter. А в результате сравнения двух объектов мы хотим получить объект типа bool, который указывает истинно ли условное выражение или ложно. Но в зависимости от задачи возвращаемые типы могут быть любыми.

Затем вместо названия метода идет ключевое слово operator и собственно сама операция. И далее в скобках перечисляются параметры. Бинарные операции принимают два параметра, унарные - один параметр. И в любом случае один из параметров должен представлять тот тип - класс или структуру, в котором определяется операция. Например, перегрузим ряд операций для класса Counter:

```
 1  class Counter
 2  {
 3      public int Value { get; set; }
 4
 5      public static Counter operator+(Counter c1, Counter c2)
 6      {
 7          return new Counter { Value = c1.Value + c2.Value };
 8      }
 9      public static bool operator>(Counter c1, Counter c2)
10      {
11          if (c1.Value > c2.Value)
```

```
12                  return true;
13              else
14                  return false;
15          }
16          public static bool operator<(Counter c1, Counter c2)
17          {
18              if (c1.Value < c2.Value)
19                  return true;
20              else
21                  return false;
22          }
23      }
```

Поскольку все перегруженные операции — бинарные, то есть проводятся над двумя объектами, то для каждой перегрузки предусмотрено по два параметра.

Так как в случае с операцией сложения мы хотим сложить два объекта класса Counter, то операция принимает два объекта этого класса. И так как мы хотим в результате сложения получить новый объект Counter, то данный класс также используется в качестве возвращаемого типа. Все действия этой операции сводятся к созданию, нового объекта, свойство Value которого объединяет значения свойства Value обоих параметров:

```
public static Counter operator+(Counter c1, Counter c2)
{
    return new Counter { Value = c1.Value + c2.Value };
}
```

Также переопределены две операции сравнения. Если мы переопределяем одну из этих операций сравнения, то мы также должны переопределить вторую из этих операций. Сами операции сравнения сравнивают значения свойств Value и, в зависимости от результата сравнения, возвращают либо true, либо false.

Теперь используем перегруженные операции в программе:

```
1   static void Main(string[] args)
2   {
3       Counter c1 = new Counter {Value = 23};
4       Counter c2 = new Counter {Value = 45};
5       bool result = c1 > c2;
6       Console.WriteLine(result);   //false
7       Counter c3 = c1 + c2;
8       Console.WriteLine(c3.Value); // 23 + 45 = 68
```

```
 9        Console.ReadKey();
10    }
```

Стоит отметить, что так как по сути определение операция представляет собой метод, то этот метод мы также можем перегрузить, то есть создать для него еще одну версию. Например, добавим в класс Counter еще одну операцию:

```
1   public static int operator+(Counter c1, int val)
2   {
3         return c1.Value + val;
4   }
```

Данный метод складывает значение свойства Value и некоторое число, возвращая их сумму. И также мы можем применить эту операцию:

```
1   Counter c1 = new Counter {Value = 23 };
2   int d = c1 + 27; // 50
3   Console.WriteLine(d);
```

Следует учитывать, что при перегрузке не должны изменяться те объекты, которые передаются в операция через параметры. Например, мы можем определить для класса Counter операцию инкремента:

```
1   public static Counter operator++(Counter c1)
2   {
3         c1.Value += 10;
4         return c1;
5   }
```

Поскольку операция унарная, она принимает только один параметр - объект того класса, в котором данная операция определена. Но это неправильное определение инкремента, так как операция не должна менять значения своих параметров. И более корректная перегрузка операции инкремента будет выглядеть так:

```
1   public static Counter operator++(Counter c1)
2   {
3         return new Counter { Value = c1.Value + 10 };
4   }
```

То есть, возвращается новый объект, который содержит в свойстве Value инкрементированное значение.

При этом нам не надо определять отдельно операции для префиксного и для постфиксного инкремента (а также декремента), так как одна реализация будет работать в обоих случаях. Например, используем операцию префиксного инкремента:

```
Counter counter = new Counter() {Value = 10};
Console.WriteLine($"{counter.Value}");      // 10
Console.WriteLine($"{(++counter).Value}"); // 20
Console.WriteLine($"{counter.Value}");      // 20
```

```
Консольный вывод:
10
20
20
```

Теперь используем постфиксный инкремент:

```
Counter counter = new Counter() {Value = 10 };
Console.WriteLine($"{counter.Value}");      // 10
Console.WriteLine($"{(counter++).Value}"); // 10
Console.WriteLine($"{counter.Value}");      // 20
```

```
Консольный вывод:
10
10
20
```

Также стоит отметить, что мы можем переопределить операции true и false. Например, определим их в классе Counter:

```
class Counter
{
    public int Value { get; set; }
    public static bool operator true(Counter c1)
    {
        return c1.Value != 0;
    }
    public static bool operator false(Counter c1)
    {
        return c1.Value == 0;
```

```
        }
        // остальное содержимое класса
    }
```

Эти операции перегружаются, когда мы хотим использовать объект типа в качестве условия. Например:

```
Counter counter = new Counter() {Value = 0 };
if (counter)
        Console.WriteLine(true);
else
        Console.WriteLine(false);
```

При перегрузке операций надо учитывать, что не все операции можно перегрузить. В частности, мы можем перегрузить следующие операции:

- унарные операции +, -, !, ~, ++, --
- бинарные операции +, -, *, /, %
- операции сравнения ==, !=, <, >, <=, >=
- логические операции &&, ||
- операции присваивания +=, -=, *=, /=, %=

И есть ряд операций, которые нельзя перегрузить, например, операцию равенства = или тернарную операцию ?:, а также ряд других.