Практическая работа 31
Работа с динамическими структурами

Цель занятия: Получить практические навыки работы с динамическими структурами

Перечень оборудования и программного обеспечения

Персональный компьютер
Microsoft Office (Word, Visio)
Microsoft Visual Studio

Краткие теоретические сведения

Динамические структуры данных – это структуры данных, память под которые выделяется и освобождается по мере необходимости.

Динамические структуры данных в процессе существования в памяти могут изменять не только число составляющих их элементов, но и характер связей между элементами. При этом не учитывается изменение содержимого самих элементов данных. Такая особенность динамических структур, как непостоянство их размера и характера отношений между элементами, приводит к тому, что на этапе создания машинного кода программа-компилятор не может выделить для всей структуры в целом участок памяти фиксированного размера, а также не может сопоставить с отдельными компонентами структуры конкретные адреса. Для решения проблемы адресации динамических структур данных используется метод, называемый динамическим распределением памяти, то есть память под отдельные элементы выделяется в момент, когда они "начинают существовать" в процессе выполнения программы, а не во время компиляции. Компилятор в этом случае выделяет фиксированный объем памяти для хранения адреса динамически размещаемого элемента, а не самого элемента.

Динамическая структура данных характеризуется тем что:

- она не имеет имени;

- ей выделяется память в процессе выполнения программы;

- количество элементов структуры может не фиксироваться;

- размерность структуры может меняться в процессе выполнения программы;

- в процессе выполнения программы может меняться характер взаимосвязи между элементами структуры.

Необходимость в динамических структурах данных обычно возникает в следующих случаях:

- используются переменные, имеющие довольно большой размер (например, массивы большой размерности), необходимые в одних частях программы и совершенно не нужные в других;

- процессе работы программы нужен массив, список или иная структура, размер которой изменяется в широких пределах и трудно предсказуем;

- размер данных, обрабатываемых в программе, превышает объем сегмента данных.

Примером динамических структур являются коллекции. Преимуществом коллекций является то, что их размер динамичен.

Еще одни плюсом коллекций является то, что они уже имеют внутреннюю реализацию различных структур данных, начиная со списка и заканчивая хеш-таблицей. Таким образом, в большинстве случаев не требуется реализовывать методы добавления/поиска/удаления элемента из структуры данных, т.к. эти методы уже реализованы внутри коллекции.

В C# коллекции содержатся в пространствах имен System.Collections и Systems.Collections.Generic. Коллекции подразделяются на универсальные и неуниверсальные. Универсальные коллекции могут быть типизированы и содержать объекты определенного типа. Это позволяет выигрывать в производительности, особенно, если коллекции хранят типы значений (int, bool, double и т.п.). В неуниверсальных коллекциях происходит упаковка хранимых объектов в тип Object, что требует их последующей распаковки при работе с ними.

Структура данных «список» в C# реализована как универсальная коллекция List, кроме того, есть аналог ArrayList - неуниверсальная коллекция. Для получения текущего количества элементов в коллекции можно использовать свойство Count.

Обозначим основные методы по работе с коллекциями List и ArrayList.

Добавление элементов:

Add (элемент_коллекции); - добавляет элемент в коллекцию. В случае универсальной коллекции тип элемента должен совпадать с типом коллекции (пример был приведен выше)

AddRange (Коллекция); - добавляет в текущую коллекцию совокупность элементов заданной коллекции. При этом, коллекция, указанная в качестве параметра, должна реализовывать интерфейс ICollection (стандартный интерфейс коллекции, замечание значимо при использовании пользовательской коллекции).

Insert (место, элемент); - вставляет в заданное место (индекс элемента) заданный элемент

InsertRange(место, коллекция); - вставляет в заданное место в коллекции заданную коллекцию

Удаление элементов:

Remove(элемент); - удаляет элемент из коллекции (первое вхождение элемента)

RemoveAt(позиция); - удаляет элемент на заданной позиции

RemoveRange(позиция, количество); - удаляет заданное количество элементов, начиная с заданной позиции

Clear(); - удаляет все элементы из коллекции

Поиск элементов:

boolContains(элемент); - возвращает, содержится ли заданный элемент в коллекции

intIndexOf(элемент); - возвращает позицию, на которой содержится в коллекции заданный элемент (первое вхождение)

intIndexOf(элемент, позиция); - возвращает позицию, на которой содержится заданный элемент (первое вхождение). Поиск осуществляется по коллекции, начиная с заданной и заканчивая концом коллекции)

intIndexOf(элемент, стартовая_позиция, финишная_позиция); - возвращает позицию, на которой содержится заданный элемент (первое вхождение). Поиск осуществляется по коллекции, начиная с заданной стартовой позиции и заканчивая заданной финишной позицией.

intLastIndexOf(элемент); - возвращает позицию, на которой содержится в коллекции заданный элемент (последнее вхождение). Метод перегружен, его перегруженные версии копируют перегрузки метода IndexOf(), рассмотренного выше

intBinarySearch(элемент); - возвращает позицию, на которой находится заданный элемент в коллекции. По коллекции осуществляется бинарный (двоичный) поиск. Коллекция должная быть отсортирована. Метод содержит несколько перегрузок, можно также задавать компаратор (метод сравнения) для элементов.

Дополнительные методы для работы с коллекциями:

Sort(); - сортирует коллекцию, используя стандартный компаратор. Если тип элементов коллекции является пользовательским, то компаратор (сравнение элементов) нужно реализовать в соответствующем классе этого типа.

stringToString(); - стандартный метод, переводящий коллекцию в строку

Reverse(); - переворачивает коллекцию, точнее, порядок элементов в коллекции. Первый элемент меняется с последним и т.д.

Reverse(стартовая_позиция, количество_элементов); - переворачивает в коллекции заданное количество элементов, начиная с заданного номера элемента

GetRange(стартовая_позиция, количество элементов); - возвращает коллекцию, элементами которой является заданное количество элементов данной коллекции, начиная с заданной стартовой позиции

GetType(); - возвращает тип элементов коллекции

CopyTo(массив); - копирует элементы коллекции в заданный массив

CopyTo(массив, номер_позиции); - копирует элементы коллекции в заданный массив, начиная с заданной позиции в массиве

CopyTo(массив, номер_позиции, количество_элементов); - копирует заданное количество элементов коллекции в заданный массив, начиная с заданной позиции в массиве.

Элемент управления ListBox является примером динамического списка.

Элемент управления ListBox позволяет отображать несколько элементов, позволяя пользователям прокручивать длинный список.

Применение коллекций ArrayList и элемент управления ListBox

Создайте класс

```
class Cat
{
  public int Age { get; set; }
  public string Name { get; set; }
}
```

Поместите на форму элементы управления ListBox и label. В обработчике Form1_Load добавьте следующий код:

```
private void Form1_Load(object sender, EventArgs e)
{       // Определим универсальную коллекцию
 List<Cat> cats = new List<Cat>
{
 new Cat(){ Name = "Push", Age=8 },
 new Cat(){ Name = "Whiskers",  Age=2 },
 new Cat(){ Name = "Sasha",     Age=14 }
};

  foreach (Cat c in cats)
     if (c != null)
        listBox1.Items.Add(c.Name);
     else
        MessageBox.Show("List не определён");
 }
```

В обработчике listBox1_SelectedIndexChanged добавьте следующий код:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  label1.Text = listBox1.SelectedItem.ToString();
}
```

Задания

1 Изучить теоретические сведения и задание к работе

2 В соответствии с вариантом задания разработать отлаженную программу.

Порядок выполнения работы

…

Содержание отчета

1 Название работы
2 Цель работы
3 Технические средства обучения
4 Задания (условия задач)
5 Порядок выполнения работы
6 Вывод

Варианты заданий

Варианты заданий указаны в практической работе 28.

Используемая литература

1. Гниденко, И. Г. Технология разработки программного обеспечения: учеб. пособие для СПО / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. — М.: Издательство Юрайт, 2017.
2. Шарп Джон Ш26 Microsoft Visual C#. Подробное руководство. 8-е изд. — СПб.: Питер, 2017.
3. Васильев А.Н. Программирование на C# для начинающих. Основные сведения. – Москва: Эксмо, 2018.
4. Васильев А.Н. Программирование на C# для начинающих. Особенности языка. – Москва: Эксмо, 2019.
5. http://msdn.microsoft.com/ru-ru/library/67ef8sbd.aspx.