ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ
(6 СЕМЕСТР)
ЛАБОРАТОНАЯ РАБОТА №1.2
СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

Скирневский И.П.

Томск – 2016

ОГЛАВЛЕНИЕ

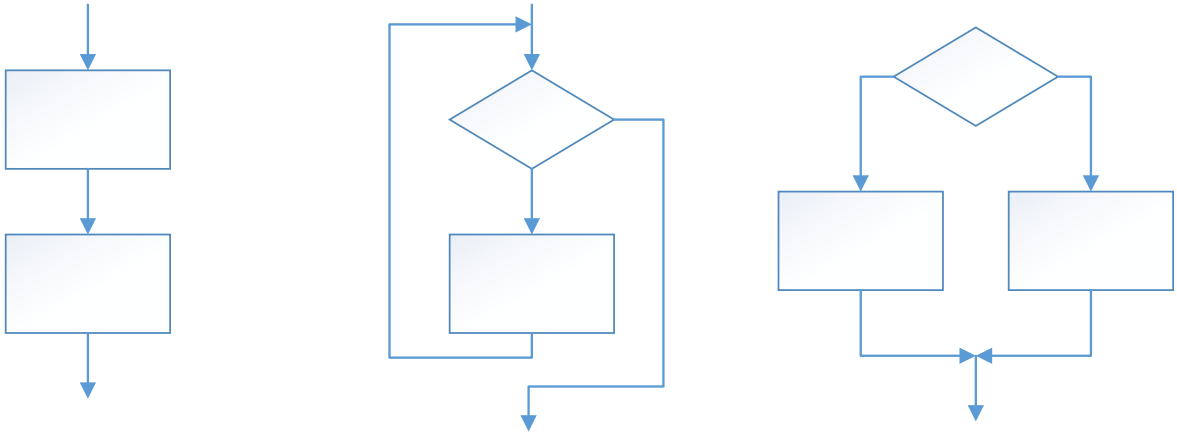
ЛАБАРАТОРНАЯ №1.2. СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

1.1.Основы структурного программирования

Структурное программирование — методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. Предложена в 1970-х годах Э. Дейкстрой и др.

В соответствии с данной методологией любая программа строится без использования оператора goto из трёх базовых управляющих структур: последовательность, ветвление, цикл; кроме того, используются подпрограммы, речь о которых пойдет в следующей лабораторной работе. При этом разработка программы ведётся пошагово, методом «сверху вниз».В теории программирования уже давно было доказано, что для решения задач совершенно любой сложности, начиная от примитивных окон и заканчивая сложными информационными системами можно составить программу, которая будет состоять только из трех структур, о которых было упомянуто выше. Официально такое исследование в 1966 году было проведено Боймом Якопини.

Схематично, структуры следование, цикл и ветвление можно представить следующим образом. (рисунок 13)



Следование Цикл Ветвление

Рисунок 1 – Управляющие структуры

Ниже приведено краткое описание каждый структуры:

– Следованием называется конструкция, представляющая собой последовательное выполнение двух или более операторов (простых или составных);

– Цикл задает многократное выполнение оператора;

– Ветвление задает выполнение либо одного, либо другого оператора в зависимости от выполнения какого-либо условия.

Идеей использования базовых конструкций является получение программы простой структуры. Такую программу легко читать (а программы чаще приходится читать, чем писать), отлаживать и при необходимости вносить в нее изменения. Структурное программирование, как вы могли заметить часто называли «программированием без goto», и в этом есть большая доля правды: частое использование операторов передачи управления в произвольные точки программы затрудняет прослеживание логики ее работы.

В большинстве языков высокого уровня существует несколько реализаций базовых конструкций; в C# есть четыре вида циклов и два вида ветвлений (на два и на произвольное количество направлений). Они введены для удобства программирования, и в каждом случае надо выбирать наиболее подходящие средства. Главное, о чем нужно помнить даже при написании самых простых программ, — что они должны состоять из четкой последовательности блоков строго определенной конфигурации.

1.1.1. Циклы

Как было указанно выше циклы являются управляющими конструкциями, позволяя в зависимости от определенных условий выполнять некоторое действие множество раз. В C# имеются следующие виды циклов:

1. for
2. foreach
3. while
4. do...while

Давайте разберем каждый вид отдельно и рассмотрим несколько примеров.

Цикл for

Цикл for имеет следующее формальное определение:

```
for ([инициализация счетчика]; [условие]; [изменение счетчика])
{
    // действия
}
```

Рассмотрим небольшой пример использования цикла for. Стандартный пример, прохождение массива из 10 элементов.

```
for (int i = 0; i< 9; i++)
{
    Console.WriteLine("Квадрат числа {0} равен {1}", i, i* i);
}
```

Первая часть объявления цикла – int i = 0 - создает и инициализирует счетчик i. (буква i выбрана в связи с устоявшейся традицией, хотя использоваться может и любой другой символ). Счетчик необязательно должен представлять тип int. Это может быть и

другой числовой тип, например, float. И перед выполнением цикла его значение будет равно 0. В данном случае это то же самое, что и объявление переменной.

Вторая часть - условие, при котором будет выполняться цикл. В данном случае цикл будет выполняться, пока i не достигнет 9.

И третья часть - приращение счетчика на единицу. Опять же нам необязательно увеличивать на единицу. Можно уменьшать: i--.

В итоге блок цикла сработает 9 раз, пока значение i не станет равным 9. И каждый раз это значение будет увеличиваться на 1. Нам необязательно указывать все условия при объявлении цикла. Например, цикл может быть реализован следующим образом:

```
int i = 0;
for (; ;)
{
    Console.WriteLine("Квадрат числа {0} равен {1}", ++i, i* i);
    System.Threading.Thread.Sleep(500);
}
```

Формально определение цикла осталось тем же, только теперь блоки в определении у нас пустые: for (; ;). У нас нет инициализированной переменной-счетчика, нет условия, поэтому цикл будет работать вечно – бесконечный цикл.

Мы также можем опустить ряд блоков, как показано в примере ниже.

```
int i = 0;
for (; i<9;)
{
    Console.WriteLine("Квадрат числа {0} равен {1}", ++i, i* i);
}
```

Этот пример по сути эквивалентен первому примеру: у нас также есть счетчик, только создан он вне цикла. У нас есть условие выполнения цикла. И есть приращение счетчика уже в самом блоке for.

Цикл foreach

Исторически сложилось, что одним из наиболее трудных для понимания циклов является цикл – foreach. На самом деле в нем ничего сложно и сейчас мы в этом убедимся.

Цикл foreach предназначен для перебора элементов в контейнерах. Формальное объявление цикла foreach:

```
foreach (тип_данных название_переменной in контейнер)
{
    // действия
}
```

Для того чтобы ситуация стала более понятной, приведем небольшой пример. Допустим у нас есть массив из 5 элементов и нам надо обратиться к каждому из элементов массива, чтобы вывести его значение на экран. Код приведен ниже.

```
int[] array = new int[] { 1, 2, 3, 4, 5 };
foreach (int i in array)
{
    Console.WriteLine(i);
}
```

Обратите внимание, что в примере в качестве контейнера выступает массив данных типа int. Поэтому мы объявляем переменную с типом int.

Внимательный читатель может заметить, что абсолютно тоже можно сделать и при помощи цикла for. Вы можете проделать этот пример самостоятельно. Несмотря на лаконичную конструкция цикла foreach, цикл for более гибкий по сравнению с foreach. Если foreach последовательно извлекает элементы контейнера и только для чтения, то в цикле for мы можем перескакивать на несколько элементов вперед в зависимости от приращения счетчика, а также можем изменять элементы

```
int[] array = new int[] { 1, 2, 3, 4, 5 };
for (int i = 0; i<array.Length; i++)
{
    array[i] = array[i] * 2;
    Console.WriteLine(array[i]);
}
```

Разберите самостоятельно, что делает пример приведенный выше.

Цикл do

В цикле do сначала выполняется код цикла, а потом происходит проверка условия в инструкции while. И пока это условие истинно, цикл повторяется. Например, следующих код уменьшает значение переменной i на единицу, до тех пор, пока переменная больше нуля.

```
int i = 6;
do
{
    Console.WriteLine(i);
    i--;
}
while (i > 0);
```

Наверно вы уже догадались, что код цикла сработает 6 раз, пока i не станет равным нулю. Но важно отметить, что цикл do гарантирует хотя бы однократное выполнение действий, даже если условие в инструкции while не будет истинно. Фактически, задавая начальное значение для i = -1 цикл гарантированно выполнится один раз. Попробуйте проверить это самостоятельно.

Цикл while

В отличие от цикла do цикл while сразу проверяет истинность некоторого условия, и если условие истинно, то код цикла выполняется:

```
int i = 6;
while (i > 0)
{
    Console.WriteLine(i);
    i--;
}
```

Попробуйте задать начальное условие для переменной i равное -1 и запустить цикл. Сравните полученный результат с циклом do…while.

Операторы continue и break

Иногда возникает ситуация, когда требуется выйти из цикла, не дожидаясь его завершения. В этом случае мы можем воспользоваться оператором break. Пример приведен ниже.

```
int[] array = new int[] { 1, 2, 3, 4, 12, 9 };
for (int i = 0; i<array.Length; i++)
{
    if (array[i] > 10)
        break;
    Console.WriteLine(array[i]);
}
```

Поскольку в цикле идет проверка, больше ли элемент массива 10. То мы никогда не увидим на консоли последние два элемента, так как, увидев, что элемент массива больше 10, сработает оператор break, и цикл завершится.

Теперь поставим себе другую задачу. А что, если мы хотим, чтобы при проверке цикл не завершался, а просто переходил к следующему элементу. Для этого мы можем воспользоваться оператором continue:

```
int[] array = new int[] { 1, 2, 3, 4, 12, 9 };
for (int i = 0; i<array.Length; i++)
{
    if (array[i] > 10)
        continue;
    Console.WriteLine(array[i]);
}
```

В этом случае цикл, когда дойдет до числа 12, которое не удовлетворяет условию проверки, просто пропустит это число и перейдет к следующему элементу массива.

1.1.2. Ветвления / Условные конструкции

Условные конструкции – один из базовых компонентов многих языков программирования, которые направляют работу программы по одному из путей в зависимости от определенных условий.

В языке C# используются следующие условные конструкции: if..else и switch..case

Конструкция if/else

Конструкция if/else проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет определенный код. Давайте рассмотрим небольшой пример использования оператора.

```
int num1 = 8;
int num2 = 6;
if(num1 > num2)
{
    Console.WriteLine("Число {0} больше числа {1}", num1, num2);
}
```

После ключевого слова if ставится условие. И если это условие выполняется, то срабатывает код, который помещен далее в блоке if после фигурных скобок. В качестве условий выступают ранее рассмотренные операции сравнения.

В данном случае у нас первое число больше второго, поэтому выражение num1 > num2 истинно и возвращает true, следовательно, управление переходит к строке Console.WriteLine("Число {0} больше числа {1}", num1, num2);

Но что, если мы захотим, чтобы при несоблюдении условия также выполнялись какие-либо действия? В этом случае мы можем добавить блок else:

```
int num1 = 8;
int num2 = 6;
if(num1 > num2)
{
    Console.WriteLine("Число {0} больше числа {1}", num1, num2);
}
else
{
    Console.WriteLine("Число {0} меньше числа {1}", num1, num2);
}
```

Но при сравнении чисел мы можем насчитать три состояния: первое число больше второго, первое число меньше второго и числа равны. Используя конструкцию else if, мы можем обрабатывать дополнительные условия:

```
int num1 = 8;
int num2 = 6;
if(num1 > num2)
{
    Console.WriteLine("Число {0} больше числа {1}", num1, num2);
}
else if (num1<num2)
{
    Console.WriteLine("Число {0} меньше числа {1}", num1, num2);
}
else
{
    Console.WriteLine("Число num1 равно числу num2");
}
```

Также мы можем соединить сразу несколько условий, используя логические операторы:

```
int num1 = 8;
int num2 = 6;
if(num1 > num2 && num1>8)
{
    Console.WriteLine("Число {0} больше числа {1}", num1, num2);
}
```

Обратите внимание на конструкцию &&, которая выполняет логическое умножение, другими словами операцию «И». Это говорит о том, что блок будет истинен только при выполнении обоих условий. Не трудно догадаться, что существует логическое сложение, оператор «ИЛИ» – в языке C# обозначается как «||» две прямых черты.

Важно!

Распространенной ошибкой является использование оператора присваивания «=» внутри блока if. Важно понимать отличие между оператором присваивания и оператором проверки на равенство «==». Внутри блока if может использоваться только оператор проверки на равенство, так как он может вернуть значение истина или ложь, в зависимости от того, равны переменные или нет. Таким образом правильно будет if (a == b). Будьте внимательны.

Конструкция switch

Конструкция switch/case аналогична конструкции if/else, так как позволяет обработать сразу несколько условий. Рассмотрим пример.

```
Console.WriteLine("Нажмите Y или N");
string selection = Console.ReadLine();
switch (selection)
{
    case "Y":
        Console.WriteLine("Вы нажали букву Y");
        break;
    case "N":
        Console.WriteLine("Вы нажали букву N");
        break;
    default:
        Console.WriteLine("Вы нажали неизвестную букву");
        break;
}
```

После ключевого слова switch в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора case. И если совпадение будет найдено, то будет выполняться определенный блок case.В конце блока case ставится оператор break, чтобы избежать выполнения других блоков. Если мы хотим также обработать ситуацию, когда совпадения не будет найдено, то можно добавить блок default, как в примере выше.

1.1.3. Тернарная операция

Тернарную операция имеет следующий синтаксис:

[первый операнд - условие] ? [второй операнд] : [третий операнд].

Здесь сразу три операнда. В зависимости от условия тернарная операция возвращает второй или третий операнд: если условие равно true, то возвращается второй операнд; если условие равно false, то третий. В некотором роде, тернарная операция представляет из себя сокращенный синтаксис операции if/else, но все-таки имеет ряд отличий. Пример ниже:

```
int x = 3;
int y = 2;
Console.WriteLine("Нажмите + или -");
string selection = Console.ReadLine();

int z = selection == "+" ? (x + y) : (x - y);
Console.WriteLine(z);
```

Здесь результатом тернарной операции является переменная z. Если мы выше вводим "+", то z будет равно второму операнду - (x+y). Иначе z будет равно третьему операнду.

1.2. Реализация приложения «Записная книжка»

В качестве примера рассмотрим создание простой записной книжки. Читатель может либо проделать всю последовательность шагов и выполнить тестовый проект, либо сразу выполнять задание в соответствии со своим вариантом.

Хотелось бы обратить внимание, что данный пример не является оптимальным решением задачи. И в ходе прохождения лабораторного практикума, мы будем модернизировать его, используя новые технологии и подходы. А пока не будем забегать вперед и создадим новое консольное приложение под названием Notebook.

Примечание. В данной лабораторной работе не будет описано создание проекта и работа со средой Visual Studio. Предполагается, что на момент выполнения лабораторной работы студент уже обладает обозначенными знаниями, которые давались в лабораторной работе №1 данного лабораторного практикума.

В процессе выполнения лабораторной работы мы будем добавлять весь код в функцию Main без использования каких-либо дополнительных конструкций.

1.2.1. Создание хранилища записей

В нашем случае в качестве будет использоваться обыкновенная переменная – строковый массив. Это говорит о том, что все данные, записанные в записную книжку будут удалены после каждого закрытия программы. Согласен с вами, не самая удобная записная книжка. Для хранения строковых данных использовать несколько видов контейнеров. Одним из самых простых решений - массив строк. Мы пропустим данный вариант, так как он не является ключевой частью лабораторной работы. Для удобства реализации в качестве

хранилища записей будущей записной книжки используем так называемый список List. Детально ознакомится со всеми особенностями списка List можно вызвав справку. Пока же просто будем использовать список List<T> не вдаваясь в подробности данной структуры. И так, создаем список записей, как показано в примере ниже.

```
List<String> notes = new List<string>();
```

В данном случае <String> указывает на то, что наш список хранит строки. Одна строка – одна запись в списке. Конструкция new List<string>() нужна для инициализации нового списка. И так, наш новый список называется notes, это объект типа List<String>. Списки очень удобны в плане добавления и удаления записей. Так, для добавления новой записи в наш список notes воспользуемся командой Add. Добавим несколько первоначальных значений, как показано в примере ниже.

Добавим пару начальных записей.

```
notes.Add("Первая запись");
notes.Add("Вторая запись");
```

И так, мы создали наше, хоть и временное, на хранилище данных. Теперь после каждого включения программы в нем будет находится две записи. Самое время внедрить основные элементы структурного программирования, циклы и ветвления.

1.2.2. Внедрение циклов и ветвлений

Вся логика нашего приложения будет заключаться в том, что пользователю предоставляется список команд:

- Добавить запись
- Удалить запись по номеру n
- Посмотреть все записи
- Посмотреть список доступных команд
- Выйти из приложения

Выбор каждой команды будет осуществляться по нажатия соотвествующей клавиши. Как вы, возможно, уже догадались для реализации подобной логики нам потребуются операторы switch и главный цикл while. И так, обо всем по порядку.

Прежде чем реализовывать циклы, добавим переменную, в которой будет хранится выбранное действие пользователем (нажатая клавиша). Удобнее всего использоваться тип char, в котором будет храниться один единственный символ. Добавляем строку:

```
char action = 'h';
```

Мы название переменную action (действие) и присвоили значение ‘h’. В будущем этот символ будет означить просмотр списка всех команд, это необходимо для того, чтобы при запуске программы первым делом отобразился список доступных команд.

Далее создадим наш главный цикл, в котором будет производится ввод и выполнение команд.

```
while (action != 'q')
{
	// основной цикл
}
```

Изучив предыдущий материал, слушатель без труда определит, что подобный цикл while выполняется пока в переменную action не записан символ q, другими словами пока пользователь не нажмет клавишу q для выхода из приложения.

Теперь необходимо осуществить выполнение действий, соответствующих введенной команде. Для этого будем использовать оператор switch. Как вы помните, оператор switch включает один или несколько разделов переключения. Каждый раздел переключения содержит одну или несколько меток case, за которыми следует один или несколько операторов. Добавляем каркас нашего будущего ветвления. Пока обработаем нажатие только одной кнопки l. В примере ниже после нажатия на l ничего не произойдет, наполнения оператора будет добавлено позже.

```
switch (action)
{
	case 'l':
		// если action равен ‘l’ то выполнить описанные здесь операции
		break;

}
```

Раз мы добавили первую команду, – нажатие на кнопку l, давайте реализуем логику этого запроса. По нажатия на l мы будем выводить для пользователя все записи. Для отображения записей выполнил проход по списку, и вывод каждой записи в отдельную строку.

```
case 'l':
     foreach (var note in notes)
     {
         Console.WriteLine((notes.IndexOf(note)+1) + ") " +note); // т.к. в
списке нумерация начинается с 0, то прибавляем 1
     }
     break;
```

Вспомним материал из предыдущего раздела. Foreach – цикл, который проходи по всем записям в переменной notes, при этом в теле цикла обращение к текущей записи происходит через переменную note. Фактически на каждом шаге цикла переменная note смещается вправо, получает следующее значение, выводит его на экран и идет дальше. (рисунок 18).

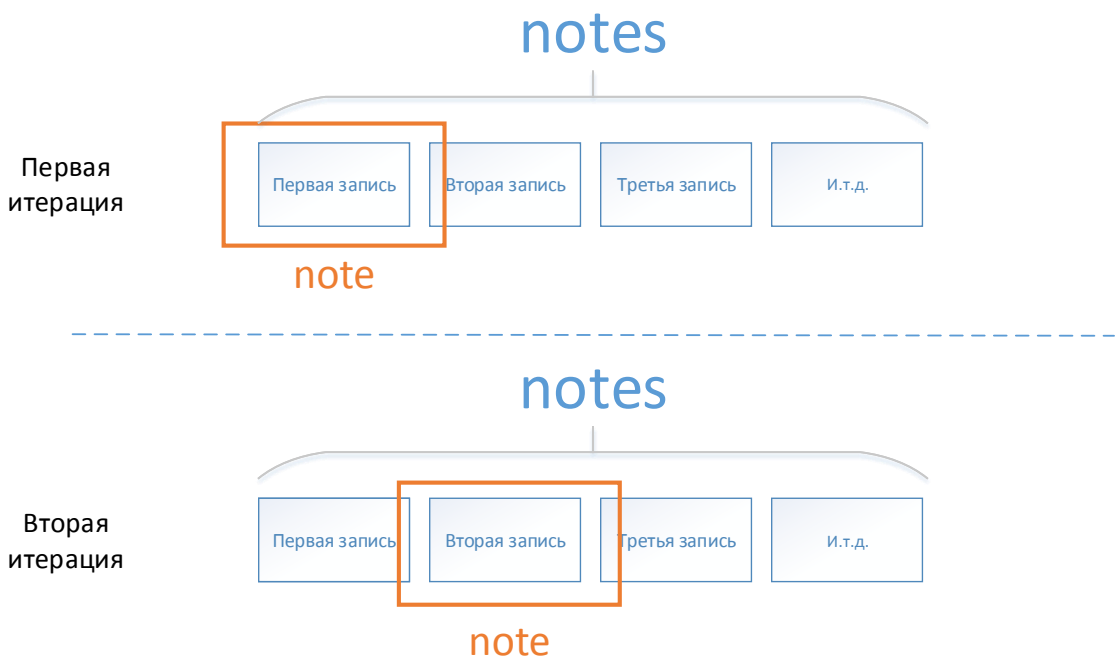


Рисунок 2 – Обход списка операцией foreach

Для вывода записи используется команда Console.WriteLine. Для получения номера записи используется команда notes.IndexOf. Фактически функция IndexOf объекта notes возвращает номер элемента списка, который был в нее передан. Стоит заметить, что в программе нумерация записей начинается с 0, поэтому при выводе мы прибавляем к текущему номеру записи 1 с помощью вот такой конструкции IndexOf(note)+1). Данная конструкция позволит выводить записи в формате «1) Название элемент».

Далее определим операцию вывода доступных команд, мы договорить, что список доступных команд будет выводится при старте программы и при нажатии пользователем кнопки h.

```
case 'h':
    Console.WriteLine("Доступные команды:");
    Console.WriteLine("a - добавить запись");
    Console.WriteLine("d - удалить запись с номером n");
    Console.WriteLine("l - список всех записей");
    Console.WriteLine("h - список доступных команд");
    Console.WriteLine("q - выйти из программы");
    break;
```

Реализуем следующую операцию – добавление новой записи. Пусть она выполняется при нажатии пользователем кнопки «a».

```
case 'a':
        Console.Write("Введите сообщение: ");
        var newNote = Console.ReadLine(); // считываем сообщение
        notes.Add(newNote); // добавляем сообщение в конец списка
        break;
```

С помощью команды Console.ReadLine мы считываем сообщение, вводимое пользователем. А с командой Add мы уже знакомы, она добавил введённую пользователем строку в конец списка.

Теперь определим операцию удаления записи. Удаление будет происходит, если пользователь нажмет клавишу «d».

```
case 'd':
        Console.Write("Введите номер записи для удаления: ");
        int n = Convert.ToInt32(Console.ReadLine())-1;
        if (n < notes.Count && n > -1)
        {
                notes.RemoveAt(n);
        } else
        {
                Console.WriteLine("Записи с указанным номером не существует");
        }
        break;
```

В данном блоке, пользователю предлагается ввести номер записи, которую он хочет удалить. Стоит обратить внимание на то, что команда Console.ReadLine возвращает в результате строку типа string, а нам нужен номер записи типа int (число), поэтому мы преобразуем строку типа string в номер типа int с помощью команды Convert.ToInt32. так же мы вычитаем 1 из введенного номера, поскольку нумерация записей в программе начинается с 0. Затем мы проверяем, что введенный номер принадлежит отрезку [0; notes.Count], где notes.Count – команда для получения количества записей в списке. Если номер введен верно, то удаляем запись из списка с помощью команды notes.RemoveAt, иначе выводим сообщение об ошибке.

Если же ни одна из операций не была выполнена, значит пользователь ввел неизвестную команду. Чтобы оповестить его об этом используем секцию default оператора switch.

```
        default:
                Console.WriteLine("Неизвестная команда");
                break;
```

На этом мы заканчиваем работу с оператором switch. Все возможные действия пользователя были обработаны.

Последнее, что остается сделать, это обеспечить ввод новой команды. Наверное, вы обратили внимание, что до сих пор мы не реализовали участок, который бы предлагал пользователю ввести команду. Добавим его сразу после блока switch. Обратите внимание, что блок ввода новой команды, тем ни менее должен находится внутри цикла while, иначе программа завершит свою работу после выполнения первого действия.

```
        Console.Write("Введите команду: ");
        action = Console.ReadKey().KeyChar; // считываем новое действие
        Console.WriteLine(); // для переноса вывода на новую строку
```

Команда Console.ReadKey в отличие от Console.ReadLine считывает нажатие одной клавиши, а метод .KeyChar позволяет получить символ нажатой клавиши, который мы и сравниваем в операторе switch описанном выше.

Используя псевдокод, структура приложения должна быть следующая:

```
функция Main
{
    Реализация хранилища

    Общий цикл While
    {
        Оператор switch
        {

        }

        Блок ввода новой команды
    }
}
```

1.3. Запуск программы

На этом реализация одной из самых простых версий справочника завершена. К сожалению, общий листинг программы приведен не будет. Результат выполнения программы изображен на рисунке 19. При выполнении тестового задания студенты могут реализовать другие команды и использовать свои обозначения для каждой команды.

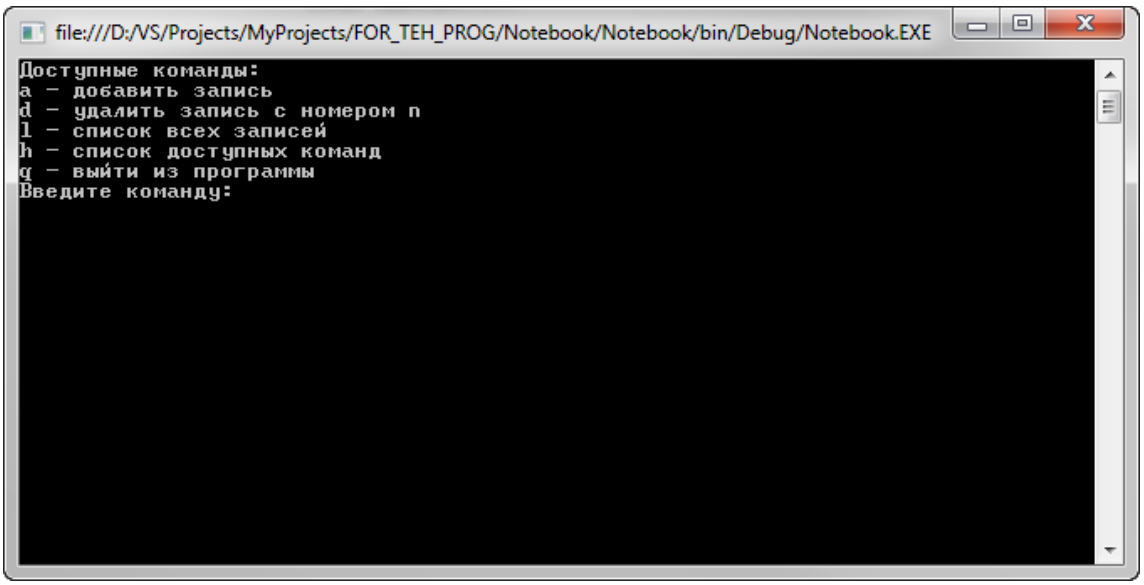


Рисунок 3 – Результат работы программы

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ №2

1. Реализовать консольное приложение в соответствии с вариантом. *Номер варианта соответствует номеру зачетной книжки.*

2. Реализовать блок-схему основных элементов структурного программирования на основе примитивов, приведенных в методическом указании.

3. Для ввода команд пользователем использовать цифры.

4. Добавить одну дополнительную команду, которая не была описана в методическом указании.

5. Оформить отчет с описанием хода выполнения вашей лабораторной работы.

Варианты заданий

1. Телефонный справочник (в одной строке вводить: «ФИО, Телефон»)
2. Записная книжка (в одной строке вводить: «Запись, Дата»)
3. Список записей к врачу (в одной строке вводить: «Название врача, Дата»)
4. Список клиентов автомойки (в одной строке вводить: «Марка машины, Время»)
5. Список заказов в кафе (в одной строке вводить: «Номер стола, Блюдо»)
6. Путеводитель (в одной строке вводить: «Название места, Описание»)
7. Гостевая книга (в одной строке вводить: «Сообщение, Время»)
8. Почтовые данные (в одной строке вводить: «Адрес, Индекс»)
9. Список студентов (в одной строке вводить: «ФИО, Номер группы»)
10. Список рейсов (в одной строке вводить: «Название рейса, Дата прибытия»)
11. Список билетов на поезд (в одной строке вводить: «Номер вагона, Номер места»)
12. База автомобилей (в одной строке вводить: «Номер машины, Марка машины»)

Требования к оформлению отчета

1 Формат листа – А4 (210 х 297 мм). Поля: слева – 25 мм; справа – 15 мм; сверху, снизу – 20 мм.

2 Основной текст документа должен быть напечатан шрифтом Times New Roman Cyr, размером 12 пт. Межстрочный интервал – 1.5, отступ красной строки – 1.25. Не добавлять разрыв между абзацами одного стиля. Выравнивание по ширине.

3 Размер заголовков – 12 пт. Заголовок первого уровня – Все заглавные, жирный, заголовок второго и последующих уровне – жирные, как в предложении. Отступ заголовка – 1.25. Выравнивание по левому краю.

4 Нумерация страниц производится автоматически.

5 Если позволяет структура документа, автоматически оформить оглавление.

6 Титульный лист оформлен по стандарту ТПУ.

7 Рекомендуемая структура отчета: Титульный лист, Оглавление (если требуется) Введение, Ход работы, Заключение. В разделе ход работы описывается последовательность выполнения заданий, если надо приводятся участки кода и скриншоты.

Требования к защите

Лабораторная считается защищенной при наличии трех файлов:

1. Архив с исходным кодом
2. Исполняемый файл
3. Отчет по лабораторной работе