Балтийский государственный технический университет  
«ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра И5 «Информационные системы и программная инженерия»

**Лабораторная работа №3**по дисциплине «Программирование на языке высокого уровня»  
по теме «Сравнение подходов к освобождению ресурсов в языках C++ и C#»

Выполнил:  
Студент Альков В. С.  
Группа И407Б  
Вариант 2  
  
Преподаватель:  
Кимсанбаев К. А.

Санкт-Петербург  
2021

# Общая постановка задачи

Работа состоит из двух частей - разобраться в предоставленном примере и дополнить программу из предыдущей работы.

## Первая часть

В первой части необходимо рассмотреть представленный ниже пример, построить диаграмму времени жизни объектов и ответить на следующие вопросы:

Какое максимальное поколение объектов в ходе выполнение программы было выявлено? Сколько их в C# всего?

Что будет если закомментировать строчку GC.Collect(0);? Изменится ли вывод программы, если да, то как и почему?

Что будет если закомментировать строчку GC.Collect(2);? Изменится ли вывод программы, если да, то как и почему?

Измените параметр метода GC.GetTotalMemory() с true на false? На что это влияет?

В методе MakeSomeGarbage() добавьте к объекту vt создание еще одного любого объекта, например, класса StringBuilder. Что изменилось в выводе программы?

## Вторая часть

Во второй части необходимо дополнить разработанную в ходе выполнения предыдущей работы программу следующим образом:

* Для каждого класса добавить деструктор и метод Dispose
* Предусмотреть в меню возможность удаления объектов из списка с их дальнейшем уничтожением
* Предусмотреть в меню вызов метода Dispose для объектов списка
* Добавить в меню вызов сборщика мусора
* Добавить в меню вывод поколения для всех объектов в списке

# Первая часть

## Листинг программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GarbageCollectorInCSharp

{

class GCProgram

{

private const long maxGarbage = 1000;

static void Main()

{

GCProgram myGCCol = new GCProgram();

Console.WriteLine("The highest generation is {0}", GC.MaxGeneration);

myGCCol.MakeSomeGarbage();

Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));

Console.WriteLine("Total Memory: {0}", GC.GetTotalMemory(false));

GC.Collect(0);

Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));

Console.WriteLine("Total Memory: {0}", GC.GetTotalMemory(false));

GC.Collect(2);

Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));

Console.WriteLine("Total Memory: {0}", GC.GetTotalMemory(false));

Console.Read();

}

void MakeSomeGarbage()

{

Version vt;

for (int i = 0; i < maxGarbage; i++)

{

vt = new Version();

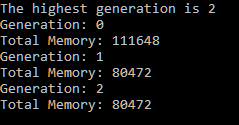
}

}

}

}

## Изначальный вывод программы



## Какое максимальное поколение объектов в ходе выполнение программы было выявлено? Сколько их в C# всего?

В ходе выполнения программы максимальное поколение объектов равнялось двум. В C# всего три поколения объектов (0, 1, 2).

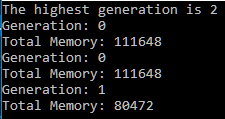
поколение 0–объекты, не подвергавшиеся сборке мусора;

поколение 1 – объекты, пережившие одну сборку мусора;

поколение 2 – объекты, пережившие более одной сборки мусора.

## Что будет если закомментировать строчку GC.Collect(0);? Изменится ли вывод программы, если да, то как и почему?

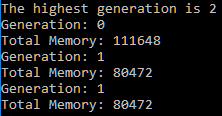
При комментировании строчки GC.Collect(0) изменится вывод программы.



Не будет произведена первая сборка мусора (поколения 0), следовательно не измениться поколение объекта MyGCCol и занимаемый объем памяти вплоть до следующего вызова сборки мусора.

## Что будет если закомментировать строчку GC.Collect(2);? Изменится ли вывод программы, если да, то как и почему?

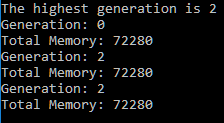
При комментировании строчки GC.Collect(2) изменится вывод программы.



Не будет произведена вторая сборка мусора (вплоть до поколения 2), следовательно поколение объекта MyGCCol останется равным 1, занимаемый объем памяти не измениться т.к. новых объектов не создавалось, а первая сборка мусора уже была произведена.

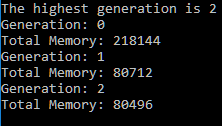
## Измените параметр метода GC.GetTotalMemory() с true на false? На что это влияет?

При смене этого параметров GetTotalMemory с false на true, вывод программы изменится следующим образом:



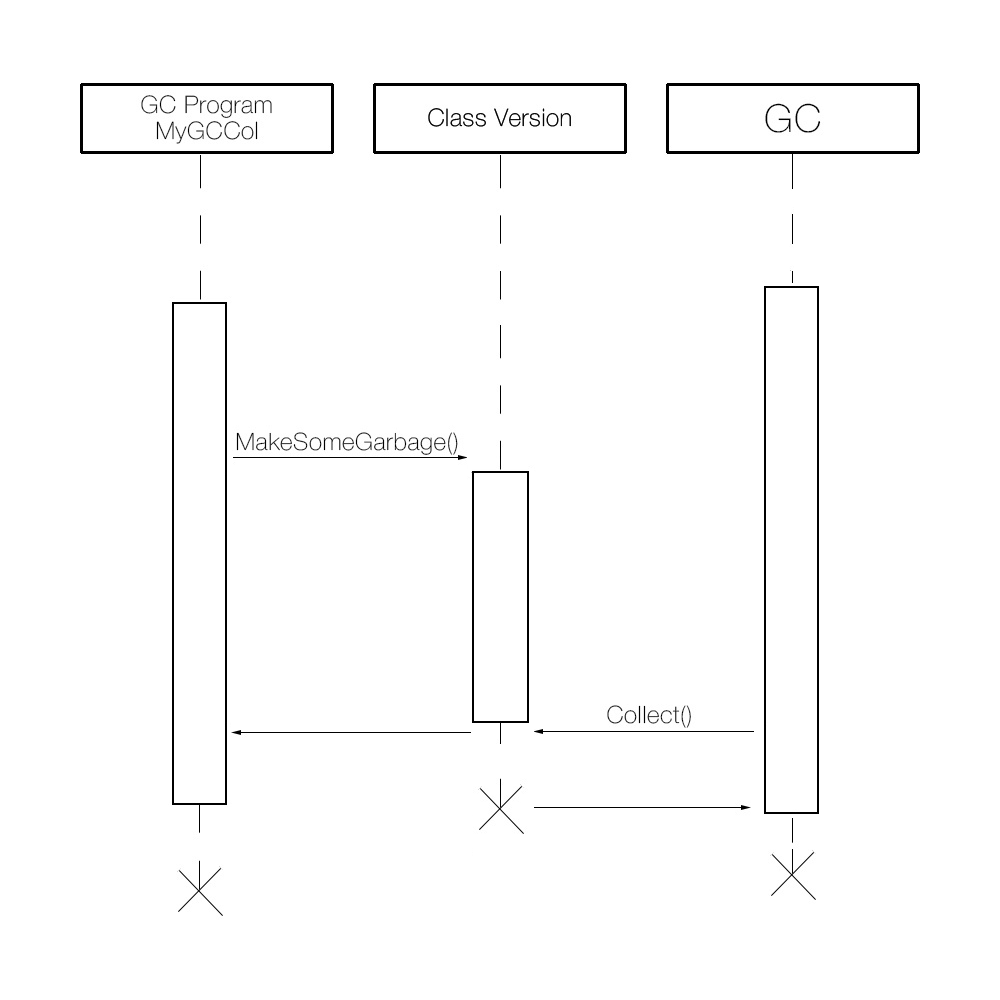
Функция GC.GetTotalMemory возвращает приблизительно занимаемый объем памяти в куче. Параметр forceFullCollection отвечает за ожидание перед возвратом значения в течении некоторого промежутка времени, пока система выполняет сборку мусора. Как я понимаю, результат получается одинаковым, так как метод перед возвращением дожидался, пока система освободит ресурсы, что происходит очень быстро.

## В методе MakeSomeGarbage() добавьте к объекту vt создание еще одного любого объекта, например, класса StringBuilder. Что изменилось в выводе программы?



Будет создано больше объектов, следовательно будет выделено больше памяти. Из-за этого изначальная выделенная память больше, чем раньше. В процессе сборки мусора большая часть объектов уничтожается, но, видимо, какая-то часть из них остается в куче.

## Диаграмма жизни объектов



# Вторая часть

## Вспомогательные функции и классы

### Функция поиска элемента в списке по имени.

Входные данные: список, имя.

Выходные данные: найденный элемент или null.

public static LinkedListNode<Sportsman>? FindNodeByName(LinkedList<Sportsman> list, string name)

{

var f = list.FirstOrDefault((o) => o.surname.ToLower().Equals(name.ToLower()));

return f !=null ? list.Find(f) : null;

}

### Функция поиска элемента в списке по индексу.

Входные данные: список, индекс.

Выходные данные: найденный элемент или null.

public static LinkedListNode<Sportsman>? FindNodeByIndex(LinkedList<Sportsman> list, int index)

{

var f = list.ElementAtOrDefault(index);

return f !=null ? list.Find(f) : null;

}

### Функция поиска спортсменок от 14 до 17 лет.

Входные данные: список.

Выходные данные: список спортсменок.

public static LinkedList<Sportsman> FindWomen(LinkedList<Sportsman> list) =>

new LinkedList<Sportsman>(list.Where(o => 2021 - o.year >= 14 && 2021 - o.year <= 17 && o.sex.ToLower()[0].Equals('ж')));

### Функция поиска самого высокого спортсмена, занимающегося плаванием среди мужчин в списке.

Входные данные: список.

Выходные данные: список с объектом или пустой список.

public static LinkedList<Sportsman> FindHighest(LinkedList<Sportsman> list)

{

var res = new LinkedList<Sportsman>();

Sportsman? entry = null;

int height = 0;

foreach(var i in list)

{

if(i.sport.ToLower().Equals("плавание")

&& i.sex.ToLower()[0].Equals('м') && i.height > height)

{

height = i.height;

entry = i;

}

}

if (entry != null)

res.AddLast((Sportsman)entry);

return res;

}

### Функция печати списка.

Входные данные: список.

Выходные данные: нет.

public static void PrintEntriesList(LinkedList<Sportsman> list)

{

int j = 0;

Console.WriteLine("{0,6}{1,15}{2,20}{3,15}{4,18}{5,11}"

,"Индекс", "Фамилия", "Вид спорта", "Пол", "Год рождения", "Рост");

foreach (var i in list)

Console.WriteLine(String.Format("{0,6}{1,15}{2,20}{3,15}{4,18}{5,11}",

++j, i.surname.CutString(10), i.sport.CutString(15), i.sex.CutString(10),

i.year.ToString().CutString(6), i.height.ToString().CutString(5)));

}

### Функция чтения данных из файла.

Входные данные: название файла.

Выходные данные: список прочитанных данных.

public static LinkedList<Sportsman> ReadFile(string filename)

{

using var file = new FileStream(filename, FileMode.OpenOrCreate);

using var reader = new StreamReader(file);

try

{

return JsonSerializer.Deserialize<LinkedList<Sportsman>>

(reader.ReadToEnd(), new JsonSerializerOptions{IncludeFields = true});

}

catch(JsonException)

{

Console.Write("Файл пуст, либо некоректен. Нажмите Enter, чтобы прододжить...");

Console.ReadLine();

return new LinkedList<Sportsman>();

}

}

### Функция записи списка в файл.

Входные данные: имя файла, список.

Выходные данные: нет.

public static void WriteFile(string filename, LinkedList<Sportsman> data)

{

using var file = new FileStream(filename, FileMode.Create);

using var writer = new StreamWriter(file);

var a = JsonSerializer.Serialize(data, new JsonSerializerOptions { IncludeFields = true });

writer.Write(a);

}

### Функция чтения строки с выводом заданного сообщения.

Входные данные: сообщение.

Выходные данные: прочитанная строка.

public static string ReadString(string message = "")

{

Console.Clear();

Console.Write(message);

return Console.ReadLine();

}

### Функция чтения int из диапазона, с выводом сообщения.

Входные данные: начало диапазона, конец диапазона, сообщение.

Выходные данные: число.

public static int ReadInt(int p1, int p2, string message = "")

{

int input;

bool success;

do

{

Console.Clear();

Console.Write(message);

success = int.TryParse(Console.ReadLine(), out input);

} while (!success || input < p1 || input > p2);

return input;

}

### Класс пункт меню.

[Flags]

public enum ItemType //тип пункта

{

None = 0,

Default = 1, //отображает детей для выбора

Move = 2, //пункт перехода в другой пункт

Exit = 4, //при переходе будет осуществлен выход

}

[Flags]

public enum ItemFlags

{

None = 0,

Action = 1, //сигнал о необходимости выполнить переданное действие

ClearScreen = 2, //флаг очистки экрана, после выполнения действий

Pause = 4, //флаг паузы, посредством чтения строки, после выполнения действий

}

class Item : IDisposable

{

ItemType type;

ItemFlags flags;

List<Item> items; //список подпунктов

Item? parent; //родитель пункта

Item? moveItem; //элемент для перехода

string? text; //название пункта

Action? action; //действие

public Item() : this(null, ItemType.Default) { }

public Item(string? text, ItemType type, ItemFlags flags = ItemFlags.None,

Action? action = null, Item? moveItem = null)

{

this.items = new List<Item>();

this.parent = null;

this.text = text;

this.type = type;

this.flags = flags;

this.moveItem = moveItem;

this.action = action;

}

public Item Add(Item newItem)

{

newItem.parent = this;

items.Add(newItem);

return newItem;

}

public Item? Update()

{

Item? tmp = this;

try

{

tmp.UpdateFlags(); //выполнение флагов

}

catch (ReturnToParentException) //обработка исключения

{

tmp = tmp.parent;

};

if (tmp != null) //обработка перехода в другие пункты

{

if (tmp.type.HasFlag(ItemType.Exit))

return tmp = null;

if (tmp.type.HasFlag(ItemType.Move))

tmp = tmp.moveItem;

if (tmp != null && tmp.type.HasFlag(ItemType.Default))

{

if (tmp.items.Count == 0)

tmp = parent;

else //выбор пользователя в меню

tmp=tmp.items[ReadInt(1,tmp.items.Count,tmp.ItemsTextToString())-1];

}

}

return tmp;

}

public void UpdateFlags()

{

if (flags.HasFlag(ItemFlags.ClearScreen))

Console.Clear();

if (flags.HasFlag(ItemFlags.Action) && action != null)

action();

if (flags.HasFlag(ItemFlags.Pause))

{

Console.Write("Нажмите Enter чтобы продожить..");

Console.ReadLine();

}

}

public string ItemsTextToString()

{

string print = "";

foreach (var i in this.items)

print += i + "\n";

return print;

}

public override string ToString() => text ?? ""; //если text == null, то text = ""

protected bool disposed = false;

// реализация интерфейса IDisposable.

public void Dispose()

{

Console.WriteLine("Dispose()");

Dispose(true);

GC.SuppressFinalize(this);

}

protected virtual void Dispose(bool disposing)

{

if (!disposed)

{

if (disposing)

{

// Освобождаем управляемые ресурсы

moveItem = parent = moveItem = null;

text = null;

action = null;

foreach (var i in items)

i.Dispose();

items.Clear();

}

// освобождаем неуправляемые объекты

disposed = true;

}

}

~Item()

{

Console.WriteLine("~Item()");

Dispose(false);

}

}

### Класс меню.

class Menu : Item , IDisposable

{

Item? current;

public Menu() { }

public void Run() //запуск меню

{

current = this;

while (current != null)

current = current.Update();

}

public void SetCurrent(Item current) => this.current = current;

protected override void Dispose(bool disposing)

{

Console.WriteLine("Dispose()");

current = null;

base.Dispose(true);

GC.SuppressFinalize(this);

}

~Menu()

{

Console.WriteLine("~Menu()");

Dispose(false);

}

}

### Класс исключения по переходу в родителя.

public class ReturnToParentException : Exception {}

### Класс меню помощник в работе со строками

public static class StringHelper

{

//функция обрезания строки до переданного количества, учитывая длину строки

public static string CutString(this string str, int lenght)

{

return str.Length > lenght ? str.Substring(0, lenght - 3) + "..." : str;

}

}

### Класс спортсмен

public class Sportsman : IDisposable

{

[JsonInclude]

public string surname, sport, sex;

[JsonInclude]

public int year, height;

public Sportsman(string surname, string sport, string sex, int year, int height)

{

this.surname = surname;

this.sport = sport;

this.sex = sex;

this.year = year;

this.height = height;

}

public static Sportsman CreateInstanseFromConsole()

{

return new Sportsman(ReadString("Фамилия: "), ReadString("Вид Спорта: "),

ReadString("Пол: "), ReadInt(1900, 2021, "Год рождения: "), ReadInt(0, 1000, "Рост: "));

}

public override string ToString()

{

return String.Format("{0}\n{1}\n{2}\n{3}\n{4}", surname, sport, sex, year, height);

}

private bool disposed = false;

// реализация интерфейса IDisposable.

public void Dispose()

{

Console.WriteLine("Dispose()");

Dispose(true);

GC.SuppressFinalize(this);

}

protected virtual void Dispose(bool disposing)

{

if (!disposed)

{

if (disposing)

{

// Освобождаем управляемые ресурсы

}

// освобождаем неуправляемые объекты

disposed = true;

}

}

~Sportsman()

{

Console.WriteLine("~Sportsman()");

Dispose(false);

}

}

## Главная функция.

public static void Main()

{

LinkedListNode<Sportsman> objectNode = null;

Sportsman objectValue = null;

Console.Write("Введите имя файла: ");

string filename = Console.ReadLine();

LinkedList<Sportsman> list = ReadFile(filename);

var menu = new Menu();

var item1 = menu.Add(new Item("1. Добавить запись", ItemType.Default, ItemFlags.ClearScreen));

item1.Add(new Item("1. В начало", ItemType.Move, ItemFlags.Action | ItemFlags.ClearScreen,

delegate () { list.AddFirst(Sportsman.CreateInstanseFromConsole()); }, menu));

item1.Add(new Item("2. В конец", ItemType.Move, ItemFlags.Action | ItemFlags.ClearScreen,

delegate () { list.AddLast(Sportsman.CreateInstanseFromConsole()); }, menu));

item1.Add(new Item("3. В произвольное место", ItemType.Move,

ItemFlags.Action | ItemFlags.ClearScreen | ItemFlags.Pause, delegate ()

{

if (list.Count != 0)

{

Console.Write("Введите фамилию записи, после которой вставить: ");

var node = FindNodeByName(list, Console.ReadLine());

if (node != null)

{

list.AddAfter(node, Sportsman.CreateInstanseFromConsole());

Console.WriteLine("Запись добавлена");

}

else

{

Console.WriteLine("Запись не найдена");

}

}

else

Console.WriteLine("Коллекция пуста");

}, menu));

menu.Add(new Item("2. Вывести записи", ItemType.Default,

ItemFlags.Action | ItemFlags.ClearScreen | ItemFlags.Pause,

delegate () { PrintEntriesList(list); }));

menu.Add(new Item("3. Найти самого высокого спортсмена, занимающегося плаванием, среди мужчин",

ItemType.Default, ItemFlags.Action | ItemFlags.ClearScreen | ItemFlags.Pause,

delegate () { PrintEntriesList(FindHighest(list)); }));

menu.Add(new Item("4. Вывести сведения о спортсменках, выступающих в юниорском разряде (14 - 17лет)",

ItemType.Default, ItemFlags.Action | ItemFlags.ClearScreen | ItemFlags.Pause,

delegate () { PrintEntriesList(FindWomen(list)); }));

menu.Add(new Item("5. Отсортировать по имени", ItemType.Default,

ItemFlags.Action | ItemFlags.ClearScreen, delegate () {

list = new LinkedList<Sportsman>(list.OrderBy(a => a.surname)); }));

menu.Add(new Item("6. Удалить запись по фамилии", ItemType.Default,

ItemFlags.Action | ItemFlags.ClearScreen | ItemFlags.Pause, delegate () {

Console.Write("Введите фамилию: ");

var entry = FindNodeByName(list, Console.ReadLine());

if (entry != null)

{

Console.WriteLine("Запись удалена");

list.Remove(entry);

entry.ValueRef.Dispose();

}

else

Console.WriteLine("Запись не найдена");

}));

menu.Add(new Item("7. Удалить запись по индексу", ItemType.Default,

ItemFlags.Action | ItemFlags.ClearScreen | ItemFlags.Pause, delegate () {

if (list.Count > 0)

{

var entry = FindNodeByIndex(list,

ReadInt(1, list.Count, $"Введите индекс элемента ( от 1 до {list.Count} ): ") - 1);

Console.WriteLine("Запись удалена");

list.Remove(entry);

entry.ValueRef.Dispose();

}

else

Console.WriteLine("Список пуст");

}));

var item2 = menu.Add(new Item("8. Изменить запись", ItemType.Default,

ItemFlags.Action | ItemFlags.ClearScreen,

delegate ()

{

if (objectNode == null)

{

Console.Write("Введите имя записи: ");

objectNode = FindNodeByName(list, Console.ReadLine());

if (objectNode == null)

{

Console.WriteLine("Запись не найдена. Нажмите Enter, чтобы продолжить...");

Console.ReadLine();

throw new ReturnToParentException();

}

objectValue = objectNode.Value;

}

}));

item2.Add(new Item("1. Фамилия", ItemType.Default, ItemFlags.Action | ItemFlags.ClearScreen,

delegate () { objectValue.surname = ReadString("Фамилия: "); }, menu));

item2.Add(new Item("2. Спорт", ItemType.Default, ItemFlags.Action | ItemFlags.ClearScreen,

delegate () { objectValue.sport = ReadString("Вид спорта: "); }, menu));

item2.Add(new Item("3. Пол", ItemType.Default, ItemFlags.Action | ItemFlags.ClearScreen,

delegate () { objectValue.sex = ReadString("Пол: "); }, menu));

item2.Add(new Item("4. Год рождения", ItemType.Default, ItemFlags.Action | ItemFlags.ClearScreen,

delegate () { objectValue.year = ReadInt(1900, 2021, "Год рождения: "); }, menu));

item2.Add(new Item("5. Рост", ItemType.Default, ItemFlags.Action | ItemFlags.ClearScreen,

delegate () { objectValue.height = ReadInt(0, 300, "Рост: "); }, menu));

item2.Add(new Item("6. Назад", ItemType.Move, ItemFlags.Action | ItemFlags.ClearScreen,

delegate () { objectNode.Value = objectValue; objectNode = null; }, menu));

menu.Add(new Item("9. Вызвать Dispose() по индексу", ItemType.Default,

ItemFlags.Action | ItemFlags.ClearScreen | ItemFlags.Pause, delegate () {

if (list.Count > 0)

{

var entry = FindNodeByIndex(list,

ReadInt(1, list.Count, $"Введите индекс элемента ( от 1 до {list.Count} ): ") - 1);

entry.ValueRef.Dispose();

}

else

Console.WriteLine("Список пуст");

}));

menu.Add(new Item("10. Вызвать Dispose() по фамилии", ItemType.Default,

ItemFlags.Action | ItemFlags.ClearScreen | ItemFlags.Pause, delegate () {

Console.Write("Введите фамилию: ");

var entry = FindNodeByName(list, Console.ReadLine());

if (entry != null)

{

entry.ValueRef.Dispose();

}

else

Console.WriteLine("Запись не найдена");

}));

menu.Add(new Item("11. Вызвать сборщик мусора", ItemType.Default,

ItemFlags.Action | ItemFlags.ClearScreen | ItemFlags.Pause, delegate (){ GC.Collect(); }));

menu.Add(new Item("12. Вывести поколения объектов", ItemType.Default,

ItemFlags.Action | ItemFlags.ClearScreen | ItemFlags.Pause, delegate () {

Console.WriteLine($"Поколение списка = {GC.GetGeneration(list)}");

var i = 1;

foreach (var a in list)

{

Console.WriteLine($"Поколение объекта {i++} = {GC.GetGeneration(a)}");

}

}));

menu.Add(new Item("13. Очистить список", ItemType.Default,

ItemFlags.Action | ItemFlags.ClearScreen | ItemFlags.Pause, delegate () { list.Clear(); }));

menu.Add(new Item("14. Сохранить и выйти", ItemType.Exit, ItemFlags.Action,

delegate () { WriteFile(filename, list); }));

menu.Add(new Item("15. Выйти", ItemType.Exit, ItemFlags.None));

menu.Run();

list.Clear();

GC.Collect();

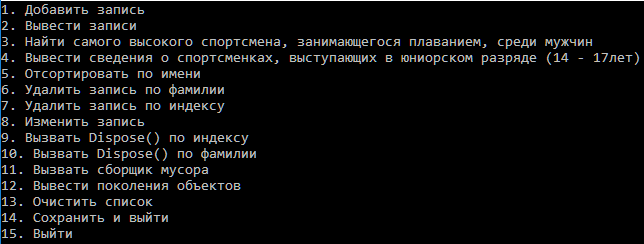
Console.WriteLine("Нажмите Enter чтобы продожить..");

Console.ReadLine();

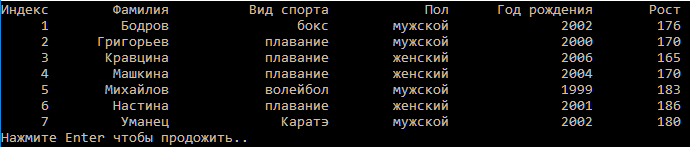
}

## Результат работы программы





### 2. Вывести записи



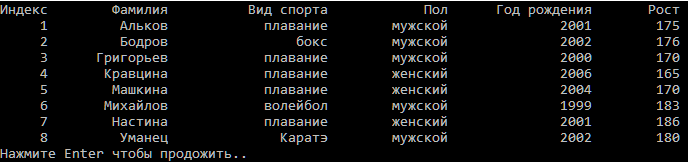
### 1. Добавить запись



### 1. В начало

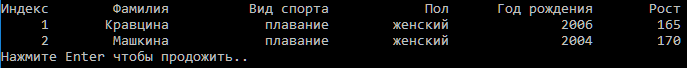
### 2. Вывести записи



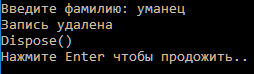
### 3. Найти самого высокого спортсмена, занимающегося плаванием, среди мужчин



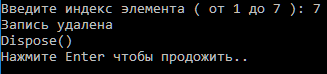
### 4. Вывести сведения о спортсменках, выступающих в юниорском разряде (14 - 17лет)



### 6. Удалить запись по фамилии

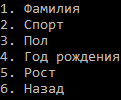


### 7. Удалить запись по индексу

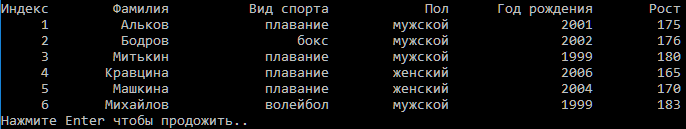


### 8. Изменить запись





### 2. Вывести записи



### 3. Найти самого высокого спортсмена, занимающегося плаванием, среди мужчин



### 9. Вызвать Dispose() по индексу



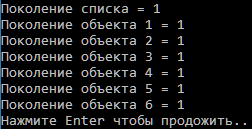
**10. Вызвать Dispose() по фамилии**



**11. Вызвать сборщик мусора**



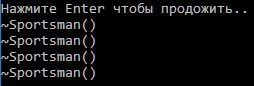
**12. Вывести поколения объектов**



**13. Очистить список**



**11. Вызвать сборщик мусора**



**12. Вывести поколения объектов**



# Выводы

В этой практической работе было доработано:

-Добавлена реализация IDisposable классам Спортсмен, пункт меню, меню.

-Пункты меню позволяющие вызывать Dispose для объекта.

Также мы изучили способы и подходы к очистке памяти на языке программирования C#.

Узнали про класс System.GC, который предоставляет возможности сборщика мусора, про его методы и, в частности, GC.Collect(), который позволяет вызвать сборку мусора в процессе работы программы, не ожидая автоматического вызова его системой.

Использовали интерфейс IDisposable, который реализует метод Dispose(), когда необходимо немедленно освободить все связанные с объектом ресурсы.