1. Каково назначение перегрузки операторов?

Способ объявления новых операторов для типа

2. Как используется ключевое слово operator?

Определяет операторный метод, который, в свою очередь, определяет действие оператора относительно своего класса

3. Какие операции нельзя перегружать в C#?

[], ->, (), += -= \*= /= и тд.

4. Можно ли перегрузкой отменить очередность выполнения операции?

Да

5. Истинно ли следующее утверждение: операция >= может быть перегружена.

Да

6. Сколько аргументов требуется для определения перегруженной унарной операции?

1

7. Можно ли перегружать операцию []?

индексатор

8. Можно ли перегружать операцию ->?

нет

9. Приведите пример оператора приведения типа

*public static MyArr operator -(MyArr obj1, MyArr obj2)*

*{*

*MyArr arr = new MyArr();*

*arr.x = obj1.x - obj2.x;*

*arr.y = obj1.y - obj2.y;*

*arr.z = obj1.z - obj2.z;*

*return arr;*

*}*

10. Что такое метод расширения? Как и где его можно использовать?

Методы расширения (extension methods) позволяют добавлять новые методы в уже существующие типы без создания нового производного класса.

11. Пусть дан фрагмент кода определения оператора преобразования типа. Определить форму преобразования.

public static implicit operator Point2D(Point3D a)

{/\* код\*/;}

12. Выберите верное утверждение. Метод расширения может:

1) получать доступ к public членам расширяемого класса

2) получать доступ к protected членам расширяемого класса

3) получать доступ к internal членам расширяемого класса

4) быть объявлен в любом классе

5) быть без параметров

13. Выберите из списка неверное правило перегрузки операторов для C#.

1) префиксные операции ++ и – – перегружаются парами

2) операции сравнения перегружаются парами: == и != ; < и >;<= и >=

3) перегруженные операции обязаны возвращать значения

4) должны объявляться как protected

5) true и false можно перегружать

Owner.cs

using System;

namespace OOP\_Lab4

{

public class Owner

{

private readonly int id;

private readonly string fio;

public Owner(int id, string fio)

{

this.id = id;

this.fio = fio;

}

public void GetInfo()

{

Console.WriteLine($"Owner – ID: {id}, FIO: {fio}.");

}

}

}

MathOperations.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OOP\_Lab4

{

static class MathOperation

{

public static string Max(MyList list)

{

string str = list.GetItemByIndex(0);

foreach (string item in list.GetList())

{

if(str.Length < item.Length)

{

str = item;

}

}

return str;

}

public static string Min(MyList list)

{

string str = list.GetItemByIndex(0);

foreach (string item in list.GetList())

{

if (str.Length > item.Length)

{

str = item;

}

}

return str;

}

public static int CountOfWords(this string str)

{

int count = 0;

string[] words = str.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

count += words.Length;

return count;

}

public static bool IsContainsEmptyElement(this MyList list)

{

foreach (string item in list.GetList())

{

if (item == "")

{

return true;

}

}

return false;

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OOP\_Lab4

{

class Program

{

static void Main(string[] args)

{

MyList list = new MyList(1, "PAV");

list.AddItem("Ivan");

list.AddItem("25");

list.AddItem("Andrew");

list.AddItem("Natalia");

list.AddItem("83");

list.AddItem("Victor");

MyList list1 = new MyList(2, "WWW");

list.AddItem("Alexander");

list.AddItem("Uri");

list.AddItem("88");

list = list + list1;

list--;

bool cn = list1 == list;

if (list)

Console.WriteLine("List is empty");

string abc = "awdg fseol ajes";

Console.WriteLine($"Count of words: {abc.CountOfWords()}");

Console.WriteLine($"Last number int list: {list.LastNumber(list)}");

Console.WriteLine($"Element number 3: {list.GetItemByIndex(3)}");

Console.WriteLine($"Delete item number 2: {list.DeleteItemByIndex(2,ref list)}");

list.GetOwner().GetInfo();

}

}

}

MyList.cs

using System;

using System.Linq;

using System.Collections;

using System.Collections.Generic;

namespace OOP\_Lab4

{

public class MyList

{

// ПОЛЯ =======================================================

private readonly Owner owner;

private readonly Date date;

private List<string> collection;

// СВОЙСТВА ===================================================

public int Size { get; }

// КОНСТРУКТОРЫ ===============================================

public MyList(int ownerID, string ownerFIO)

{

this.owner = new Owner(ownerID, ownerFIO);

this.date = new Date();

this.collection = new List<string>();

}

// МЕТОДЫ

public string LastNumber (MyList list)

{

int i = 0;

int j = 100;

foreach(string item in list.collection)

{

foreach (char c in item)

{

if (c > '0' && c < '9')

j = i;

}

i++;

}

return list.GetItemByIndex(j);

}

public List<string> GetList()

{

return collection;

}

public Owner GetOwner()

{

return owner;

}

public void AddItem(string item)

{

collection.Add(item);

}

public int GetSize()

{

int size = 0;

foreach(string item in collection)

{

size++;

}

return size;

}

public string GetItemByIndex(int index)

{

if (index > this.GetSize() - 1)

throw new Exception("GetItemByIndex: OutOfRange");

int size = -1;

foreach (string item in collection)

{

size++;

if (size == index)

return item;

}

return "";

}

public string DeleteItemByIndex(int index, ref MyList list)

{

if (index > list.GetSize() - 1)

throw new Exception("GetItemByIndex: OutOfRange");

list.collection.RemoveAt(index);

return "";

}

public override bool Equals(object obj)

{

if(obj is MyList)

{

MyList tmp = (MyList)obj;

foreach (string item1 in this.collection)

{

foreach (string item2 in tmp.collection)

{

if (item1 != item2)

{

return false;

}

}

}

return true;

}

return false;

}

public override int GetHashCode()

{

int counter = 0;

foreach(string item in collection)

{

counter += item.Length;

}

return counter \* this.GetSize();

}

// ПЕРЕГРУЗКИ =================================================

public static MyList operator + (MyList list, string item)

{

list.collection.Add(item);

return list;

}

public static MyList operator + (MyList listOne, MyList listTwo)

{

foreach(string item in listTwo.collection)

{

listOne.AddItem(item);

}

return listOne;

}

public static MyList operator -- (MyList list)

{

list.collection.RemoveAt(list.GetSize() - (list.GetSize()-1));

return list;

}

public static bool operator != (MyList listOne, MyList listTwo)

{

foreach (string item1 in listOne.collection)

{

foreach (string item2 in listTwo.collection)

{

if(item1 != item2)

{

return true;

}

}

}

return false;

}

public static bool operator == (MyList listOne, MyList listTwo)

{

foreach (string item1 in listOne.collection)

{

foreach (string item2 in listTwo.collection)

{

if (item1 != item2)

{

return false;

}

}

}

return true;

}

public static bool operator true (MyList list)

{

if (list.GetSize() == 0)

return true;

else return false;

}

public static bool operator false (MyList list)

{

if (list.GetSize() != 0)

return false;

else return true;

}

public class Date

{

public readonly DateTime time;

public Date()

{

time = DateTime.Now;

}

}

}

}