МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Брянский государственный технический университет

Утверждаю
Ректор университета
_____О.Н. Федонин
«_____»_____2013 г.

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

РАБОТА С КОЛЛЕКЦИЯМИ

Методические указания
к выполнению лабораторной работы
для студентов очной формы обучения
специальностей 090303 – «Информационная безопасность автоматизированных систем»,
090900 – «Информационная безопасность»

Брянск 2013

УДК 004.43

Языки программирования. Работа с коллекциями: методические указания к выполнению лабораторной работы для студентов очной формы обучения специальностей 090303 – «Информационная безопасность автоматизированных систем», 090900 – «Информационная безопасность». – Брянск: БГТУ, 2013. – 19 с.

Разработали:
Ю.А. Леонов, к.т.н., доц.,
Е.А. Леонов, к.т.н., доц.

Научный редактор: Ю.М. Казаков
Редактор издательства: Л.И. Афонина
Компьютерный набор: Ю.А. Леонов

Рекомендовано кафедрой «Компьютерные технологии и системы» БГТУ (протокол № 2 от 24.10.12)

Темплан 2013 г., п.

Подписано в печать Формат 60x84 1/16. Бумага офсетная.
Офсетная печать.
Усл. печ. л. 1,1 Уч. – изд. л. 1,1 Тираж 20 экз. Заказ Бесплатно

Издательство брянского государственного технического университета,
241035, Брянск, бульвар 50-летия Октября, 7, БГТУ. 58-82-49
Лаборатория оперативной полиграфии БГТУ, ул. Харьковская, 9

1. ЦЕЛЬ РАБОТЫ

Целью работы является изучение основных классов предназначенных для работы с коллекциями, а также овладение практическими навыками составления алгоритмов с их использованием.

Продолжительность работы – 4 ч.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Для хранения однородных данных могут использоваться массивы. Недостатком использования массивов является то, что количество элементов жестко определяется при выделении памяти под массив. Если для решаемых задач необходимо динамическое изменение количества элементов, то нужно воспользоваться коллекциями.

Основные классы, предназначенные для работы с коллекциями, находятся в пространстве имен *System.Collections*. Обобщенные классы коллекций находятся в пространстве имен *System.Collections.Generic*. Классы коллекций, предназначенные для работы с типизированными данными, находятся в пространстве имен *System.Collections.Specialized*.

В табл. 1 приведены классы коллекций и их назначение.

Таблица 1

Основные классы для работы с коллекциями данных

Название	Описание
Пространство имен System.Collections	
ArrayList	Необобщенный список, предназначен для работы с данными любого типа производного от *Object*.
BitArray	Предназначен для работы с динамическим массивом, состоящим из булевских значений, где значение *true* представлено единицей, а *false* – 0.
Hashtable	Представляет коллекцию пар *ключ*/*значение*, которые организованы на основе хэш-кода ключа.
Queue	Динамический список «очередь», работающий по правилу FIFO («first in, first out», с англ. «первым пришел, первым ушел»).

Окончание табл. 1

Название	Описание
SortedList	Представляет коллекцию пар *ключ*/*значение*, которые отсортированы по ключам и доступны по ключу и по индексу.
Stack	Динамический список «стек», работающий по правилу LIFO («last in, first out», с англ. «последним пришел, первым ушел»).
Пространство имен System.Collections.Generic	
Dictionary<TKey, TValue>	Представляет коллекцию ключей и значений.
LinkedList<T>	Представляет двунаправленный список.
List<T>	Представляет строго типизированный список объектов, которые могут быть доступны по индексу.
Queue<T>	Динамический список «очередь», работающий по правилу FIFO («first in, first out», с англ. «первым пришел, первым ушел»).
SortedList<TKey, TValue>	Представляет коллекцию пар *ключ*/*значение*, которые отсортированы по ключу.
Stack<T>	Динамический список «стек», работающий по правилу LIFO («last in, first out», с англ. «последним пришел, первым ушел»).
Пространство имен System.Collections.Specialized	
ListDictionary	Однонаправленный список. Рекомендуется для коллекций, которые обычно включают менее 10 элементов.
NameValueCollection	Представляет коллекцию ассоциированных строковых ключей и строковых значений, которые могут быть доступны по ключу или по индексу.
OrderedDictionary	Представляет коллекцию пар *ключ*/*значение*, которые доступны с помощью ключа или индекса.
StringCollection	Представляет коллекцию строк.
StringDictionary	Реализует хэш-таблицу с ключами и значениями строкового типа.

Рассмотрим подробно особенности работы со следующими классами: *ArrayList, List<T>, LinkedList<T>, Queue<T>, Stack<T>*. Следует отметить, что классы: *ArrayList*, *List<T>* доступны в консольном приложении, а классы: *LinkedList<T>, Queue<T>, Stack<T>* в приложениях «Windows Forms».

2.1. Работа со списками (*ArrayList, List<T>*)

Для работы со списками можно использовать классы *ArrayList* и *List<T>*. Для создания списка необходимо вызвать его конструктор:

```
ArrayList array = new ArrayList();
List<int> list = new List<int>(); // указывается тип (int) элементов списка
```

Каждый из списков имеет емкость, измеряющуюся в количестве элементов. В данном случае емкость массива равна нулю (значение по умолчанию). При добавлении в список одного элемента емкость становится равной 4 элементам. При добавлении 5-го элемента емкость массива увеличивается в два раза. Далее описанный принцип сохраняется.

В случае, если мы знаем количество элементов списка можно при создании списка вызвать конструктор с конкретным значением емкости:

```
ArrayList array = new ArrayList(10);
List<int> list = new List<int>(10);
```

При добавлении 11-го элемента в список, он будет увеличен в два раза, т.е. количество элементов для рассматриваемого примера станет равным 20.

Также можно самостоятельно изменять значение емкости при помощи свойства *Capacity*, например:

```
array.Capacity = 10;
list.Capacity = 10;
```

Во время выделения памяти под список можно сразу заполнить список значениями при помощи инициализатора коллекций:

```
ArrayList array = new ArrayList() { 1, 5 };
List<int> list = new List<int>() { 1, 5 };
```

Для того чтобы получить размер списка можно воспользоваться свойством *Count*:

```
int lengthArray = array.Count; // lengthArray = 2, для предыдущего примера
```

int lengthList = list.Count; // lengthList = 2, для предыдущего примера

Синтаксис обращения к *i*-му элементу массива:

array[i]

list[i]

Приведем некоторые из основных методов для работы с классом *List<T>* (табл. 2), который включает методы, реализованные в классе *ArrayList* и дополняет их новыми.

Таблица 2

Основные методы для работы с классом *List<T>*

Название	Описание
Add	Добавляет объект в конец списка
AddRange	Добавляет коллекцию элементов в конец списка
Clear	Удаляет все элементы из списка
Contains	Определяет, содержится ли указанный элемент в списке
CopyTo	Копирует весь список *List<T>* в совместимый одномерный массив, начиная с первого элемента
Exists	Определяет, содержит ли список *List<T>* элементы, удовлетворяющие условиям указанного предиката.
Find	Поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает первое вхождение в пределах всего списка *List<T>*.
FindAll	Возвращает список элементов, удовлетворяющих условиям указанного предиката.
FindIndex	Поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает отсчитываемый от нуля индекс первого вхождения в список *List<T>*.
FindLast	Поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает последнее вхождение в пределах всего списка *List<T>*.

Окончание табл. 2

FindLastIndex	Поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает отсчитываемый от нуля индекс последнего вхождения в список *List<T>*.
ForEach	Выполняет указанное действие с каждым элементом списка *List<T>*.
IndexOf	Поиск указанного объекта и возвращает индекс первого вхождения в список *List<T>*.
Insert	Вставляет элемент в список *List<T>* по указанному индексу.
InsertRange	Вставляет элементы коллекции в список *List<T>* по указанному индексу.
Remove	Удаляет первое вхождение указанного объекта из списка *List<T>*.
RemoveAll	Удаляет все элементы, удовлетворяющие условиям указанного предиката.
RemoveAt	Удаляет элемент по указанному индексу списка *List<T>*.
Sort	Сортирует элементы списка *List<T>*

Примеры использования методов

Почти каждый из рассматриваемых методов имеет множество вариантов реализации (перегрузок). Рассмотрим некоторые из реализаций представленных ранее методов.

Add

public void Add(T item)

```
List<int> list = new List<int>() {1, 2};
list.Add(5); // list = {1, 2, 5}
```

AddRange

public void AddRange(IEnumerable<T> collection)

```
List<int> list = new List<int>() {1, 2, 3};
List<int> addList = new List<int>() {4, 5};
list.AddRange(addList); // list = {1, 2, 3, 4, 5}
```

Clear

public void Clear()

```
List<int> list = new List<int>() {1, 2, 3};
```

list.Clear(); *// list = {}*

Contains

public bool Contains(T item)

List<int> list = new List<int>() {1, 2, 3};

bool b = list.Contains(3); *// b = true*

CopyTo

public void CopyTo(T[] array)

List<int> list = new List<int>() {1, 2, 3};

int[] array = new int[3];

list.CopyTo(array); *// array = {1, 2, 3}*

Exists

public bool Exists(Predicate<T> match)

List<int> list = new List<int>() {1, 2, 3};

// Передаем адрес функции-предиката (функция, которая возвращает значение булевского типа) isEven, описанную заранее, которая возвращает значение true в случае, если текущий элемент списка четное число

bool b = list.Exists(isEven); *// b = true, т.к. в списке list есть четное число*

// Программный код функции isEven

```
static bool isEven(int dig)
{
  if (dig % 2 == 0) return true;
    else return false;
}
```

Find

public T Find(Predicate<T> match)

List<int> list = new List<int>() {1, -2, -3};

// Передаем адрес функции-предиката isNegative, описанную заранее, которая возвращает значение true в случае, если текущий элемент списка отрицательное число

int elem = list.Find(isNegative); *// elem = -2, первое отрицательное число в списке list*

// Программный код функции isNegative

```
static bool isNegative(int dig)
{
  if (dig < 0) return true;
```

```
    else return false;
}
```

FindAll

public List<T> FindAll(Predicate<T> match)

```
List<int> list = new List<int>() {1, -2, -3};
List<int> list2 = new List<int>();
```

// Передаем адрес функции isNegative, описанную ранее

```
list2 = list.FindAll(isNegative);
```

// list2 = {-2, -3}, список из отрицательных чисел

FindIndex

public int FindIndex(Predicate<T> match)

```
List<int> list = new List<int>() {1, -2, -3};
int index = list.FindIndex(isNegative);
```

// index = 1, индекс первого отрицательного элемента списка

FindLast

public T FindLast(Predicate<T> match)

```
List<int> list = new List<int>() {1, -2, -3};
```

// Передаем адрес функции isNegative, описанную ранее

```
int elem = list.FindLast(isNegative);
```

// elem = -3, последнее отрицательное число в списке list

FindLastIndex

public int FindLastIndex(Predicate<T> match)

```
List<int> list = new List<int>() {1, -2, -3};
int index = list.FindLastIndex(isNegative);
```

// index = 2, индекс последнего отрицательного элемента списка

ForEach

public void ForEach(Action<T> action)

```
List<int> list = new List<int>() {1, -2, -3};
```

// Передаем адрес функции Print, в которой описаны действия для одного элемента списка, а именно вывод элемента списка на экран

```
list.ForEach(Print);
```

// Вывод на экран: 1 -2 -3

// Программный код функции Print

```
static void Print(int dig)
{
  Console.Write("{0, 3}", dig);
}
```

IndexOf

public int IndexOf(T item)

List<int> list = new List<int>() {1, 5, 5};

int index = list.IndexOf(5); *// index = 1*

Insert

public void Insert(int index, T item)

List<int> list = new List<int>() {1, 2, 4};

list.Insert(2, 3); *// list = {1, 2, 3, 4}*

InsertRange

public void InsertRange(int index, IEnumerable<T> collection)

List<int> list = new List<int>() {1, 4, 5};

List<int> list2 = new List<int>() {2, 3};

list.InsertRange(1, list2); *// list = {1, 2, 3, 4, 5}*

Remove

public bool Remove(T item)

List<int> list = new List<int>() {1, 3, 3};

list.Remove(3); *// list = {1, 3}*

RemoveAll

public int RemoveAll(Predicate<T> match)

List<int> list = new List<int>() {1, -2, -3};

// Используем ранее описанный предикат для нахождения отрицательных чисел

list.RemoveAll(isNegative); *// list = {1}*

RemoveAt

public void RemoveAt(int index)

List<int> list = new List<int>() {1, -2, -3};

list.RemoveAt(1); *// list = {1, -3}*

Sort

public void Sort()

List<int> list = new List<int>() {5, 2, -3, 7, 1};

list.Sort(); *// list = {-3, 1, 2, 5, 7}, используется компаратор по умолчанию*

public void Sort(Comparison<T> comparison)

list.Sort(CompareDig); *// list = {7, 5, 2, 1, -3}, используется переданный компаратор*

// Программный код функции CompareDig

static int CompareDig(int dig1, int dig2)

```
{
 if (dig1 < dig2) return 1;
   else return -1;
}
```

2.2. Работа со связанным списком (*LinkedList<T>*)

Принципы работы со связанным списком *LinkedList<T>* такие же, как и с обычным списком *List<T>*, поэтому рассмотрим только особенности работы со списком *LinkedList<T>*.

Приведем некоторые из основных методов для работы с классом *LinkedList<T>* (табл. 3).

Таблица 3

Основные методы для работы с классом *LinkedList<T>*

Название	Описание
AddAfter	Добавляет указанный новый узел после указанного существующего узла в список *LinkedList<T>*.
AddBefore	Добавляет указанный новый узел перед указанным существующим узлом в список *LinkedList<T>*.
AddFirst	Добавляет новый узел, содержащий заданное значение в начале списка *LinkedList<T>*.
AddLast	Добавляет новый узел, содержащий заданное значение в конце списка *LinkedList<T>*.
RemoveFirst	Удаляет узел в начале списка *LinkedList<T>*.
RemoveLast	Удаляет узел в конце списка *LinkedList<T>*.

Примеры использования методов

Почти каждый из рассматриваемых методов имеет множество вариантов реализации (перегрузок). Рассмотрим некоторые из реализаций представленных ранее методов.

Для каждого из представленных ниже примеров предварительно инициализируем связанный список *list* класса *LinkedList<T>* с помощью массива *array*:

```
int[] array = { 1, 2, 3 };
LinkedList<int> list = new LinkedList<int>(array);
```

AddAfter

public LinkedListNode<T> AddAfter(LinkedListNode<T> node, T value)

// В свойствах First и Last хранятся первый и последний элемент списка

list.AddAfter(list.First, 5); *// list = {1, 5, 2, 3}*

AddBefore

public LinkedListNode<T> AddBefore(LinkedListNode<T> node, T value)

list.AddBefore(list.Last, 5); *// list = {1, 2, 5, 3}*

AddFirst

public LinkedListNode<T> AddFirst(T value)

list.AddFirst(5); *// list = {5, 1, 2, 3}*

AddLast

public LinkedListNode<T> AddLast(T value)

list.AddLast(5); *// list = {1, 2, 3, 5}*

RemoveFirst

public void RemoveFirst()

list.RemoveFirst(); *// list = {2, 3}*

RemoveLast

public void RemoveLast()

list.RemoveLast(); *// list = {1, 2}*

2.3. Работа с очередью (*Queue<T>*)

Принципы работы со связанным списком *Queue<T>* такие же, как и с обычным списком *List<T>*, поэтому рассмотрим только особенности работы со списком *Queue<T>*.

Приведем некоторые из основных методов для работы с классом *Queue<T>* (табл. 4).

Таблица 4

Основные методы для работы с классом *Queue<T>*

Название	Описание
Dequeue	Удаляет и возвращает объект, находящийся в начале очереди *Queue<T>*.
Enqueue	Добавляет объект в конец очереди *Queue<T>*.
Peek	Возвращает объект, в начале очереди *Queue<T>*, не удаляя его.

Примеры использования методов

Почти каждый из рассматриваемых методов имеет множество вариантов реализации (перегрузок). Рассмотрим некоторые из реализаций представленных ранее методов.

Для каждого из представленных ниже примеров предварительно инициализируем очередь *queue* класса *Queue<T>* с помощью массива *array*:

```
int[] array = { 1, 2, 3 };
Queue<int> queue = new Queue<int>(array);
```

Dequeue

```
public T Dequeue()
queue.Dequeue(); // queue = {2, 3}
```

Enqueue

```
public void Enqueue(T item)
queue.Enqueue(5); // queue = {1, 2, 3, 5}
```

Peek

```
public T Peek()
int elem = queue.Peek(); // elem = 1
```

2.4. Работа со стеком (*Stack<T>*)

Принципы работы со связанным списком *Stack<T>* такие же, как и с обычным списком *Stack<T>*, поэтому рассмотрим только особенности работы со списком *Stack<T>*.

Приведем некоторые из основных методов для работы с классом *Stack<T>* (табл. 5).

Таблица 5

Основные методы для работы с классом *Stack<T>*

Название	Описание
Peek	Возвращает объект из вершины стека *Stack<T>*, не удаляя его.
Pop	Удаляет и возвращает объект из вершины стека *Stack<T>*.
Push	Вставляет объект в вершину стека *Stack<T>*.

Примеры использования методов

Почти каждый из рассматриваемых методов имеет множество вариантов реализации (перегрузок). Рассмотрим некоторые из реализаций представленных ранее методов.

Для каждого из представленных ниже примеров предварительно инициализируем стек *stack* класса *Stack<T>* с помощью массива *array*:

```
int[] array = { 3, 2, 1 };
Stack<int> stack = new Stack<int>(array);
```

Peek

```
public T Peek()
int elem = stack.Peek(); // elem = 1
```

Pop

```
public T Pop()
stack.Pop(); // stack = {3, 2}
```

Push

```
public void Push(T item)
stack.Push(5); // stack = {3, 2, 1, 5}
```

2.5. Использование цикла *foreach* для работы с коллекциями

При работе с коллекциями удобно использовать цикл *foreach*, который позволяет обратиться к каждому элементу коллекции.

Синтаксис цикла foreach:

foreach (<элемент> in <массив или коллекция>) <оператор>

Оператор *foreach* повторяет группу вложенных операторов для каждого элемента массива или коллекции объектов.

Оператор *foreach* используется для итерации коллекции с целью получения необходимой информации, однако его не следует использовать для добавления или удаления элементов исходной коллекции во избежание непредвиденных побочных эффектов. Если нужно добавить или удалить элементы исходной коллекции, следует использовать цикл *for*.

Для примера работы цикла *foreach* создадим коллекцию класса *ArrayList* и выведем на экран все элементы коллекции.

```
ArrayList array = new ArrayList() {1, 2, 3};
foreach (int elem in array) Console.Write("{0, 2}", elem);
```

После выполнения цикла на экране получим: *1 2 3*

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Для выполнения лабораторной работы необходимо предварительно ознакомиться с теоретической частью, в случае необходимости можно использовать дополнительный теоретический материал. После ознакомления с теорией необходимо ознакомиться с заданием (см. список заданий) и составить алгоритм решения задачи. Далее требуется на языке C# написать программу, реализующую составленный алгоритм.

Общие требования к программе

1. Каждый элемент списка должен быть структурного типа данных (*struct*), который включает необходимые описания по выданной предметной области, например для задания «Написать программу учета книг в библиотеке» структура будет выглядеть следующим образом:

```
struct EventRecord {
        DateTime dateStart; // дата выдачи книги
        DateTime timeStart; // время выдачи книги
        DateTime dateReturn; // дата возврата книги
        string readerName; // имя читателя
        string readerSurname; // фамилия читателя
        long   readerCode; // уникальный код читателя
        string bookTitle; // название книги
        string bookAuthor; // фамилия первого автора книги
        string bookCode; // уникальный код книги
}
```

Список для такой структуры будет выглядеть следующим образом:

```
List<EventRecord> listEvents = new List<EventRecord>();
```

2. Необходимо обеспечить добавление, удаление и редактирование выбранного элемента списка.

3. Требуется обеспечить поиск информации в списке с указанием данных по основным полям структуры, например для приведенного примера, мы указываем уникальный код читателя (*readerCode*) и получаем все события связанные с ним.

4. СПИСОК ЗАДАНИЙ

Ниже представлен список заданий. При выполнении задания необходимо соблюдать общие требования к программе.

1. Написать программу посещения студентами компьютерной аудитории. Аудитория для внеурочных занятий свободна, когда там не проводятся занятия по расписанию.
2. Написать программу учета сдачи экзаменов студентами одной группы. Предусмотреть выдачу задолжников, а также студентов заслуживших повышенную стипендию.
3. Написать программу расписания пассажирских поездов. При желании можно получить информацию на любой временной период, при этом для каждого поезда должен быть установлен статус (прибыл, выехал, задерживается, производится посадка).
4. Написать программу учета выполнения распоряжений руководства на предприятии. При выполнении задания необходимо его помечать как выполненное, при невыполнении в указанную дату, требуется помечать на какую дату было перенесено его выполнение.
5. Написать программу учета книг в библиотеке. Необходимо предусмотреть выдачу информации о задолжниках.
6. Написать программу учета продажи и поступления товаров в магазин. Предусмотреть выдачу товаров с истекающим сроком годности.
7. Написать программу учета студентов выступающих на конференции. Должны быть предусмотрены секции конференции и определена последовательность выступающих.
8. Написать программу учета платежей осуществляемых на автозаправочных станциях.
9. Написать программу учета звонков (входящий, исходящий, непринятый) мобильного телефона.
10. Написать программу учета купленных товаров в супермаркете.
11. Написать программу, ведущую электронный учет детей на зачисление их в детский сад.
12. Написать программу учета сдачи студентами спортивных нормативов.
13. Написать программу расписания пассажирских авиарейсов. При желании можно получить список текущих рейсов, а также информацию на любой временной период, при этом для каждого рейса должен быть установлен статус (вылетел, прилетел, задерживается, производится посадка).
14. Человек путешествует по городам и посещает музеи. Человек отмечает посещенные города, музеи и время посещения. Программа

по указанию временного интервала распечатывает посещенные города и музеи с возможностью получения оплаченной стоимости и длительности посещения.

15. Написать программу учета пациентов на прием к врачу в поликлинике. По запросу можно узнать, сколько пациентов и какие именно записаны к конкретному врачу, а также, сколько врач принял пациентов за указанный период времени.
16. Написать программу учета машин на автостоянке.
17. Написать программу учета вопросов граждан России к президенту РФ. Для каждого вопроса должна быть определена категория: политика, социальный, пожелание, личный и т.п. Предусмотреть возможность случайного выбора «желательного» вопроса.
18. Написать программу маршрута автобусов и троллейбусов. В каждом элементе списка можно хранить информацию об остановках. Обеспечить возможность просмотра номеров маршрутов автобусов и троллейбусов, идущих до необходимой остановки.
19. Написать программу учета мебели и оргтехники на предприятии.
20. Написать программу банковских операций со счетом. При желании можно получить список совершенных действий со счетом.
21. Написать программу учета лекарственных средств сети аптек одного владельца. При желании можно узнать есть ли указанное лекарство в этой сети, и в каком объеме, а также, по какому адресу расположена найденная аптека.
22. Составить список людей, находящихся в спортивной секции. Участники разбиты по возрастным группам. При желании можно получить список людей с высокими спортивными достижениями и хорошим потенциалом.
23. Написать программу учета кредитных займов. Для каждого клиента должна быть определена своя схема выплаты денег по кредиту. Обеспечить возможность хранения процентной ставки отдельно для каждого клиента.
24. Написать программу работы агентства недвижимости. Клиенты оставляют предложения о покупках и продажах. Требуется организовать систему запросов, позволяющую найти похожие варианты.
25. Написать программу учета посещений в тренажерном зале. Каждое занятие включает описание программы тренировки.

26. Написать программу аренды рекламных щитов. При желании можно получить список свободных щитов с указанием их местоположения (адреса) для запрашиваемого временного периода.
27. Написать программу «Записная книжка». Каждый элемент списка является множеством записей сделанных на конкретный день. При желании можно получить список встреч, событий на указанный период времени.
28. Написать программу учета торговых операций на валютной бирже. По запросу можно получить максимальную и минимальную стоимость указанной валюты в течение указанного периода.
29. Написать программу учета ремонтных операций в автомастерских. По запросу можно получить список наиболее прибыльных работ.
30. Написать программу учета расходов и доходов семейного бюджета. При желании можно получить информацию о расходах и доходах для указанного периода.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В каких пространствах имен находятся классы для работы с коллекциями?
2. Назовите основные классы для работы с коллекциями.
3. Как инициализировать списки *ArrayList* и *List<T>*?
4. Каким образом можно установить емкость списка?
5. Как обратиться к элементам списков созданных на основе следующих классов: *ArrayList*, *List<T>*, *LinkedList<T>*, *Queue<T>*, *Stack<T>*?
6. Каким образом добавить и удалить элементы классов: *Queue<T>*, *Stack<T>*?
7. Что такое функция-предикат и как ее можно использовать при работе с коллекциями?
8. Чем отличается список *List<T>* от списка *LinkedList<T>*?
9. Каково назначение списков *Queue<T>* и *Stack<T>*?

6. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основная

1. Павловская Т. А. C#. Программирование на языке высокого уровня. – Изд.: Питер, 2009. – 432с.

2. Эндрю Троелсен. Язык программирования C# 2010 и платформа .NET 4. – Изд.: Вильямс, 2011. – 1392с.
3. Кристиан Нейгел, Билл Ивьен, Джей Глинн, Карли Уотсон, Морган Скиннер. C# 4.0 и платформа .NET 4 для профессионалов. – Изд.: Питер, 2011. – 1440с.

Дополнительная

4. Тыртышников Е.Е. Методы численного анализа. Учебное пособие. – Изд.: МГУ, 2006. – 281с.
5. Джесс Либерти. Программирование на C#. – Изд.: КноРус, 2003. – 688с.
6. Харви Дейтел. C# в подлиннике. Наиболее полное руководство. – Изд.: БХВ-Петербург, 2006. – 1056с.