МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

Факультет *компьютерных наук*

*Тема курсовой работы*

C# Программирование на языке высокого уровня. Среда разработки Microsoft Visual Studio.NET

*Курсовая работа*

230200 Информационные системы

Информационные системы и сетевые технологии

Студент *М.Б. Леонтьев, 1 курс, д/о*

Воронеж 2015

Оглавление

[Оглавление 2](#_Toc437451424)

[Введение 4](#_Toc437451425)

[1.1 Платформа .NET 4](#_Toc437451426)

[1.2 Среда Visual Studio.NET 11](#_Toc437451427)

[1.3 Заготовка консольной программы 13](#_Toc437451428)

[1.4 Запуск программы 16](#_Toc437451429)

[2 Приложения C# для расчетов по формулам, консольный ввод-вывод. 16](#_Toc437451430)

[2.1 Структура приложения на языке С#. 18](#_Toc437451431)

[2.2 Константы 19](#_Toc437451432)

[2.3 Переменные 20](#_Toc437451433)

[2.4 Математические функции 22](#_Toc437451434)

[2.5 Организация ввода-вывода данных. 24](#_Toc437451435)

[2.6 Ввод данных 24](#_Toc437451436)

[2.7 Вывод данных 25](#_Toc437451437)

[3 Ввод-вывод информации, с использованием файлов. Форматирование значений данных. 27](#_Toc437451438)

[3.1 Управление форматом числовых данных 27](#_Toc437451439)

[3.2 Организация ввода вывода с использованием файлов 31](#_Toc437451440)

[4 Управляющие операторы условного и безусловного переходов. Разветвляющиеся программы 34](#_Toc437451441)

[4.1 Оператор условного перехода if и его конструкции 35](#_Toc437451442)

[4.2 Конструкция if-else (если - иначе) 36](#_Toc437451443)

[4.3 Вложенные конструкции оператора if 37](#_Toc437451444)

[4.4 Операторы логического сравнения 39](#_Toc437451445)

[4.5 Логические операторы 39](#_Toc437451446)

[4.6 Оператор проверки 40](#_Toc437451447)

[4.7 Оператор безусловного перехода goto 40](#_Toc437451448)

[4.8 Конструкция switch (переключатель) 41](#_Toc437451449)

[4.9 Разветвляющиеся программы 43](#_Toc437451450)

[5 Управляющие операторы для организации циклов. Программы с циклами 44](#_Toc437451451)

[5.1 Оператор цикла for 44](#_Toc437451452)

[5.2 Оператор цикла while 46](#_Toc437451453)

[5.3 Оператор цикла do-while 47](#_Toc437451454)

[5.4 Операторы break и continue 48](#_Toc437451455)

[5.5 Вложенные циклы 48](#_Toc437451456)

[5.6 Суммирование рядов 49](#_Toc437451457)

[6 Одномерные массивы 50](#_Toc437451458)

[6.1 Инициализация массивов 53](#_Toc437451459)

[6.2 Ввод-вывод массивов 54](#_Toc437451460)

[7 Многомерные массивы 64](#_Toc437451461)

[8 Классы и методы 78](#_Toc437451462)

[8.1 Методы 80](#_Toc437451463)

[8.2 Объявление методов 80](#_Toc437451464)

[8.3 Вызов методов. 82](#_Toc437451465)

[Литература 86](#_Toc437451466)

Введение

* 1. Платформа .NET

Программист пишет программу, компьютер ее выполняет. Программа создается на языке, понятном человеку, а компьютер умеет исполнять только программы, написанные на его языке - в машинных кодах.

Совокупность средств, с помощью которых программы пишут, корректируют, преобразуют в машинные коды, отлаживают и запускают, называют средой разработки. Среда разработки обычно содержит:

текстовый редактор, предназначенный для ввода и корректировки текста программы;

компилятор, с помощью которого программа переводится с языка, на котором она написана, в машинные коды;

средства отладки и запуска программ;

общие библиотеки, содержащие многократно используемые элементы программ;

справочную систему и другие элементы.

Платформа .NET (произносится «дот нэт») включает не только среду разработки для нескольких языков программирования, называемую Visual Studio.NET, но и множество других средств, например, механизмы поддержки баз данных, электронной почты и коммерции необходимых для интенсификации труда программиста.

Важнейшими задачами при создании программ в настоящее время становятся:

переносимость - возможность выполнения на различных типах компьютеров;

безопасность - невозможность несанкционированных действий;

надежность - способность выполнять необходимые действия в определённых условиях.

использование готовых компонентов - для ускорения разработки;

межъязыковое взаимодействие - возможность применять одновременно несколько языков программирования. Платформа .NET позволяет успешно решать все эти задачи.

Для обеспечения переносимости компиляторы, входящие в состав платформы, переводят программу не в машинные коды, а в промежуточный язык MSIL (Microsoft Intermediate Language), или просто IL), который не содержит команд, зависящих от языка, операционной системы и типа компьютера. Программа на этом языке выполняется не самостоятельно, а под управлением системы, которая называется общеязыковой средой выполнения (Common Language Runtime, CLR).

Среда CLR может быть реализована для любой операционной системы. При выполнении программы CLR вызывает так называемый JIT-компилятор, переводящий код с языка IL в машинные команды конкретного процессора, которые немедленно выполняются. JIT означает «just in time», что можно перевести как «во время», то есть компилируются только те части программы, которые требуется выполнить в данный момент. Каждая часть программы компилируется один раз и сохраняется в памяти для дальнейшего использования.

Схема выполнения программы при использовании платформы .NET приведена на рис. 1.

Початковий текст програми

Компілятор

Виконуваний файл

(IL або метадані)

Базові класи

(IL або метадані)

Завантажувач класів

ЛТ компілятор

Виконуваний код

Виконання програми

Виклики не скомпільованих даних

1. Схема выполнения программы в .NET

Компилятор в качестве результата своего выполнения создает так называемую сборку - файл с расширением ехе или dll, который содержит код на языке IL и метаданные. Метаданные представляют собой сведения об объектах, используемых в программе, а также сведения о самой сборке. Они позволяют организовать межъязыковое взаимодействие, обеспечивают безопасность и облегчают развертывание приложений, то есть установку программ на компьютеры пользователей.

Во время работы программы среда CLR следит за тем, чтобы выполнялись только разрешенные операции, осуществляет распределение и очистку памяти и обрабатывает возникающие ошибки. Это многократно повышает безопасность и надежность программ.

Платформа .NET содержит огромную библиотеку классов, которые можно использовать при программировании на любом языке .NET. Её изучение - трудоёмкая, но необходимая задача.

Платформа .NET рассчитана на объектно-ориентированную технологию создания программ, поэтому прежде чем начинать изучение языка С#, необходимо познакомиться с основными понятиями объектно-ориентированного программирования (ООП).

Принципы ООП проще всего понять на примере программ моделирования. В реальном мире каждый предмет или процесс обладает набором статических и динамических характеристик, иными словами, свойствами и поведением. Поведение объекта зависит от его состояния и внешних воздействий. Например, объект «автомобиль» никуда не поедет, если в баке нет бензина, а если повернуть руль, изменится положение колес. Объект представляется как совокупность данных, характеризующих его состояние, и функций их обработки, моделирующих его поведение. Вызов функции на выполнение часто называют посылкой сообщения объекту.

При создании объектно-ориентированной программы предметная область представляется в виде совокупности объектов. Выполнение программы состоит в том, что объекты обмениваются сообщениями. Это позволяет использовать при программировании понятия, более адекватно отражающие предметную область.

Для моделирования реального объекта с помощью программного обеспечения выделяют его существенные особенности. Их набор зависит от цели моделирования. Выделение существенных с той или иной точки зрения свойств называется абстрагированием. Таким образом, программный объект - это абстракция.

Важным свойством объекта является его обособленность. Детали реализации объекта, то есть внутренние структуры данных и алгоритмы их обработки, скрыты от пользователя объекта и недоступны для непреднамеренных изменений. Объект используется через его интерфейс - совокупность правил доступа.

Скрытие деталей реализации называется инкапсуляцией (от слова «капсула»). Таким образом, объект является «черным ящиком», замкнутым по отношению к внешнему миру. Это позволяет представить программу в укрупненном виде - на уровне объектов и их взаимосвязей, а следовательно, управлять большим объемом информации и успешно отлаживать сложные программы.

Сказанное можно сформулировать более кратко и строго: объект - это инкапсулированная абстракция с четко определенным интерфейсом.

Инкапсуляция позволяет изменить реализацию объекта без модификации основной части программы, если его интерфейс остался прежним. Простота модификации является очень важным критерием качества программы, ведь любой программный продукт в течение своего жизненного цикла претерпевает множество изменений и дополнений. Кроме того, инкапсуляция позволяет использовать объект в другом окружении и быть уверенным, что он не испортит не принадлежащие ему области памяти, а также создавать библиотеки объектов для применения во многих программах.

В мире пишется огромное количество новых программ, и важнейшее значение приобретает возможность многократного использования кода. Преимущество объектно-ориентированного программирования состоит в том, что для объекта можно определить наследников, корректирующих или дополняющих его поведение. При этом нет необходимости не только повторять исходный код родительского объекта, но даже иметь к нему доступ.

Наследование является мощнейшим инструментом ООП и применяется для следующих взаимосвязанных целей:

исключения из программы повторяющихся фрагментов кода;

упрощения модификации программы;

упрощения создания новых программ на основе существующих.

Благодаря наследованию появляется возможность использовать объекты, исходный код которых недоступен, но в которые требуется внести изменения. Наследование позволяет создавать иерархии объектов. Иерархия представляется в виде дерева, в котором более общие объекты располагаются ближе к корню, а более специализированные - на ветвях и листьях. Наследование облегчает использование библиотек объектов, поскольку программист может взять за основу объекты, разработанные кем-то другим, и создать наследников с требуемыми свойствами.

Объект, на основании которого строится новый объект, называется родительским объектом, объектом-предком, базовым классом, или суперклассом, а унаследованный от него объект - потомком, подклассом, или производным классом.

ООП позволяет писать гибкие, расширяемые и читабельные программы. Во многом это обеспечивается благодаря полиморфизму, под которым понимается возможность во время выполнения программы с помощью одного и того же имени выполнять разные действия или обращаться к объектам разного типа.

Итак, объект это «инкапсуляция множества операций (методов), доступных для внешних вызовов, и состояния, запоминающего результаты выполнения указанных операций».

Достоинства ООП:

использование при программировании понятий, близких к предметной области;

возможность успешно управлять большими объемами исходного кода благодаря инкапсуляции, то есть скрытию деталей реализации объектов и упрощению структуры программы;

возможность многократного использования кода за счет наследования;

сравнительно простая возможность модификации программ;

возможность создания и использования библиотек объектов.

Однако создание объектно-ориентированной программы представляет собой весьма непростую задачу, поскольку требует разработки иерархии объектов, а плохо спроектированная иерархия может свести к нулю все преимущества объектно-ориентированного подхода. Кроме того, идеи ООП не просты для понимания и в особенности для практического применения. Чтобы эффективно использовать готовые объекты из библиотек, необходимо освоить большой объем достаточно сложной информации. Неграмотное же применение ООП способно привести к созданию излишне сложных программ, которые невозможно отлаживать и усовершенствовать.

Для представления объектов в языках С#, Java, С++, Delphi и др. используется понятие класс, аналогичное обыденному смыслу этого слова в контексте «класс членистоногих», «класс млекопитающих», «класс задач» и т. п. Класс является обобщенным понятием, определяющим характеристики и поведение некоторого множества конкретных объектов этого класса, называемых экземплярами класса. В последнее время в класс часто добавляется третья составляющая - события, на которые может реагировать объект класса.

Все классы библиотеки .NET, а также все классы, которые создает программист в среде .NET, имеют одного общего предка - класс object и организованы в единую иерархическую структуру. Внутри нее классы логически сгруппированы в так называемые пространства имен, которые служат для упорядочивания имен. Пространства имен могут быть вложенными, их идея аналогична знакомой вам иерархической структуре каталогов на компьютере.

Обычно в одно пространство имен объединяют взаимосвязанные классы. Например, пространство System.Net содержит классы, относящиеся к передаче данных по сети, System.Windows . Forms - элементы графического интерфейса пользователя, такие как формы, кнопки и т. д. Имя каждого пространства имен представляет собой неделимую сущность, однозначно его определяющую.

* 1. Среда Visual Studio.NET

Среда разработки Visual Studio.NET предоставляет мощные и удобные средства написания, корректировки, компиляции, отладки и запуска приложений, использующих .NET-совместимые языки. Корпорация Microsoft включила в платформу средства разработки для четырех языков: С#, VB.NET, С++ и J#.

Платформа .NET является открытой средой. Это значит, что компиляторы для нее могут поставляться и сторонними разработчиками. К настоящему времени разработаны десятки компиляторов для .NET, например, Ada, COBOL, Delphi, Eiffel, Fortran, Lisp, Obe- ron, Perl и Python.

Приложение в процессе разработки называется проектом. Проект объединяет все необходимое для создания приложения: файлы, папки, ссылки и прочие ресурсы. Среда Visual Studio.NET позволяет создавать проекты различных типов, например:

Windows-приложение использует элементы интерфейса Windows, включая формы, кнопки, флажки и пр.;

консольное приложение выполняет вывод «на консоль», то есть в окно командного процессора;

библиотека классов объединяет классы, которые предназначены для использования в других приложениях;

веб-сервис - компонент, методы которого могут вызываться через Интернет.

Консольные приложения наилучшим образом подходят для изучения языка, так как в них не используется множество стандартных объектов, необходимых для создания графического интерфейса.

Знакомство со средой Microsoft Visual Studio 2010 (язык С#).

Microsoft Visual Studio 2010 доступна в следующих вариантах:

Express - бесплатная среда разработки, включающая только базовый набор возможностей и библиотек;

Professional - поставка, ориентированная на профессиональное создание программного обеспечения, и командную разработку, при которой созданием программы одновременно занимаются несколько человек;

Premium - издание, включающее дополнительные инструменты для работы и исходным кодом программ и создания баз данных;

Ultimate - наиболее полное издание Visual Studio, включающие все доступные инструменты для написания, тестирования, отладки и анализа программ, а также дополнительные инструменты для работы с базами данных и проектирования архитектуры ПО.

Отличительной особенностью среды Microsoft Visual Studio 2010 является то, что она поддерживает работу с несколькими языками программирования и программными платформами. Поэтому, перед тем, как начать создание программы на языке С#, необходимо выполнить несколько подготовительных шагов по созданию проекта и выбора и настройки компилятора языка С# для трансляции исходного кода.

После щелчка на кнопке OK среда создаст решение и проект с указанным именем.

В верхней части экрана располагается главное меню (с разделами Файл, Правка, Вид и т. д.) и панели инструментов. Панелей инструментов в среде великое множество, и если включить их все, они займут половину экрана. При создании проекта необходимо указать язык C# и тип проекта.

В верхней правой части экрана располагается окно управления проектом. Обозреватель решении (если оно не отображается, следует воспользоваться командой Вид • Обозреватель решении главного меню). В окне перечислены все ресурсы, входящие в проект: ссылки на библиотеку, и информация о сборке и файл с исходным текстом класса (Classl.cs). В этом же окне можно увидеть и другую информацию, если перейти на вкладку Командный обозреватель, ярлычок которой находится в верхней части окна. На этой вкладке представлен список всех классов, входящих в приложение, их элементов и предков. С помощью проводника Windows можно увидеть какие файлы создала среда для поддержки проекта. На заданном диске появилась папка с указанным именем, содержащая несколько других файлов и вложенных папок. Среди них - файл проекта (с расширением csproj), файл решения (с расширением sln) и файл с кодом класса (Classl.cs).

Основное пространство экрана занимает окно редактора, в котором располагается текст программы, созданный средой автоматически. Текст представляет собой каркас, в который программист добавляет код по мере необходимости. Ключевые (зарезервированные) слова отображаются синим цветом, комментарии (2 различных типов) - серым и темно-зеленым, остальной текст - черным.

Слева от текста находятся символы структуры: щелкнув на любом квадратике с минусом, можно скрыть соответствующий блок кода. При этом минус превращается в плюс, щелкнув на котором, можно опять вывести блок на экран. Это средство хорошо визуально структурирует код и позволяет сфокусировать внимание на нужных фрагментах.

* 1. Заготовка консольной программы

Рассмотрим каждую строку заготовки программы (пример 1). Не надо пытаться сразу понять абсолютно все, что в ней написано. Цель - изучить принципы работы в оболочке, а не досконально разобраться в программе.

Пример 1. Заготовка консольной программы

using System;

using System.Collections.Generic;

using System.Linq; using System.Text;

namespace ConsoleApplication1

{

/// <summary>

/// Summary description for Classl. /// </summary>

class Classl

{

/// <summary>

/// The main entry point for the application. /// </summary>

static void Main(string[] args)

{

//

// TODO: Add code to start application

// here //

}

}

}

Директива using System разрешает использовать имена стандартных классов из пространства имен System непосредственно (без указания имени пространства).

Директивы :

using System.Collections.Generic;

using System.Linq;

using System.Text;

относятся к общему случаю и для создания консольного приложения не нужны. Поэтому в дальнейшем они использоваться не будут.

Ключевое слово namespace создает для проекта собственное пространство имен, названное по умолчанию ConsoleApplicationl. Это сделано для того, чтобы можно было давать программным объектам имена, не заботясь о том, что они могут совпасть с именами в других пространствах имен.

Строки, начинающиеся с двух или трех косых черт, являются комментариями и предназначены для документирования текста программы.

С# - объектно-ориентированный язык, поэтому написанная на нем программа представляет собой совокупность взаимодействующих между собой классов.

В нашей заготовке программы всего один класс, которому по умолчанию задано имя Class. Описание класса начинается с ключевого слова class, за которым следуют его имя и далее в фигурных скобках - список элементов класса (его данных и функций, называемых также методами).

ВНИМАНИЕ!!! Фигурные скобки являются важным элементом синтаксиса. Каждой открывающей скобке соответствует своя закрывающая, которая обычно располагается ниже по тексту с тем же отступом. Эти скобки ограничивают блок, внутри которого могут располагаться другие блоки, вложенные в него, как матрешки. Блок может применяться в любом месте, где допускается отдельный оператор.

В данном случае внутри класса только один элемент - метод Main. Каждое приложение должно содержать метод Main - с него начинается выполнение программы. Все методы описываются по единым правилам.

Упрощенный синтаксис метода:

[ спецификаторы ] тип имя\_метода ( [ параметры ] )

{

тело метода: действия, выполняемые методом

}

Наряду с понятием «метод» часто используется другое - функция-член класса. Метод является частным случаем функции - законченного фрагмента кода, который можно вызвать по имени.

Среда поместила внутрь метода Main комментарий:

// TODO: Add code to start application here

Это означает: «Добавьте сюда свой код, выполняемый при запуске приложения». Добавим после строк комментария (но не в той же строке!) строку

Console.WriteLine("BceM привет!!!");

Здесь Console - это имя стандартного класса из пространства имен System. Его метод WriteLine выводит на экран заданный в кавычках текст. Как видите, для обращения к методу класса используется конструкция

имя\_класса.имя\_метода.

Если вы не сделали ошибку в первом же слове, то сразу после ввода с клавиатуры следом за словом Console символа точки среда выведет подсказку, содержащую список всех доступных элементов класса Console. Выбор нужного имени выполняется либо мышью, либо клавишами управления курсором, либо вводом одного или нескольких начальных символов имени. При нажатии клавиши Enter выбранное имя появляется в тексте программы.

Не стоит пренебрегать возможностью автоматического ввода - это убережет от опечаток и сэкономит время. Если подсказка не появляется, это свидетельствует об ошибке в имени или в месте рас - положения в программе вводимого текста.

Программа должна приобрести вид, приведенный в Примере 2. Обратите внимание на то, что после внесения изменений около имени файла на ярлычке в верхней части окна редактора появился символ \* - это означает, что текст, сохраненный на диске, и текст, представленный в окне редактора, не совпадают. Для сохранения файла воспользуйтесь командой Файл • Сохранить главного меню или кнопкой Save на панели инструментов (текстовый курсор должен при этом находиться в окне редактора). Впрочем, при выходе из программы среда сохранит исходный текст самостоятельно.

* 1. Запуск программы

Самый простой способ запустить программу - нажать клавишу F5 (или выбрать в меню команду Отладка • Начать отладку). Если программа написана без ошибок, то фраза

Всем привет! ! !

промелькнет перед вашими глазами в консольном окне, которое незамедлительно закроется. Это хороший результат, но для того чтобы пронаблюдать его спокойно, следует воспользоваться клавишами Ctrl+F5.

После внесения изменений компилятор может обнаружить в тексте программы синтаксические ошибки. Об этом он сообщает в окне, расположенном в нижней части экрана.

1. Приложения C# для расчетов по формулам, консольный ввод-вывод.

Язык C# имеет набор встроенных типов, которые рассматриваются как псевдонимы типов в пространстве имен System. Например, тип string - это псевдоним типа System.String, а тип int - псевдоним типа System.Int32. Все встроенные типы подразделены на группы: целочисленные типы; вещественные типы; логический тип; символьные типы; объектный тип (object). Описание типов приведено в таблице 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип даних | Ключове слово | Псевдонім класу бібліотеки NET | Опис | Розмір (байт) |
| Логічний | bool | System.Boolean | - | - |
| Цілий | Int  Shot  Byte  Sbyte  Long | System.Int32  System.Int16  System.Byte  System.SByte  System.Int64 | зі знаком  зі знаком  без знаку  зі знаком  зі знаком | 32  16  8  8  64 |
| Дійсний | Float  Double | System.Single  System.Double | 7 цифр  15 цифр | 32  64 |
| Рядковий символьний | String  Char | System.String  System. Char | Рядок символів Unicode | -  16 |
| Будь-який тип | object | System.Object | Об’єктний | - |

1. Типы данных

Иерархия классов NET Framework имеет один общий корень - класс System.Object. Все типы разделяются на две категории: размерные типы и ссылочные типы.

При создании переменной размерного типа под нее в стеке выделяется определенный объем памяти, соответствующий типу этой переменной. При передаче такой переменной в качестве параметра выполняется передача значения, а не ссылки на него. Значение размерного типа не может быть равным null. К размерным типам, например, относятся целочисленные и вещественные типы, структуры.

При создании переменной ссылочного типа память под созданный объект выделяется в другой области памяти, называемой кучей. Ссылка всегда указывает на объект заданного типа.

* 1. Структура приложения на языке С#.

Проектом называется совокупность файлов, содержащих информацию об установках, конфигурации, ресурсах проекта, а также файлов исходного кода и заголовочных файлов.

Интегрированная среда проектирования Visual Studio позволяет для создания проектов на разных языках программирования использовать различные инструментальные средства проектирования (например, Microsoft Visual Basic, Microsoft Visual C#).

Любое приложение на языке C#, разрабатываемое в среде проектирования Visual Studio, реализуется как отдельный проект. Приложение на языке С# может состоять из нескольких модулей. Каждый модуль C# может содержать код нескольких классов (при создании приложения в среде Visual Studio.NET каждый класс С# автоматически помещается в отдельный модуль - файл с расширением cs).

Для консольного приложения один из классов, реализуемых модулем, должен содержать метод Main. В языке C# нет аппарата заголовочных файлов, используемого в языке С++, поэтому код модуля должен содержать как объявление, так и реализацию класса. По умолчанию весь код класса, представляющего консольное приложение, заключается в одно пространство имен, одноименное с именем приложения.

Точкой входа в программу на языке C# является метод Main. Этот метод может записываться как без параметров, так и с одним параметром типа string - указателем на массив строк, который содержит значения параметров, введенных при запуске программы. В отличие от списка параметров, задаваемых при запуске С-прило- жения, список параметров С#-приложения не содержит в качестве первого параметра имя самого приложения. Код метода указывается внутри фигурных скобок:

static void Main(string[] args)

{

…

}

Ключевое слово static определяет, что метод Main является статическим методом, вызываемым без создания экземпляра объекта типа класса, в котором этот метод определен. Метод, не возвращающий никакого значения, указывается с ключевым словом void. Однако метод Main может возвращать значение типа int.

Пример 1. Вывод сообщения на консоль.

static void Main()

{

Console.WriteLine("Уpa!\n Сегодня \"Информатика\"!!!");

}

Замечание. Для отладки можно использовать команду меню Debug\Start Without Debugging. На экране появится окно с результатом исполнения. Обратите внимание на надпись в конце программы: Press any key to continue, которая не была предусмотрена. При нажатии любой клавиши окно закрывается. Это результат срабатывания встроенной разработчиками компилятора функции «остановки экрана» для того, чтобы можно было бы сколь угодно долго его рассматривать.

Можно использовать команду Debug\Start Debugging, но тогда окно закроется и мы не сможем рассмотреть искомый результат. Для того чтобы обойти это неудобство, следует при разработке программы предусмотреть собственную остановку экрана. Для этого используется команда Console.Read();

* 1. Константы

Это неизменяемые в процессе выполнения программы величины.

Целые константы - наиболее распространенный тип int. Это целое число, которое может быть отрицательным, положительным или нулем -12, 5, 0 (все целые со знаком 32 бита). Их можно записывать с суффиксом -12L (длинное целое 64 бита), 5u (целое без знака 8 бит)

Вещественные константы с фиксированной точкой. При записи константы типа float(32 бита) необходимо, чтобы за значением шел суффикс символ f или F 1.2, -1.234, при записи константы типа double (64 бита) можно записать суффикс «d» или «D», но это не является обязательным условием: 1234.5678, 12.3d. Дробная часть отделяется от целой части точкой.

Вещественные константы с плавающей точкой. При записи константы типа float(32 бита) необходимо, чтобы за значением шел суффикс символ f или F: 1.2E-3f (число 0.0012), при записи константы типа double (64 бита) -1.34E5 (число -134000) наличие суффикса не требуется.

Символьные константы. Символьная константа char может представлять собой 16-битный символ Unicode ('a') или управляющие символы (возврат каретки ('\r'), перевод страницы ('\f'), горизонтальную табуляцию ('\t'), и другие), заключенный в апострофы.

Строковые константы - это последовательность символов, заключенная в кавычки, или константы string. Строка, состоящая из символов, например "Уpa!\n Сегодня \"Информатика\"!!!"

Логическая константа. Задается одним из двух значений true («истина») или false («ложь»). Используется в C# в логических выражениях, операторах условного перехода.

Именованные константы. Применяются для того, чтобы вместо значений констант, использовать в программе их имена, например константа р вещественная одинарной точности

const float p = 3.14159f

* 1. Переменные

Переменная - именованная область памяти, для хранения данных определенного типа. При выполнении программы значение переменной величины можно изменять. Все переменные должны быть описаны явно, при описании переменной задается ее значение и тип. При объявлении переменной может быть задано начальное значение.

Имя переменной может содержать буквы, цифры и символ подчеркивания. Прописные и строчные буквы различаются. Например, переменные Long, LONG, long - три разных переменные.

Имя переменной может начинаться с буквы или знака подчеркивания, но не цифры. Имя переменной не должно совпадать с ключевыми словами. Не рекомендуется начинать имя с двух подчеркиваний (такие имена зарезервированы для служебного использования).

Правильные имена переменных: MaxLen, iMaxLen, Max\_Len

Неправильные имена переменных: 2Len, Le#

Примеры описания переменных:

int a = -14; // числовая целая 32 бита

float c = -0.00151f; // числовая вещественная 32

// бита

double i = 1234.56789; // числовая вещественная 64

// бита

bool l = false; // логическая 16 бит

string name = "Petrov"; // строковая

Выражение - состоит из одного или более операндов (которые могут быть переменными, константами, функциями или символьными значениями), знаков операций и круглых скобок.

Примеры выражений:

2 \* 2 + 1 полученное значение 5

1 / 2 - 3 полученное значение -3

1 / 2 - 3 полученное значение -2.5

Присвоение значения переменной представляет оператор присваивания (знаки основных операций приведены в таблице 2) : y = 2\*x\*x + 3\*x - 1.

В этом примере сначала производятся вычисления правой части оператора присваивания « = », а затем полученное значение присваивается переменной у. Для текстовых данных выражение можно записать в следующем виде:

string kaf = "Кафедра" + "ПМиСИ";

В этом примере строки по правую сторону от оператора присваивания объединяются, чтобы получить строку "Кафедра + ПМиСИ", которая затем присваивается переменной kaf.

|  |  |
| --- | --- |
| Знак операции | Название |
| + | Сложение |
| - | Вычитание |
| \* | Умножение |
| / | Деление |
| % | Остаток от деления |

1. Знаки операций

Если в арифметических выражениях используются целые числа, то результатом вычислений будет целое число, и любой остаток от деления будет отброшен. Для получения остатка можно использовать соответствующую операцию %, например 10 % 3 возвращает остаток от целочисленного деления, равный 1.

Когда в арифметических выражениях используются числа с плавающей точкой, то результатом деления 10f / 3f будет число 3,333333.

* 1. Математические функции

С# содержит большое количество встроенных математических функций, которые реализованы в классе Math пространства имен System.

Рассмотрим краткое описание некоторых математических функций, подробнее с ними можно познакомиться в справочной системе VS или технической документации. Особое внимание следует обратить на типы операндов и результатов, т. к. каждая функция может иметь несколько перегруженных версий.

Замечание. Использование нескольких функций с одним и тем же именем, но с различными типами параметров, называется перегрузкой функции. Например, функция Math.Abs(), вычисляющая модуль числа, имеет 7 перегруженных версий: double Math.Abs (double x), float Math.Abs (float x), int Math.Abs(int x), и т. д. (таблица 3)

|  |  |  |
| --- | --- | --- |
| № | Название | Описание |
| 1. | Math.Abs(вираз) | Модуль |
| 2. | Math.Ceiling(вираз) | Округление до большего целого |
| 3. | Math.Cos ( вираз ) | Косинус |
| 4. | Math.E | Число е |
| 5. | Math.Exp(вираз) | Экспонента |
| 6. | Math.Floor(вираз) | Округление до меньшего целого |
| 7. | Math.Log(вираз) | Натуральный логарифм |
| 8. | Math.Log10(вираз) | Десятичный логарифм |
| 9. | Math.Max(вираз1, вираз2) | Максимум ив двух |
|  |  | значений |
| 10. | Math.Min (вираз1, вираз2) | Минимум из двух значений |
| 11. | Math.PI | Число |
| 12. | Math.Pow(вираз1, вираз2) | Возведение в степень |
| 13. | Math.Roundl(вираз) | Простое округление |
|  | Math.Round(вираз,число) | Округление до заданного |
|  |  | числа цифр |
| 14. | Math.Sign(вираз) | Знак числа |
| 15. | Math.Sin(вираз) | Синус |
| 16. | Math.Sqrt(вираз) | Квадратный корень |
| 17. | Math.Tan (вираз) | Тангенс |

1. Математические функции

Пример 2. Вычислить значения функции при х = 2,5

using System;

using System.Collections.Generic;

using System.Linq; using System.Text;

namespace ConsoleApplication1

{

class Example2 // начало описания класса

// Example2

{

static void Main()

{

double p = 3.14159; double x = 2.5;

double y = Math.Cos(p \* x)/(1 + x\*x);

Console.WriteLine();

Console.WriteLine(" x = {0} \t y = {1} ",x, y);

}

}

}

Замечание. Функция Console.WriteLine() выводит на экран пустую строку. Это сделано для более комфортной работы

* 1. Организация ввода-вывода данных.

Программа при вводе данных и выводе результатов взаимодействует с внешними устройствами. Совокупность стандартных устройств ввода (клавиатура) и вывода (экран) называется консолью. В языке С# нет операторов ввода и вывода. Вместо них для обмена данными с внешними устройствами используются специальные объекты. В частности, для работы с консолью используется стандартный класс Console, определенный в пространстве имен System.

* 1. Ввод данных

Для ввода данных обычно используется метод ReadLine, реализованный в классе Console. Особенностью данного метода является то, что в качестве результата он возвращает строку (string).

Пример:

static void Main()

{

string s = Console.ReadLine(); Console.WriteLine(s);

}

Для того чтобы получить числовое значение необходимо вое - пользоваться преобразованием данных.

Пример:

static void Main()

{

string s = Console.ReadLine();

int x = int.Parse(s); // преобразование строки в

// число

Console.WriteLine(x);

}

Или сокращенный вариант:

static void Main()

{

//преобразование введенной строки в число

int x = int.Parse(Console.ReadLine());

Console.WriteLine(x);

}

Для преобразования строкового представления целого числа в тип int мы используем метод int.Parse(), который реализован для всех числовых типов данных. Таким образом, если нам потребуется преобразовать строковое представление в вещественное, мы можем воспользоваться методом float.Parse() или double. Parse(). В случае, если соответствующее преобразование выполнить невозможно, то выполнение программы прерывается и генерируется исключение System.FormatExeption (входная строка имела неверный формат).

* 1. Вывод данных

В приведенных выше примерах мы уже рассматривали метод WriteLine, реализованный в классе Console, который позволяет организовывать вывод данных на экран. Однако существует несколько способов применения данного метода:

|  |  |
| --- | --- |
| Console.WriteLine(x); | на экран выводится значение идентификатора х |
| Console.WriteLine("x=" + x +"y=" + y); | на экран выводится строка, образованная последовательным слиянием строки "х=", значения х, строки "у=" и значения у |
| Console.WriteLine ("x={0} y={1}", x, y); | на экран выводится строка, формат которой задан первым аргументом метода, при этом вместо параметра {0} выводится значение х, а вместо {1} - значение |

1. Способы вывода

программа формула цикл форматирование

Если использовать при выводе вместо метода WriteLine метод Write, вывод будет выполняться без перевода строки.

|  |  |  |  |
| --- | --- | --- | --- |
| \n | Перевод строки | \’ | Апостроф |
| \r | Возврат каретки | \” | Кавычки |

1. Использование управляющих последовательностей.

|  |  |  |  |
| --- | --- | --- | --- |
| Вид | Наименование | Вид | Наименование |
| \a | Звуковой сигнал | \t | Горизонтальная табуляция |
| \b | Возврат на шаг назад | \v | Вертикальная табуляция |
| \f | Перевод страницы | \\ | Обратная косая черта |

1. Управляющие символы

Управляющей последовательностью называют определенный символ, предваряемый обратной косой чертой. Данная совокупность символов интерпретируется как одиночный символ и используется для представления кодов символов, не имеющих графического обозначения (например, символа перевода курсора на новую строку) или символов, имеющих специальное обозначение в символьных и строковых константах (например, апостроф).

Пример 3. Вывести сообщение о версии установленной операционной системы, текущую дату и время.

using System;

using System.Collections.Generic;

using System.Linq; using System.Text;

namespace ConsoleApplicationl

{

class Program

{

static void Main(string[] args)

{

// вывести версию операционной системы

OperatingSystem os = System.Environment.OSVersion;

Console.WriteLine("Platform: {0}",os.Platform);

System.Console.WriteLine("The current date and time is

" + System.DateTime.Now);

// дата и время System.Console.ReadLine();

}

}

}

Пример 4. Использование консольного ввода для вычисления значений функции

using System;

using System.Collections.Generic;

using System.Linq; using System.Text;

namespace lab0

{

class Program

{

static void Main(string[] args)

{

System.Console.WriteLine("The current date and time is

" + System.DateTime.Now);

double pi = 3.14159;

Console.WriteLine("Input x =\r");

double x = Convert.ToDouble(Console.ReadLine());

double y = Math.Cos(pi \* x)/(1 + x\*x);

Console.WriteLine(" x = {0} \t y = {1} ",x,y);

Console.ReadKey();

}

}

}

1. Ввод-вывод информации, с использованием файлов. Форматирование значений данных.

Вывод, производимый методами System.Console.Write() и System.Console.WriteLine(), можно форматировать. Форматирование позволяет указывать формат целых чисел, чисел с плавающей точкой и других типов данных.

* 1. Управление форматом числовых данных

Пусть в программе определена переменная типа int с именем

value:

int value = 250;

До этого момента переменные выводились следующим образом:

System.Console.WriteLine ("value =" + value);

Результат вывода:

value = 250

Можно вывести значение value, используя требуемое число позиций (например 5):

System.Console.WriteLine("value = {0, 5}", value);

Первое число в фигурных скобках означает номер переменной - это 0, что соответствует первой переменной value в списке метода System.Console.WriteLine(). Второе число в фигурных скобках означает количество позиций, отведенное для отображения переменной. В данном примере оно равно 5. При выводе переменной длина ее представления будет дополнена пробелами слева. Если количество позиций меньше чем число знаков переменной, то оно будет выведено без форматирования.

Можно задать форматирование для вывода каждой переменной:

int a = -12;

int b = 20;

System.Console.WriteLine("a = {0, 4}, b = {1, 3}", a, b);

Результат вывода:

a = -12, b = 20

Форматированный вывод чисел с плавающей точкой немного более сложный. Предположим, определена переменная типа double с именем

myDouble:

double myDouble = 1234.56789;

Следующий пример выводит значения myDouble, отведя под него десять знакомест, и округлив его до трех цифр после запятой:

System.Console.WriteLine("myDouble = {0, 10:f3}"; myDouble);

Символы f3 в этом примере означают, что значение выводится как число с плавающей точкой (символ f) , в дробной части будет выведено три цифры.

Точно такое же форматирование можно применять для типов float и decimal. Например:

float myFloat = 1234.56789f;

System.Console.Write("myFloat ={0, 0:f3}"myFloat);

decimal myDecimal = 1234.56789m;

System.Console.Write("myDecimal = {0, 10:f3}", myDecimal);

Результат вывода:

myFloat = 1234.568;

myDecimal 1234.568;

В списке аргументов методов WriteLine или Write задается строка вида {n, w: спецификатор k} - где n определяет номер идентификатора из списка аргументов метода WriteLine, спецификатор - определяет формат данных, w - целая константа без знака, задает количество символов (длину поля), a k - количество позиций для дробной части значения идентификатора.

Для каждого типа данных существует своя форма представления. Данные сведены в таблицу 7.

|  |  |  |
| --- | --- | --- |
| Тип данных | | Форма |
|  | Целые | W |
| Числовые | Вещественные с фиксированной точностью | W:Fk |
|  | Вещественные в экспоненциальном формате | W:Ek |
| Логические | | W |
| Символьные | | W |

1. Типы данных

Символы форматирования F, E - определяют тип и характеристики объектов ввода-вывода. Параметр w - целая константа без знака, задает количество символов (длину поля), отводимых для ввода- вывода объекта. Параметр k - целая константа без знака определяет для числовых данных:

количество позиций, для цифр в дробной части числа (форма F);

количество позиций для цифр, в дробной части мантиссы числа (форма E или G).

|  |  |  |
| --- | --- | --- |
| Символ | Формат | Значение |
| C или c | Денежный. По умолчанию ставит знак р. Изменить его можно с помощь объекта NumberFormatInfo | Задается количество десятичных разрядов. |
| D или d | Целочисленный (используется только с целыми числами) | Задается минимальное количество цифр. При необходимости результат дополняется начальными нулями |
| E или e | Экспоненциальное представление чисел | Задается количество символов после запятой. По умолчанию используется 6 |
| F или f | Представление чисел с фиксированной точкой | Задается количество символов после запятой |
| G или g | Общий формат (или экспоненциальный, или с фиксированной точкой) | Задается количество символов после запятой. По умолчанию выводится целая часть |
| N или n | Стандартное форматирование с использованием запятых и пробелов в качестве разделителей между разрядами | Задается количество символов после запятой. По умолчанию - 2, если число целое, то ставятся нули |
| X или x | Шестнадцатеричный формат |  |
| P или p | Процентный |  |

1. Символы форматирования

В качестве спецификаторов могут использоваться следующие значения:

Пример 1. Форматированный вывод данных различного типа.

public static void Main()

{

int a = -14;

float c = -0.00151f;

double i = 1234.56789;

bool l=false;

string name="Petrov";

System.Console.WriteLine("name = {0, 6}, l = {1, 4}", name,l);

System.Console.WriteLine("a ={0, 4}, c = {1,10:f5}, i =

{1,20:e8}", a,c,i);

System.Console.WriteLine(" ");

System.Console.WriteLine(" Для выхода нажмите на Enter");

System.Console.ReadLine();

}

Пример 2. Ввод в диалоге и форматированный вывод данных различного типа.

public static void Main()

{

int a; // = -14;

float c; // = -0.00151f;

double i; // = 1234.56789;

bool l; // = false;

string name; //="Petrov";

Console.WriteLine("Input фамилию ");

name = Console.ReadLine();

Console.WriteLine("Input a");

a = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Input c");

c = Convert.ToSingle(Console.ReadLine());

Console.WriteLine("Input i");

i = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Input l");

l = Convert.ToBoolean(Console.ReadLine());

System.Console.WriteLine(" Результаты форматирования \n

name = {0, 6}, l = {1, 4}", name,l);

System.Console.WriteLine("a ={0, 4}, c = {1,10:f5},

i = {2,20:e8}", a,c,i);

System.Console.WriteLine(" ");

System.Console.WriteLine("Для выхода нажмите на Enter");

System.Console.ReadLine();

}

* 1. Организация ввода вывода с использованием файлов

Сопрограммы выполняют операции ввода-вывода посредством потоков, которые построены на иерархии классов. Поток (stream) - это абстракция, которая генерирует и принимает данные. С помощью потока можно читать данные из различных источников (клавиатура, файл) и записывать в различные источники (принтер, экран, файл). Несмотря на то, что потоки связываются с различными физическими устройствами, характер поведения всех потоков одинаков. Поэтому классы и методы ввода-вывода можно применить ко многим типам устройств.

На самом низком уровне иерархии потоков ввода-вывода находятся потоки, оперирующие байтами. Это объясняется тем, что многие устройства при выполнении операций ввода-вывода ориентированы на байты. Однако для человека привычнее оперировать символами, поэтому разработаны символьные потоки, которые фактически представляют собой оболочки, выполняющие преобразование байтовых потоков в символьные и наоборот. Кроме этого, реализованы потоки для работы с int-, double-, short- значениями, которые также представляют оболочку для байтовых потоков, но работают не с самими значениями, а с их внутренним представлением в виде двоичных кодов.

Центральную часть потоковой С#-системы занимает класс Stream пространства имен System.IO. Класс Stream представляет байтовый поток и является базовым для всех остальных потоковых классов. Чтобы создать символьный поток нужно поместить объект класса Stream (например, FileStream) "внутрь" объекта класса StreamWriter или объекта класса StreamReader. В этом случае байтовый поток будет автоматически преобразовываться в символьный.

Класс StreamWriter предназначен для организации выходного символьного потока. Этот класс содержит несколько конструкторов. Так, например, создать экземпляр класса StreamWriter можно следующим образом:

StreamWriter fileOut = new StreamWriter(new FileStream("text.txt",

FileMode.Create, FileAccess.Write));

Эта версия конструктора позволяет ограничить доступ только чтением или только записью:

FileStream(string filename, FileMode mode, FileAccess how)

где:

параметры filename и mode имеют то же назначение, что и в предыдущей версии конструктора;

параметр how, определяет способ доступа к файлу и может принимать одно из значений, определенных перечислением FileAccess:

FileAccess.Read - только чтение;

FileAccess.Write - только запись;

FileAccess.ReadWrite - и чтение, и запись.

Другой вид конструктора позволяет открыть поток сразу через обращения к файлу:

StreamWriter(string name),

где параметр name определяет имя открываемого файла.

Например, обратиться к данному конструктору можно следующим образом:

StreamWriter fileOut = new StreamWriter("c:\temp\t.txt");

И еще один вариант конструктора StreamWriter:

StreamWriter(string name, bool appendFlag),

где параметр name определяет имя открываемого файла; параметр appendFlag может принимать значение true - если нужно добавлять данные в конец файла, или false - если файл необходимо перезаписать. Например:

StreamWriter fileOut=new StreamWriter("t.txt", true);

Теперь для записи данных в поток fileOut можно обратиться к методу WriteLine. Это можно сделать следующим образом:

fileOut.WriteLine("test");

В данном случае в конец файла t.txt будет дописано слово test.

Класс StreamReader предназначен для организации входного символьного потока. Один из его конструкторов выглядит следующим образом:

StreamReader(Stream stream),

где параметр stream определяет имя уже открытого байтового потока. Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для ввода.

Например, создать экземпляр класса StreamWriter можно следующим образом:

StreamReader fileIn = new StreamReader(new

FileStream("text.txt", FileMode.Open, FileAccess.Read));

Как и в случае с классом StreamWriter у класса StreamReader есть и другой вид конструктора, который позволяет открыть файл напрямую:

StreamReader (string name);

где параметр name определяет имя открываемого файла.

Обратиться к данному конструктору можно следующим образом:

StreamReader fileIn=new StreamReader ("z:\temp\t.txt");

В C# символы реализуются кодировкой Unicode. Для того, чтобы можно было обрабатывать текстовые файлы, содержащие русский символы, созданные, например, в Блокноте, рекомендуется вызывать следующий вид конструктора StreamReader:

StreamReader fileIn=new StreamReader ("z:\temp\t.txt",

Encoding.GetEncoding(1251));

Параметр Encoding.GetEncoding(1251) говорит о том, что будет выполняться преобразование из кода Windows-1251 (одна из модификаций кода ASCII, содержащая русские символы) в Unicode. Encoding.GetEncoding(1251) реализован в пространстве имен System.Text.

Теперь для чтения данных из потока fileIn можно воспользоваться методом ReadLine. При этом если будет достигнут конец файла, то метод ReadLine вернет значение null.

По завершении работы с файлом его необходимо закрыть. Для этого достаточно вызвать метод Close(). При закрытии файла освобождаются системные ресурсы, ранее выделенные для этого файла, что дает возможность использовать их для работы с другими файлами.

1. Управляющие операторы условного и безусловного переходов. Разветвляющиеся программы

Как известно, из предыдущих лабораторных работ, все программы состоят из последовательности операторов, которые обычно выполняются поочерёдно в том порядке, в каком они записаны в программе. Однако часто возникает необходимость изменить очерёдность выполнения операторов, т.е. пропустить или наоборот выполнить какую-то группу операторов в зависимости от выполнения или не выполнения некоторых заданных условий. Кроме того иногда необходимо повторить группу операторов несколько раз, то есть организовать цикл. Для выполнения этих задач служат управляющие операторы. Управляющие операторы подразделяются на операторы принятия решения, к ним относятся операторы условного и безусловного переходов, и операторы для организации циклов, которые будут рассмотрены в следующей лабораторной работе.

Одной из важнейших возможностей компьютерного процессора является возможность принятия решения. Под этим выражением подразумевается, что процессор может направить поток выполнения запрограммированных команд по тому или иному пути в зависимости от того истинно или ложно некоторое заданное условие. Любой язык программирования обеспечивает возможность принятия решения. В алгоритмическом языке С#, как и во многих других, основой для такой возможности является оператор условного перехода if, который действует в C# практически также, как и оператор IF в любом другом языке программирования./

* 1. Оператор условного перехода if и его конструкции

Оператор условного перехода if (если), как было уже сказано, предназначен для выбора одного из возможных вариантов исполнения операторов программы, поэтому его и называют оператором принятия решения. Существует несколько разновидностей конструкций этого оператора. Рассмотрим их последовательно по мере усложнения.

if (условие выбора)

{

// Записанные в скобках операторы (оператор)

// будут выполняться, если условие выбора истинно

}

// Записанные далее операторы будут выполняться

// в любом случае, независимо от условия выбора.

В качестве условия выбора используется значение логического выражения. При выполнении этой конструкции вначале вычисляется значение логического выражения, записанного в скобках. Результат вычисления имеет тип boolean. Если вычисленное выражение имеет значение true (истина), то выполняются операторы в фигурных скобках и все следующие за ними. Если получено значение false (ложь) то операторы в фигурных скобках пропускаются и выполняются только операторы следующие за ними.

Пример 1.

// Сохраняем наибольшее значение из двух, a и b,

// в переменной

max max = b;

if (a > b)

{

max = a;

}

В данном фрагменте программы изначально предполагается, что наибольшее значение имеет переменная b, и она присваивается переменной max. Если это ни так, то переменной max присваивается значение переменной a.

Операторы, записываемые в фигурных скобках можно размещать в одной строке, как в следующем примере.

Пример 2.

max = b;

if (a > b){max = a;}

При записи в фигурных скобках нескольких операторов в конце каждого из них ставиться точка с запятой. Если в фигурных скобках записывается один оператор, то фигурные скобки можно опустить (см. пример 3).

Пример 3.

max = b;

if (a > b) max = a;

* 1. Конструкция if-else (если - иначе)

Данную конструкцию целесообразно использовать, когда необходимо задать выполнение одного из двух блоков операторов (или одного из двух операторов), в зависимости от результата проверки условия выбора. Конструкция имеет следующий вид записи.

If (условия выбора)

{

// Если условие выбора истинно, то будут выполняться

// оператор или операторы блока 1.

}

else

{

// В противном случае (иначе)

// если условие выбора ложно, то будут выполняться

// оператор или операторы блока 2.

}

// Записанные далее операторы будут выполняться

// в любом случае, независимо от условия выбора.

Если результатом проверки условия выбора является значение true (истина), то будут выполнены операторы блока 1. Далее будет выполняться первый оператор, следующий за последней фигурной скобкой. Операторы блока 2 выполняться не будут. Если проверка условия выбора даст результат false (ложь), то операторы блока 1 будут пропущены, и будут выполнены операторы блока 2. Далее будет выполняться первый

оператор, следующий за последней фигурной скобкой. Каждый из указанных блоков может состоять из одного оператора, тогда фигурные скобки могут быть опущены.

Пример 4.

If (a > b)

{

max = a;

// Эти операторы будут выполняться, min = b;

// если условие выбора a>b истинно.

}

else

{

max = b;

// Эти операторы будут выполняться, min = a;

// если условие выбора a > b ложно.

}

// Записанные далее операторы будут выполняться

// в любом случае, независимо от условия выбора.

Если a > b, то переменной max будет присвоено значении a, переменной min - значение b, в противном случае наоборот переменной max присваивается значение b, а переменной min - значение a. Рассмотренную конструкцию допускается записывать в одной строке, как в следующем примере 5.

Пример 5.

if (a > b)

{

max = a; min = b;

}

else

{

max = b; min = a;

}

Если в фигурных скобках записано по одному оператору, то скобки можно опустить, как в примере 6.

Пример 6.

if (a > b) max = a;

else max = b;

* 1. Вложенные конструкции оператора if

В том случае, когда определённый блок операторов (или один оператор) нужно выполнить после проверки ни одного, а нескольких условий, то используют несколько конструкций оператора if. Операторы if, находящиеся внутри другого оператора if, называются вложенными конструкциями оператора if.

if (Условие 1 выбора)

{

// Если условие 1 выбора истинно будут выполняться,

// записанные в скобках операторы блока 1.

}

еlse

// В противном случаи будет выполняться

// вложенная конструкция оператора

{

// Начало вложенной конструкции оператора

if (условие 2 выбора)

{

// Если условие 2 выбора истинно будут выполняться,

// записанные здесь в скобках операторы блока 2.

}

else

{

// В противном случае, если условие 2 выбора ложно

// будут выполняться, записанные здесь в скобках

// операторы блока 3.

}

}

// Конец вложенной конструкции оператора if

Если условие 1 выбора истинно, то выполняются операторы блока 1, и далее первый оператор, который следует за последней фигурной скобкой, концом вложенной конструкции оператора В противном случае выполняется вложенная конструкция оператора if. Если условие 2 выбора вложенного оператора if истинно, то выполняются записанные в фигурных скобках операторы блока 2, и далее первый оператор, который следует за последней фигурной скобкой, концом вложенной конструкции оператора if. В противном случае, если условие 2 выбора ложно выполняются операторы блока 3. Рассмотрим пример записи вложенной конструкции оператора if.

Пример 7.

if (х < -1) { n = 1; }

else

{

// Начало вложенной конструкции if.

if (х > 1)

{ n = 2; }

else

{ n = 0; }

// Конец вложенной конструкции if.

}

Допускаются и другие виды записи вложенной конструкции оператора if, например запись в одной строке.

Пример 8.

if (х < -1)

{ n = 1; }

else

{

if (х > 1) { n = 2; }

else { n = 0; }

}

Поскольку фигурные, операторные скобки являются обязательными только в случае записи в них нескольких операторов, поэтому в данном случае они могут быть отпущены.

Пример 9.

…

if (х < -1) n = 1;

else

if (х > 1) n = 2;

else n = 0;

…

* 1. Операторы логического сравнения

Эти операторы называются логическими сравнениями (logical comparisons), поскольку они возвращают результат сравнения в виде значения true (истина) или false (ложь) имеющие тип bool. Для записи и проверки условия равенства двух выражений, в алгоритмическом язык C# используется символ ==. Аналогично: символ > используется для проверки условия «больше»; символ < для проверки условия «меньше»; >= - «больше или равно»; <= - «меньше или равно»; != «не равно». Например: a!=b, означает, что оператор логического сравнения != возвращает значение true, если a не равно b.

* 1. Логические операторы

Для переменных типа bool используются специальные составные логические операторы:

& - конъюнкция (логическое и, and), используется для логического объединения двух выражений;

| - дизъюнкция (логическое или, or), используется, чтобы убедиться в том, что хотя бы одно из выражений true, истинно;

! - отрицание (логическое не, not), возвращает обратное логическое выражение;

^ - исключение (логическое исключающее или), используется для того, чтобы убедиться в том, что одно из двух выражений true, истинно.

Операторы &, | и а обычно используются с целыми типами данных, а также могут применяться к типу данных bool.

Кроме того могут применяться операторы && и ||, которые отличаются от своих односимвольных версий тем, что выполняют ускоренные вычисления. Например в выражении a && b, b вычисляется лишь в том случае, если а равно true, истинно. В выражении a\\b, b вычисляется в том случае, если a равно false, ложно.

Пример 10.

if (х > -1 && x < 1)

В условии оператора if записано обычное алгебраическое неравенство -1 < x < 1.

Пример 11.

if (x < -1 || x > 1)

В условии оператора if записаны алгебраические неравенства x < -1либо x > 1.

* 1. Оператор проверки

Оператор проверки выбирает одно из двух выражений в зависимости от проверки значения логического условия. Его синтаксис:

Имя переменной = (условие выбора)?Значение1:значение2

Пример 12.

int value = (x < 25) ? 5:15

В этом примере сначала вычисляется выражение x < 25 являющееся условием выбора. Если оно равно true, то переменной value будет присвоено значение равное 5, в противном случае - равное 15.

* 1. Оператор безусловного перехода goto

Оператор безусловного перехода goto (перейти к) осуществляет переход, без проверки каких-либо условий, к оператору, обозначенному соответствующей меткой. Синтаксис этого оператора выглядит следующим образом: метка: оператор

goto метка

где метка - метка. Это любой допустимый идентификатор C#, который помещается слева от оператора, которому надо передать управление выполнением программы и отделяется от него двоеточием. Причём метка может ставиться у оператора расположенного как до оператора goto, так и после него. В случае если оператор goto используется самостоятельно, без каких либо конструкций, то первый оператор, следующий за оператором goto, должен иметь свою метку, иначе он не будет выполнен в процессе работы программы. Обычно оператор goto используется совместно с оператором условного перехода if, и используется в программах редко, т. к. есть более эффективные операторы.

* 1. Конструкция switch (переключатель)

Этот оператор позволяет сделать выбор среди нескольких альтернативных вариантов дальнейшего выполнения программы. Несмотря на то, что это может быть организовано с помощью последовательной записи вложенных операторов if, во многих случаях более эффективным оказывается применение оператора switch. Ниже приведена общая форма оператора.

switch (выражение)

{

case константа 1:

последовательность операторов блока 1

break;

case константа 2:

последовательность операторов блока 2

break;

…

default

последовательность операторов блока n

break;

}

Этот оператор работает следующим образом. Значение выражения последовательно сравнивается с константами. Как только будет обнаружено совпадение, выполняется оператор или последовательность операторов, связанных с этим совпадением, до оператора break. Оператор break передаёт управление оператору, следующему за конструкцией switch. Если совпадений нет, то выполняется последовательность операторов, следующая после оператора default. Эта ветвь не является обязательной.

При использовании конструкции switch действуют следующие правила:

выражение в конструкции switch должно быть целочисленного типа (char, byte, short или int) перечислимого типа или же типа строкового;

нельзя использовать числа с плавающей точкой;

константы оператора case должны иметь тот же тип, что и выражение в конструкции switch;

в одном операторе switch не допускается наличие двух одинаковых по значению констант;

допускается использовать одну и ту же последовательность операторов, в этом случае оператор break не записывается.

Пример 13.

int n;

ml: Console.WriteLine("Возьмите целое число");

int a = int.Parse(( Console.ReadLine()));

switch (a)

{

case 1:

n = 10;

break;

case 2:

case 3:

n = 20;

break;

default:

n = 0;

break;

}

Console.WriteLine("a = " + a +" n = " + n);

if (a != 0) goto m1;

Console.Read();

В данном примере в программу вводится и присваивается переменной а, любое целое число. С помощью конструкции switch происходит анализ. Если переменная а имеет значение равное 1, переменной n присваивается значение 10 и далее следует вывод этих переменных. Если а имеет значение равное 2 или 3, то переменной n присваивается значение 20 и далее вывод этих переменных. Во всех остальных случаях переменной n присваивается значение 0. Программа продолжает работать до тех пор, пока переменой а, не будет задано значение 0.

Один оператор switch может быть частью последовательности другого внешнего оператора switch. Такой оператор называется вложенным. Константы внешнего и внутреннего операторов switch могут содержать общие значения, не вызывая каких либо конфликтов.

В операторе switch отсутствует возможность задания диапазона выбора, что является его недостатком. Например, в языке программирования Visual Basic в аналогичном операторе задание диапазона выбора допускается.

* 1. Разветвляющиеся программы

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название блока | Графическое изображение блока | Операторы и функции эквивалентные блоку |
| 1 | Блок ввода |  | Операторы ввода, функция InputВох и другие |
| 2 | Блок вывода |  | Операторы вывода, функция MsgBox и другие |
| 3 | Блок присваивания |  | Оператор присваивания |
| 4 | Блоки сравнения |  | Условный оператор if |

1. Блоки

Разветвляющиеся программы это - такие программы, в которых на определённых этапах происходит анализ значений тех или иных параметров и в зависимости от этого выбирается один из возможных вариантов дальнейшего хода программы. Практически все более или менее сложные программы являются разветвляющимися. Для их написания используются рассмотренные конструкции управляющих операторов принятия решения.

При написании разветвляющих программ предварительно составляется блок-схема алгоритма решения задачи. Блок-схема это - графическое изображение алгоритма или последовательности решения задачи программирования.

Для составления блок-схем используются стандартизованные графические изображения (блоки) определённых операторов алгоритмического языка. Некоторые из них представлены в таблице 3.1.

Далее рассмотрены примеры написания разветвляющихся программ, аналогичные тем, которые должен выполнить студент в данной лабораторной работе.

1. Управляющие операторы для организации циклов. Программы с циклами

Циклом называется процесс исполнения группы операторов программы заданное количество раз, либо до тех пор, пока выполняется или не выполняется некоторое поставленное условие. Процесс исполнения группы операторов цикла один раз называется итерацией цикла. Группа операторов, расположенная между началом и концом цикла называется телом цикла. В С# есть три основных вида циклов: цикл for с параметром (счётчиком), цикл while с предусловием и цикл do while с постусловием.

* 1. Оператор цикла for

Оператор цикла for (для) служит для организации циклов с параметром (счётчиком). Это наиболее распространённый оператор цикла. Он проще не только для чтения и понимания, но и проверки корректности цикл. Оператор цикла for имеет следующую структуру.

for (выражение 1; условие; выражение 2)

{

// тело цикла

}

Открывающиеся и закрывающаяся фигурная скобки являются соответственно началом и концом тела цикла. В круглых скобках указаны следующие параметры цикла:

выражение 1, в нём указывается имя переменной управления циклом и присваивается ей начальное значение;

условие, представляет собой булево выражение, проверяющее значение переменной управления циклом, если результат проверки истинен, то цикл продолжается, если ложен, то цикл завершается;

выражение 2, как правило, это арифметическое выражение, счётчик, определяет на каждой итерации цикла порядок изменения переменной управления циклом, на определённую величину - шаг цикла.

При работе цикла сначала выполняется выражение 1. В результате переменная управления циклом принимает своё первоначальное значение. Затем вычисляется условие и, если оно истинно, выполняются операторы тела цикла, заключённые в фигурные скобки. По достижению конца тела цикла (закрывающаяся фигурная скобка) управление переходит к вычислению выражение 2. В результате переменная цикла принимает новое значение, после чего вновь вычисляется условие и в случае его истинности цикл повторяется, в противном случае управление передаётся первому оператору, следующему за закрывающейся фигурной скобкой.

Пример 1.

int s, p, n; s = 0; P = 1;

for (n = 1; n <= 10; n = n + 1)

{

s = s + n; p = p \* n;

}

Console.WriteLine(,, s=" + s + " р=" + p);

В данной программе определяется сумма s и произведение р чисел от одного до десяти включительно. Имя переменной управления циклом n, её заданное начальное значение 0. Условие продолжения цикла n <= 10. Порядок изменения переменной управления циклом, счётчик - n = n + 1. Таким образом, n изменяется от 1 до 10 с шагом 1.При этом на каждой итерации сумма s увеличивается на n, а произведение р в n раз. Ниже представлены результаты работы программы.

Если в рассмотренной программе вывод организовать внутри цикла, то можно проанализировать увеличение сумм s и произведения p на каждой итерации цикла. Кроме того можно записать программу более компактно см. пример 2.

Пример 2.

int s, p, n;

s = 0;

p = 1;

for (n = 1; n <= 10; n ++) // или n += 1

{

s = s + n; p = p \* n;

Console.WriteLine("n =" + n +"s=" + s + "p=" +p);

}

Поскольку добавление 1 к переменной, в частности к n=n+1, является распространённой операцией в языке С# предусмотрена сокращённая запись этой операции, а именно n++, как в примере 2, или n+=1 . Этот оператор называется оператором инкремента, он часто используется как в цикле for так и в других операторах цикла.

В теле цикла for нельзя изменять его параметры. Если в цикле отсутствует параметр условие, то это соответствует значению true. Цикл, записанный в виде for(;;) - является бесконечным циклом.

* 1. Оператор цикла while

Оператор цикла while (пока), как и оператор цикла for, является циклом с предусловием. Это означает, что условие проверяется до начала цикла, и если оно имеет значение false, то цикл ни разу не выполняется. Отличие оператор цикла while от оператора цикла for состоит в том, что в нём заранее не определено количество итераций в цикле. Форма записи оператора.

while (условие)

{

// тело цикла

}

Операторы в теле цикла исполняются до тех пор, пока условие цикла выполняется, т.е. имеет значение true. Если вместо условия указано служебное слово true, т.е. while (true), то цикл будет бесконечным.

Пример 3. Составить программу, которая определяет сумму s и количество k целых чисел лежащих на заданном отрезке [a;b]. Границы отрезка целые числа.

int a,b,s,k;

Console.WriteLine(" Введите а" + "- левую границу отрезка.");

a = int.Parse((Console.ReadLine()));

Console.WriteLine(" Введите b" + "- правую границу отрезка.");

b = int.Parse((Console.ReadLine()));

s = 0;

k = 0;

while ( a <= b)

{

S += a; // означает s = s + a

k++; // означает k = k+1

a++; // означает a = a + 1

}

Console.WriteLine('\t' + " РЕЗУЛЬТАТ:" + " s =" + s + " k=" + k + '\n');

Переменная a управления циклом имеет начальное значение равное левой границе заданного отрезка. Перед каждым шагом она сравнивается с переменной b, которая имеет значение правой границы заданного отрезка. Если результат сравнения true, то цикл продолжается. Сумма чисел отрезка, переменная s, увеличивается на величину значения переменой a, которая является очередным числом заданного отрезка. Количество чисел отрезка, переменная k, и переменная a увеличиваются на единицу. Если результат сравнения false, то цикл заканчивается, и выдаются, например следующие результаты работы программы.

* 1. Оператор цикла do-while

Оператор цикла do-while является версией цикла while с постусловием. Это означает, что условие цикла проверяется после исполнения операторов тела цикла. Следовательно, в таком цикле одна, первая итерация, всегда будет выполняться. Иногда это удобно использовать. Форма записи оператора.

do

{

//тело цикла

}

while (условие)

После исполнения первой итерации цикла цикл продолжает работать до тех пор, пока условие цикла имеет значение true, в противном случае цикл заканчивается. В случае использования данного оператора в программе примера 3 она будет выглядеть следующим образом, см. пример 4.

Пример 4.

int a, b, s, k;

Console.WriteLine(" Введите a"+ " - левую границу отрезка");

a =int.Parse((Console.ReadLine()));

Console.WriteLine(" Введите b"+ " - правую границу

отрезка");

b = int.Parse((Console.ReadLine()));

s = 0;

k = 0;

do

{s += a; k++; a++;}

while (a <= b);

Console.WriteLine('\t' + " РЕЗУЛЬТАТ:" + " s =" + s + " k=" + k

+ '\n');

* 1. Операторы break и continue

Для управления циклом имеются специальные операторы - break и continue. Оператор break вызывает прекращение выполнения цикла и передачу управления первому оператору, следующему непосредственно за циклом. Оператор continue передаёт управление в начало цикла, к проверке условия. Обычно эти операторы используются совместно с оператором if.

* 1. Вложенные циклы

Цикл можно размещать внутри другого цикла. Размещение одной конструкции цикла в другой называется вложением циклов, а сам цикл вложенным.

При организации циклов, в том числе и вложенных, необходимо соблюдать следующие правила:

переменная, объявленная в теле цикла, определена только внутри этого цикла;

вход в цикл осуществляется через его заголовок;

вход во внутренний цикл осуществляется после входа во внешний;

выход из цикла, в том числе и вложенного возможен в любом месте тела цикла;

параметры циклов со счётчиком не должны изменяться в теле цикла;

вложенные циклы не должны пересекаться, т.е. начало и конец внутреннего цикла должны находиться во внешнем цикле;

допускается делать вложение разных конструкций циклов.

* 1. Суммирование рядов

Выше было рассмотрено использование циклических конструкций для вычисления, накапливания сумм (примеры 1-4). Накапливание сумм используется при вычислении, суммировании рядов, их членов. С помощью рядов могут вычисляться различные функции. Рассмотрим такую задачу на примере.

Пример 6.

Написать и отладить процедуру для приближённого вычисления функции y = sin(x) с помощью ряда

Суммирование членов ряда проводить включительно до члена ряда, значение которого по абсолютной величине меньше чем 10-6.

Вычислить сумму членов заданного ряда для угла в 30 градусов и сравнить полученное значение со значением, вычисленным непосредственно, с помощью функции y = sin(x).

Программа примера 6

double g, pi, x, y, s, a, p, d, чл, зн;

int n,k;

m1: Console.WriteLine(" Введите значение угла в градусах");

g = double.Parse((Console.ReadLine()));

pi = Math.PI; // Задание числа pi

x = g\*pi/180; // Перевод градусов в радианы

Console.WriteLine('\t' Промежуточные результаты" + '\n');

K = 0; // Номер итерации.

S = 0;

p = 1;

n = 3;

d = 1;

a = x; // Первый член ряда.

while (Math.Abs(a)>= 0.000001)

{

S += a; // Сумма ряда на текущей итерации,

чл = x\*x\*x\*d; // Вычисление числителя ряда

p = -p\*(n-1)\*n; // Вычисление факториала

зн = p; // Знаменатель ряда.

a = чл/зн; // Член ряда на текущей итерации.

d = x\*x;

n += 2;

k++; // Счётчик итераций.

Console.WriteLine('\t + " Итерация №" + k + '\n' + " a=" + a + " s=" + s +'\n');

}

y = Math.Sin(x); // Непосредственное вычисление синуса.

Console.WriteLine('\t' + " РЕЗУЛЬТАТЫ:" +'\n' + " Заданное значение угла в

град.=" + g +\n' + " Вычисленная сумма ряда S =" + s + '\n' +

" Количество членов ряда - " + k + '\n' + " Функция SIN(X)=" + y + '\n');

Console.WriteLine(,\n' + " Для повторного ввода" + " набрать любую

клавишу." +'\n' + " Для завершения программы нажмите Enter.");

string rep = Console.ReadLine();

if (rep != "") goto m1;

1. Одномерные массивы

Массивом называется последовательная группа переменных одного типа. Массивы служат для размещения набора данных, которые необходимо сохранить и использовать в процессе выполнения программы.

Элементы массива имеют одно и то же имя, но различаются порядковым номером (индексом), что позволяет компактно записывать множество операций с помощью циклов. В языке С#, как и во многих других языках, индексы задаются целочисленным типом.

Число индексов характеризует размерность массива. Каждый индекс изменяется в некотором диапазоне [a, b], который называется граничной парой, где а - нижняя граница, a b - верхняя граница индекса. При объявлении массива границы задаются выражениями. Если все границы заданы константными выражениями, то число элементов массива известно в момент его объявления и ему может быть выделена память еще на этапе трансляции. Такие массивы называются статическими. Если же выражения, задающие границы, зависят от переменных, то такие массивы называются динамическими.

Язык С# поддерживает два вида или две категории типов: типы значений (value types) и типы ссылок (reference types). Элементами массива могут быть как значения, так и ссылки. Массив значимых типов хранит значения, массив ссылочных типов - ссылки на элементы. Всем элементам при создании массива присваиваются значения по умолчанию: нули для значимых типов и null - для ссылочных. Массивы ссылочного типа являются массивами динамическими и память им отводится динамически в области памяти, называемой «кучей» (heap).

Массивами в С# можно пользоваться практически так же, как и в других языках программирования. Тем не менее у них имеется одна особенность: они реализованы в виде объектов. Реализация массивов в виде объектов дает ряд существенных преимуществ, и далеко не самым последним среди них является возможность утилизировать неиспользуемые массивы посредством "сборки мусора".

Поскольку в С# массивы реализованы в виде объектов, то для того чтобы воспользоваться массивом в программе, требуется двух- этапная процедура. Во-первых, необходимо объявить переменную, которая может обращаться к массиву:

тип[] имя\_массива;

во-вторых, нужно создать экземпляр массива, используя оператор new:

имя\_массива = new тип[размер];

где тип объявляет конкретный тип элемента массива, а размер определяет число элементов массива. Тип элемента определяет тип данных каждого элемента, составляющего массив. Квадратные скобки указывают на то, что объявляется одномерный массив.

Здесь необходимо отметить, что в отличии от других языков программирования (С, C++, Fortran или VBA), квадратные (или круглые) скобки следуют после названия типа, а не имени массива.

Рассмотрим конкретный пример. В приведенной ниже строке кода создается массив типа int из десяти элементов, и переменная array, которая является ссылкой на массив типа int[] с элементами типа int.

int[ ] array = new int[10];

В переменной array хранится ссылка на область памяти, выделяемой для массива оператором new. Эта область памяти должна быть достаточно большой, чтобы в ней могли храниться десять элементов массиватипа int.

Приведенное выше объявление массива можно разделить на два отдельных оператора. Например:

int[ ] array; // объявление массива с именем array

array = new int[10]; // резервирование памяти для

// 10 чисел типа int

Доступ к отдельному элементу массива осуществляется по индексу. В языке С# индекс первого элемента всех массивов является нулевым. В частности, массив array состоит из 10 элементов с индексами от 0 до 9. Для индексирования массива достаточно указать номер требуемого элемента в квадратных скобках. Так, первый элемент массива array обозначается как array[0], а последний его элемент - как array[9]. Ниже приведен пример программы, в которой заполняются все элементы массива iArray и массива chArray.

Пример 1.

// Заполнение массивов using System;

namespace Example5

{

class Example5\_1

{

static void Main()

{

int j;

Console.WriteLine("\n\n Одномерный массив iArray");

int[] iArray = new int[10];

for (j = 0; j < 10; j++)

iArray[j] = j \* j;

// присваивание значений

// элементам в цикле

for (j = 0; j < 10; j++)

// вывод элементов

Console.WriteLine("\n " + j + " " + iArray[j]);

Console.WriteLine("\n Одномерный массив chArray с

инициализацией");

char[] chArray = { 'a', 'b', 'c', 'd' };

// Объявление с инициализацией

j = -i;

do

{

j++;

Console.WriteLine("\n " + j + " " + chArray[j]);

}

while (chArray[j] != 'd');

// вывод элементов массива

Console.WriteLine();

Console.Write("\n Значения присвоены ");

Console.WriteLine("He всем элемента массива

iiArray \n");

int[] iiArray = new int[10];

for (j = 0; j < 6; j++)

iiArray[j] = j \* j;

iiArray[9] = 81;

foreach (int jj in iiArray)

{ Console.Write(" " + jj); }

Console.WriteLine("\n\n");

Console.Write(" ");

}

}

}

В начале программы объявлен массив iArray из 8 целых чисел. Потом в цикле присваиваются значения элементам. Аналогичный цикл используется и для вывода значений элементов на экран.

Далее объявляется массив символов chArray без указания количества элементов с инициализацией, после чего поэлементно выводится в цикле do-while.

Кроме описанных ранее, в С# определен цикл foreach. Он перебирает подряд все элементы массива. В программе foreach примененяется к массиву chArray.

Выражение

foreach(char jj in iiArray){...}

показывает, что все элементы массива iiArray поочередно присваиваются переменной цикла char, тип которой должен соответствовать типу массива. На местах элементов, которым не присвоены значения, цикл foreach выводит нули, что демонстрируется на примере массива iiArray.

Следует, однако, иметь в виду, что переменная цикла в операторе foreach служит только для вывода. Это означает, что, присваивая этой переменной новое значение, нельзя изменить содержимое массива.

* 1. Инициализация массивов

Как уже отмечалось, массив - это структура данных, содержащая несколько элементов одного типа. В следующих примерах показано создание и инициализация одномерных массивов.

// Объявление массива

int[ ] arrayl = new int[5];

// Объявление и инициализация массива

int[ ] array2 = new int[ ] {1, 3, 5, 7, 9} ;

// Альтернативный синтаксис

int[ ] array3 = {1, 2, 3, 4, 5, 6} ;

// при инициализации массива его размер можно

// указывать явным образом, но этот размер

// должен совпадать с числом инициализаторов

int[ ] array4 = new int[10] {99, 10, 100, 18, 1, 78, 22, 69};

* 1. Ввод-вывод массивов

Заполнить массив, т. е. определить значения элементов массива можно следующими способами:

при помощи оператора присваивания;

непосредственным вводом с клавиатуры;

подготовкой и вводом данных из текстового файла;

использования датчика случайных чисел;

заполнением массива при помощи стандартных функций

Пример 2.

// Ввод массива с клавиатуры

using System;

namespace Example5

{

class Example5\_2

{

static void Main()

{

int j;

// начальное значение

string strValue;

int[ ] iArray = new int[10];

for (j = 0; j < 10; j++)

{

strValue = Console.ReadLine();

// ввод и присваивание значений

iArray[j]= Convert.ToInt32(strValue);

}

for (j = 0; j < 10; j++)

// вывод элементов

Console.WriteLine("\n " + j + " " + iArray[j]);

}

}

}

Для организации ввода необходимо объявить строковую переменную, которой присваивается очередное введенное с клавиатуры число. Следующий оператор

iArray[j] = Convert.ToInt32(strValue);

преобразует строковую переменную strValue в целое 32-разрядное число. Ввод и преобразование происходит в цикле, после завершения которого массив iArray содержит исходные данные.

Каждый объект (переменная), каждый операнд при вычислении выражения, в том числе и само выражение характеризуется парой (значение, тип), задающей значение выражения и его тип. В процессе вычислений зачастую возникает необходимость преобразования типов - необходимость преобразовать пару (значение1, тип1) к паре (значение2, тип2).

Исходная пара называется источником преобразования, заключительная - целью преобразования. Некоторые преобразования типов выполняются автоматически. Такие преобразования называются неявными, и они часто встречаются при вычислении выражений. Все остальные преобразования называются явными и для них существуют разные способы таких преобразований - операция кастинга (приведение к типу), методы специального класса Convert, специальные методы ToString, Parse.

Все скалярные типы (арифметический, логический, символьный) имеют статический метод Parse, аргументом которого является строка, а возвращаемым результатом - объект соответствующего типа. Метод явно выполняет преобразование текстового представления в тот тип данных, который был целью вызова статического метода.

Для преобразований внутри арифметического типа можно использовать кастинг - приведение типа. Для преобразований строкового типа в скалярный тип можно применять метод Parse, а в обратную сторону - метод ToString.

Однако, во всех ситуациях, когда требуется выполнить преобразование из одного базового встроенного типа в другой базовый тип, можно использовать методы универсального класса Convert, встроенного в пространство имен System.

Методы класса Convert поддерживают общий способ выполнения преобразований между типами. Класс Convert содержит 15 статических методов вида (ToInt16(),ToInt32(), ToInt64(), ..., ToDouble, ToDecimal, ..., ). Единственным исключением является тип object - метода ToObject нет по понятным причинам, поскольку для всех типов существует неявное преобразование к типу object. Каждый из этих 15 методов перегружен, и его аргумент может принадлежать к любому из упомянутых типов. С учетом перегрузки с помощью методов этого класса можно осуществить любое из возможных преобразований одного типа в другой.

Пример 3.

// Заполнение массива с помощью генератора случайных

// чисел using System;

namespace Example5

{

class Example5\_3

{

static void Main()

{

int j, num1, num2;

string str;

double db1, db2;

Random rnd = new Random();

int[ ] iArray1 = new int[10];

int[ ] iArray2 = new int[10];

double[ ] dArray1 = new double[10];

double[ ] dArray2 = new double[10];

for (j = 0; j < 10; j++)

{

iArray1[j] = rnd.Next(1, 101);

iArray2[j] = 50 - rnd.Next(1,101);

}

for (j = 0; j < 10; j++)

{

num1 = rnd.Next(1, 101);

db1 = Convert.ToDouble(num1);

dArray1[j] = db1 + Convert.ToDouble(rnd.Next(1, 101)) / 100;

num2 = 50 - rnd.Next(1, 101);

db2 = Convert.ToDouble(num2);

dArray2[j] = db2 - Convert.ToDouble(rnd.Next(1, 101)) / 100;

}

Console.WriteLine("\n -------------------------- -------");

Console.WriteLine("\n Массивы типа int Массивы типа double");

Console.WriteLine("\n ----------------------------------");

for (j = 0; j < 10; j++)

{

str = string.Format("\n {0, 4:D} {1, 6:D} {2, 6:D} {3, 8:D} {4, 8:F2} {5, 8:F2}",

j, iArray1[j], iArray2[j], j, dArray1[j], dArray2[j],];

Console.WniteLine(str);

Console.WriteLine("\n ----------------------------------");

Console.WniteLine();

}

}

}

В данном примере для заполнения массива используется генератор случайных чисел. Для генерирования последовательного ряда случайных чисел служит класс Random. Такие последовательности чисел оказываются полезными в самых разных ситуациях включая имитационное моделирование. Начало последовательности случайных чисел определяется некоторым начальным числом, которое может задаваться автоматически или указываться явным образом.

В классе Random определяются два конструктора:

public Random ()

public Random(int seed)

Первый конструктор создает объект типа Random, использующий системное время определения начального числа. А во втором конструкторе используется начальное значение seed, задаваемое явным образом.

В первом цикле заполняются массивы iArray1 и iArray2, причем массив iArray1 заполняется числами от 0 до 100, массив iArray2 заполняется числами от -50 до 50. В этих же интервалах находятся и числа num1 и num2, которые в следующих строках преобразуются к типу double. Оператор Convert.ToDouble(rnd.Next(1, 101)) / 100; генерирует случайные числа, находящиеся в интервале от 0.0 до 1.0. Таким образом, массивы dArray1 и dArray2 будут содержать числа типа double. Основные методы класса Random представлены в таблице 10:

|  |  |
| --- | --- |
| Метод | Назначение |
| Public virtual int Next(int UpperBound) | Возвращает следующее случайное целое число, которое будет находиться в пределах от 0 до UpperBound-l включительно |
| Public virtual int Next (int LowerBound, int UpperBound) | Возвращает следующее случайное целое число, которое будет находиться в пределах от LowerBound до UpperBound-1 включительно |
| Public virtual double  NextDouble  (int UpperBound) | Возвращает следующее случайное число с плавающей точкой, больше или равно 0,0 и меньше 1,0 |

1. Методы класса Random.

Пример 4.

// Ввод массива в файл и вывод массива из файла

using System;

using System.IO;

namespace Example5

{

class Example5\_4

{

static void Main()

{

int j;

string strValue;

int[ ] iArray1 = new int[10];

int[ ]iArray2 = new int[10];

StreamReader sRead = new StreamReader("C:\\dat.txt");

StreamWriter sWrite = new StreamWriter ("C:\\res.txt");

for (j = 0; j < 10; j++)

{

strValue = sRead.ReadLine();

iArray1 [j] = Conveгt.ToInt32(stгValue);

iArray2[j] = 10 \* iArray1 [j];

stгValue = stгing.Foгmat("\n {0, 4:D} {1, 6:D} {2, 6:D}", j, iArray1[j],

iArray2 [j]);

Console.WriteLine(strValue);

Console.WгiteLine();

sWгite.WгiteLine(iArray2[j]);

}

sRead.Close();

sWгite.Close();

}

}

}

В этом примере происходит считывание из файла и запись в файл. Имя файла включает полный путь и, следовательно, содержит служебный знак \ (обратный слеш). Если используемые символы являются служебными, то на экран они не выводятся. Однако, в случае необходимости использовать эти символы в выводимом тексте, следует перед каждым таким символом поставить дополнительный символ \.

Все действия, т. е. считывание из файла и запись в файл, а также вывод на экран организованы в одном цикле. В переменной strValue формируется текстовая строка вывода на экран, содержащая форматы, заключенные в фигурные скобки

"\n {0, 4:D} {1, 6:D} {2, 6:D}",

где в первой позиции расположены цифры 0, 1 и 2, которые являются соответственно указателями на j, iArray1, iArray2. Код форматирования 4:D определяет число позиций (4), отведенных в строке вывода для данного целого числа (D - формат типа decimal).

Пример 5.

// Вычислить сумму и среднеарифметическое всех

// элементов одномерного массива,

// состоящего из 15 элементов.

// Заполнение массива происходит при

// помощи генератора случайных чисел

using System;

namespace Example5

{

class Example5\_5

{

static void Main()

{

int j, num;

string str;

string str1 = "Сумма", str2 = "Сумма";

string str3 = "СрАрф", str4 = "СрАрф";

double db1, db2;

double sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0;

Random rnd = new Random();

int[ ] iArray1 = new int[15];

int[ ] iArray2 = new int[15];

double[ ] dArray1 = new double[15];

double[ ] dArray2 = new double[15];

for (j = 0; j < 15; j++)

{

iArray1 [j] = rnd.Next(1, 101);

iArray2 [j] = 50 - rnd.Next(1, 101);

sum1 += iArray1 [j];

sum2 += iArray2 [j] ;

}

for (j = 0; j < 15; j++)

{

num = rnd.Next(1, 101);

db1 = Conveгt.ToDouble(num);

dArray1[j] = db1 + Conveгt.ToDouble(гnd.Next(1, 101)) / 100;

num = 50 - rnd.Next(1, 101);

db2 = Conveгt.ToDouble(num);

dArray2[j] = db2 - Conveгt.ToDouble(гnd.Next(1, 101)) / 100;

sum3 += dArray1[j];

sum4 += dArray2[j];

}

Console.WriteLine(''\n ------------------------------------------------------------------------'');

Console.WriteLine(''\n Массивы типа int

Массивы типа double '');

Console.WriteLine(''\n ------------------------------------------------------------------------'');

for (j = 0; j < 15; j++)

{

str = string.Format(''\n {0, 10:D} {1, 10:D} {2, 10:D} {3, 10:D} {4, 10:F2}

{S, 10:F2}'', j, iArray1[j], iArray2[j], j, dArray[j], dArray2[j]);

Console.WriteLine(str);

}

Console.WriteLine(''\n ------------------------------------------------------------------------'');

Console.WriteLine(''\n {0, 10} {1, 10} {2, 10} {3, 10} {4, 10:F2} {S, 10:F2}'', str1,

sum1, sum2, str2, sum3, sum4);

Console.WriteLine(''\n {0, 10} {1, 10:F2} {2, 10:F2} {3, 10} {4, 10:F2}

{S, 10:F2}'', str3, sum1/15, sum2/15, str4, sum3/15, sum4/15);

Console.WriteLine();

}

}

}

Массивы типа int iArray1, iArray2, а также типа double dArray1, dArray2 заполняются генератором случайных чисел. В первом цикле случайными числами инициализируется массивы целого типа и одновременно подсчитывается сумма элементов обоих массивов – sum1 и sum2. Переменная этого цикла одновременно является индексом массива. На каждом шаге цикла к сумме элементов каждого массива добавляется очередной элемент. По окончании цикла переменные sum1 и sum2 будут содержать полную сумму всех элементов массивов iArray1, iArray2.

В втором цикле инициализируется массивы чисел с плавающей точкой и одновременно по аналогичной схеме подсчитывается сумма элементов массивов dArray1, dArray2 - sum3 і sum4.

Далее формируется заголовок таблицы вывода, которая заполняется в третьем цикле. Для массивов типа int использован формат D, а для массивов типа double - формат F (fixed point).

Пример 6.

// Вычислить сумму всех четных элементов

// одномерного массива, состоящих из 10 элементов

// Заполнение одномерного массива происходит при

// помощи генератора случайных чисел

using System;

namespace Example5

{

class Example5\_6

{

static void Main()

{

int j, num, sum = 0;

Random rnd = new Random();

int[ ] iArray = new int[10];

for (j = 0; j < 10; j++)

{

iArray[j] = rnd.Next(1, 101);

}

for (j = 0; j < 10; j++)

{

num = Convert.ToInt32(iArray[j] % 2);

if (num == 0) sum += iArray[j];

}

foreach (int jj in iArray)

{ Console.Write(" " + jj); }

Console.WriteLine("\n\n");

Console.WriteLine("\n Сумма четных элементов = " + sum);

Console.WriteLine();

Console.Write(" ");

}

}

}

Отличие данного примера от предыдущего заключается в том, что требуется подсчитать сумму не всех элементов массива, а только четных

num = Convert.ToInt32(iArray[j] % 2);

if (num == 0) sum += iArray[j];

Сначала определяется остаток отделения элемента массива на 2 с помощью выражения iArray[j] % 2, результат которого преобразуется к целому типу. Если выполняется условие num == 0 , то значение элемента iArray[j] является величиной четной и происходит суммирование данного элемента с переменной sum.

Пример 7.

// Определить индексы второго положительного

// и третьего отрицательного элементов

// одномерного массива, состоящих из 20 элементов

// Заполнение массива происходит

// при помощи генератора случайных чисел

using System;

namespace Example5

{

class Example5\_7

{

static void Main()

{

int j, jnum = 0;

Random rnd = new Random();

int[ ] iArray = new int[20];

for (j = 0; j < 20; j++)

{

iArray[j] = 50 - rnd.Next(1, 101);

}

for (j = 0; j < 20; j++)

{

if (iArray[j] > 0) jnum += 1;

if (jnum == 2) break;

}

foreach (int jj in iArray)

{ Console.Write(" " + jj); }

Console.WriteLine("\n\n");

Console.WriteLine("\n индекс второго положительного

элемента = " + j);

Console.WriteLine("\n iArray[" + j + "] = " + iArray[j]);

jnum = 0;

for (j = 0; j < 20; j++)

{

if (iArray[j] < 0) jnum += 1;

if (jnum == 3) break;

}

Console.WriteLine("\n индекс третьего отрицательного

элемента = " + j);

Console.WriteLine("\n iArray[" + j + "] = " + iArray[j]);

Console.WriteLine();

Console.Write(" ");

}

Поиск второго положительного элемента массива iArray происходит в цикле. Переменная jnum определяет количество положительных элементов, которые встретились при выполнении цикла. Как только значение выражения jnum == 2 будет равно true оператор break прерывает выполнение цикла.

Аналогичный алгоритм используется при нахождении третьего отрицательного элемента.

Пример 8.

// Задан одномерный массив размером N.

// Сформировать два массива, включив

// в первый - четные элементы исходного

// массива, а во второй - нечетные элементы.

// Отсортировать массивы в порядке возрастания.

// Заполнение массива происходит при помощи

// генератора случайных чисел

using System;

namespace Example5

{

class Example5\_8

{

static void Main()

{

int jnum = 0, N = 20;

int jAA = 0, jBB = 0;

int j, k, temp;

Random rnd = new Random();

int[ ] iArray = new int[N];

int[ ] jA = new int[N];

int[ ] jB = new int[N];

for (j = 0; j < N; j++)

{

iArray[j] = rnd.Next(1, 101);

}

Console.WriteLine("\n исходный массив \n\n");

foreach (int jj in iArray)

{ Console.Write(" " + jj); }

Console.WriteLine("\n\n");

for (j = 0; j < N; j++)

{

jnum = iArray[j] / 2;

iArray[j] = Convert.ToInt32(iArray[j]);

if (iArray[j] == jnum \* 2)

{

jA[jAA] = iArray[j];

jAA += 1;

}

else

{

jB[jBB] = iArray[j];

jBB += 1;

}

}

Console.WriteLine("\n массив A[ ] \n\n");

foreach (int jj in jA)

{ Console.Write(" " + jj); }

Console.WriteLine("\n\n");

Console.WriteLine("\n массив B[ ] \n\n");

foreach (int jj in jB)

{ Console.Write(" " + jj); }

Console.WriteLine("\n\n");

// Сортировка массива А

jAA - = 1;

for (k = 0; k < jAA; k++)

{

for (j = 0; j < jAA; j++)

{

if (jA[j + 1] < jA[j])

{

temp = jA[j];

jA[j] = jA[j + 1];

jA[j + 1] = temp;

}

}

}

// Сортировка массива B jBB -= 1;

for (k = 0; k < jBB; k++)

{

for (j = 0; j < jBB; j++)

{

if (jB[j + 1] < jB[j])

{

temp = jB[j];

jB[j] = jB[j + 1];

jB[j + 1] = temp;

}

}

Console.WriteLine("\n отсортированный массив A[ ] \n\n");

foreach (int jj in jA)

{ Console.Write(" " + jj); }

Console.WriteLine("n\n");

Console.WriteLine("\n отсортированный массив B[ ] \n\n");

foreach (int jj in jB)

{ Console.Write(" " + jj); }

Console.WriteLine(" \n\n");

}

}

}

Для определения четности или нечетности очередного элемента массива. Если результат данного преобразования равен исходному значению, то данный элемент массива содержит четное число, в противном случае - число нечетное. По окончании данной процедуры значения всех элементов массива jA - четные числа, массива jA - нечетные числа.

Далее происходит сортировка элементов массивов jA и jВ по возрастанию. Процесс сортировки происходит следующим образом: если элемент с индексом j больше элемента с индексом j+1, то выполняется процедура перестановки

temp = jB[j];

jB[j] = jB[j + 1];

jB[j + 1] = temp;

и завершении цикла массивы jA и jB будут отсортированы по возрастанию.

1. Многомерные массивы

В языке С++ все массивы являются статическими; более того, все массивы являются 0-базируемыми. Это означает, что нижняя граница всех индексов массива фиксирована и равна нулю.

В языке С# снято существенное ограничение языка C++ на статичность массивов. Массивы в языке С# являются настоящими динамическими массивами. Они относятся к ссылочным типам и память им отводится динамически в "куче". Однако, не снято ограничение 0-базируемости, хотя во многих задачах гораздо удобнее работать с массивами, у которых нижняя граница не равна нулю.

В языке C++ "классических" многомерных массивов нет. Вместо них введены одномерные массивы и массивы массивов, которые являются более общей структурой данных и позволяют задать не только многомерный куб, но и изрезанную, ступенчатую структуру.

В языке С#, соблюдая преемственность, сохранены одномерные массивы и массивы массивов. В дополнение к ним в язык добавлены многомерные массивы.

Простейшей формой многомерного массива является двумерный массив. Местоположение любого элемента в двумерном массиве обозначается двумя индексами. Такой массив можно представить в виде таблицы, первый индекс которого указывает на строки данной таблицы, а второй - на ее столбцы.

В следующей строке кода объявляется двумерный массив integer размерами 10x15:

int[,] table = new int[10, 15];

При объявлении этого массива оба его размера разделяются запятой. В первой части этого объявления синтаксическое обозначение означает, что создается переменная типа ссылки на двумерный массив. Если же память распределяется для массива с помощью оператора new, то используется следующее синтаксическое обозначение:

int[10, 15]

Данное объявлении создает массив размерами 10x15. Для доступа к элементу двумерного массива следует указывать оба индекса, разделив их запятой.

В следующей строке кода элементу массива myArray с координатами положения, т. е. индексами (3,5) присваивается значение 10:

myArray [3, 5] = 10;

В С# допускаются массивы трех и более измерений.

Общая форма объявления многомерного массива.

тип[,•••,] имя\_массива = new тип[размер13 размер23 ... размеры];

Например, в приведенном ниже объявлении создается трехмерный целочисленный массив размерами 4x10x3.

int [,,] multiArray = new int[4, 10, 3];

А в следующем операторе элементу массива multiArray, положение которого определяется индексами (2,4,1) присваивается значение 100:

multidim[2, 4, 1] = 100;

Ниже приведен пример программы, в которой сначала организуется трехмерный массив, содержащий матрицу значении 3x3x3, а затем вычисляется сумма значений диагональных элементов этого массива.

// Суммировать значения по одной из диагоналей

// матрицы 3x3x3.

using System;

class Example

{

static void Main()

{

int [,,] m = new int[3,-3,3];

int sum = 0;

int n = 1;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

for (int k = 0; k < 3; k++)

m[i, j, k] = n++;

sum = m[0, 0, 0] + m[l, 1, 1] + m[2, 2, 2];

Console.WriteLine("Cyммa значений по диагонали: " + sum);

}

}

Результат выполнения этой программы:

Сумма значений по диагонали: 42

Инициализация многомерных массивов

Для инициализации многомерного массива достаточно заключить в фигурные скобки список инициализаторов каждого его размера. Общая форма инициализации двумерного массива имеет следующий вид:

тип[,] имя\_массива =

{

{vaL, vaL, vaL, ..., val},

{val, vaL, vaL, ..., vaL},

{vaL, vaL, vaL, ... , vaL} };

где vaL обозначает инициализирующее значение, а каждый внутренний блок - отдельный ряд. Первое значение в каждом ряду сохраняется на первой позиции в массиве, второе значение - на второй позиции и т. д. Блоки инициализаторов разделяются запятыми, а после завершающей эти блоки закрывающей фигурной скобки ставится точка с запятой.

Следующие два примера иллюстрируют возможность инициализации массива непосредственным вводом с клавиатуры и считыванием информации из текстового файла.

Пример 1.

// Заполнить двумерный массив

// непосредственно вводом с клавиатуры

using System;

namespace Example6

{

class Example6\_1

{

static void Main()

{

int i, j;

string strValue;

int[,] iArray = new int[3, 4];

int[,] jArray = new int[3, 4];

for (i = 0; i < 3; ++i)

{

for (j = 0; j < 4; ++j)

{

// ввод и присваивание значений

strValue = Console.ReadLine();

iArray[i, j] = Convert.ToInt32(strValue);

}

}

// вывод значений массива iArray на экран

for (i = 0; i < 3; ++i)

{

for (j = 0; j < 4; ++j)

{

Console.Write(" iArray[" + i + " + j + "] = " + iArray[i, j]);

}

Console.WriteLine();

}

Console.WriteLine();

// вывод значений массива jArray на экран

for (i = 0; i < 3; ++i)

{

for (j = 0; j < 4; ++j)

{

jArray[i, j] = iArray[i, j] \* 10;

Console.Write(" jArray[" + i + ", " + j + "] = " + jArray[i, j]);

}

Console.WriteLine();

}

Console.WriteLine();

}

}

}

Основным методом, используемым для чтения данных с консоли, является метод ReadLine. Он читает с консоли строку текста, завершаемую признаком конца строки. Эта строка и является результатом, возвращаемым методом ReadLine. Введенная строка с помощью метода ToInt32 конвертируется в значение типа int и это значение присваивается элементу iArray[i, j]. Далее в двойном цикле заполняется массив

jArray jArray[i, j] = iArray[i, j] \* 10;

и затем оба массива выводятся на экран.

Для ввода данных из файла используется конструктор класса StreamReader. В классе StreamReader определено несколько конструкторов. Наиболее часто используемый

StreamReader(string имя\_файла),

где имя\_файла - это имя открываемого файла, включая полный путь к нему. Здесь необходимо отметить, что в имени файла присутствует служебный символ \, который обычно на экран не выводится. Если все же необходимо использовать служебные символы в выводимом тексте, следует перед таким символом поставить дополнительный символ \ (обратный слеш).

Пример 2.

// Двумерный массив

// Ввод массива в файл и

// вывод массива из файла

using System;

using System.IO;

namespace Example6

{

class Example6\_2

{

static void Main()

{

int i, j;

string strValue;

int[,] iArray1 = new int[3, 4];

int[,] iArray2 = new int[3, 4];

StreamReader sRead = new StreamReader(''С:\\C#\\dat.txt'');

StreamWriter sWrite = new StreamWriter(''С:\\C#\\res.txt'');

for (i = О; i < З; i++)

{

for (j = О; j < 4; j++)

{

strValue = sRead.ReadLine();

iArray1[i, j] = Convert.ToInt32(strValue);

iArray2[i, j] = iArray1[i, j] \* 100;

strValue = string.Format(''\n {0, 4:D} {1, 4:D} {2, 6:D} {З, 6:D}'',

i, j, iArray1[i, j], iArray2[i, j]);

Console.WriteLine(strValue); Console.WriteLine();

}

}

for (i = 0; i < 3; i++)

{

for (j = 0; j < 4; j++)

{

sWrite.WriteLine(iArray2[i, j]);

}

}

sWrite.Close();

}

}

}

Смысл этого примера достаточно прозрачен и отличается только тем, что вместо класса Console (Example6\_1) используются классы StreamReader и StreamWriter (Example6\_2).

Следующая программа содержит пример ввода и вывода массива из файла. Массив iArray1 содержит три столбца, первый из которых заполнен случайными числами типа int, второй - случайными числами типа double и третий значениями функции Sin(x). Далее формируется строковая переменная strValue, которая форматируется и используется для вывода массива iArrayl на экран. Следующий оператор

iArray2[i, j] = iArray1[i, j] \* 100;

инициализирует массив iArray2 и производит запись символьного потока в текстовой файл res3.txt.

Класс Array - это класс, обслуживающий массивы в пространстве имен System. Его свойства и методы можно использовать для массивов любого из встроенных типов.

Свойство Length доступно только для чтения, имеет тип int и содержит количество элементов массива.

Метод Copy (Array source, Array dest, count) копирует число элементов, задаваемых параметром count, из исходного массива source в целевой массив dest, начиная с первого элемента массива. Далее происходит вывод массива iArray1на экран.

Пример 3.

// Заполнить двухмерный массив

// и организовать вывод на экран.

// Ввод массива в файл и

// вывод массива из файла

using System;

using System.IO;

namespace Example6

{

class Example6\_3

{

static void Main()

{

int i, j, num;

double x = 0;

string strValue;

double[,] iArray1 = new double[10, 3];

double[,] iArray2 = new double[10, 3];

Random rnd = new Random();

StreamWriter sWrite = new StreamWriter("C:\\C#\\res3.txt");

Console.WriteLine("\n ---");

for (i = 0; i < 10; i++)

{

num = rnd.Next(1, 51);

iArray1[i, 0] = Convert.ToDouble(num);

num = (rnd.Next(1, 101) - 50) / 10;

iArray1[i, 1] = Convert.ToDouble(num);

iArray1[i, 2] = Math.Sin(x);

x = x + 0.314159;

strValue = string.Format("\n {0, 4:D} {1, 10:F0} {2, 10:F2} {3, 10:F2}",

i, iArray1[i, 0], iArray1[i, 1], iArray1[i, 2]);

Console.WriteLine(strValue);

}

Console.WriteLine("\n --------------------------------------------------------------------");

for (i = 0; i < 10; i++)

{

for (j = 0; j < 3; j++)

{

iArray2[i, j] = iArray1[i, j] \* 100;

sWrite.WriteLine(iArray2[i, j]);

}

}

num = iArray2.Length;

Array.Copy(iArray2, iArray1, num);

for (i = 0; i < 10; i++)

{

strValue = string.Format("\n {0, 4:D} {1, 10:F0} {2, 10:F2} {3, 10:F2}",

i, iArray2[i, 0], iArray2[i, 1], iArray2[i, 2]);

Console.WriteLine(strValue);

sWrite.WriteLine(iArray2[i, j]);

}

Console.WriteLine("\n --------------------------------------------------------------------");

sWrite.Close();

}

}

}

Пример Example6\_4 демонстрирует метод вычисления суммы и среднеарифметического всех элементов массива, который заполняется случайными числами. Эти вычисления аналогичны тем, которые проводились для одномерного массива. Разница лишь в том, что в случае одномерного массива использовался одинарный цикл, а для двумерного массива - двойной.

Пример 4.

// Заполнить двухмерный массив случайными числами.

// Вычислить сумму и среднеарифметическое всех

// элементов

using System;

using System.IO;

namespace Example6

{

class Example6\_4

{

static void Main()

{

int i, j;

int num, sum = 0;

int[,] iArray = new int[5, 6];

int[ ] iA = new int[6];

Random rnd = new Random();

string strValue = "\n ";

Console.Write("\n ");

// номера столбцов

for (j = 0; j < 6; j++)

Console.Write("{0, 5}", j);

Console.WriteLine(strValue);

// заполнение массива; сумма и

// среднеарифметическое всех элементов

for (i = 0; i < 5; i++)

{

for (j = 0; j < 6; j++)

{

num = rnd.Next(1, 101);

iArray[i, j] = Convert.ToInt32(num);

iA[j] = iArray[i, j];

sum = sum + iArray[i, j];

}

}

// вывод массива

for (i = 0; i < 5; i++)

{

// номер строки

Console.Write("\n " + i);

for (j = 0; j < 6; j++)

{

// элементы строки

Console.Write("{0, 5}", iArray[i, j]);

}

Console.WriteLine(strValue);

Console.WriteLine();

Console.WriteLine(" Сумма элементов массива: " + sum);

Console.WriteLine(" Среднеарифметическое: " + sum / 30);

Console.WriteLine();

}

}

}

Следующий пример демонстрирует перестановку двух произвольных строк в двумерном массиве А, который инициализируется случайными числами. Все происходит аналогично перестановке любых двух элементов одномерного массива. Разница, как и раньше, лишь в том, что в случае одномерного массива использовался одинарный цикл, а для двумерного массива - двойной.

Пример 5.

// Заполнить двухмерный массив A(M, N)

// случайными числами.

// Переставить в этом массиве строки 2 иМ - 2

using System;

namespace Example6

{

class Example6\_5

{

static void Main()

{

int M = 6, N = 8;

int i, j, temp;

int[,] iArray1 = new int[M, N];

int[,] iArray2 = new int[M, N];

int[ ] iArray3 = new int[N];

Random rnd = new Random();

string strValue = "\n ";

Console.Write("\n ");

Console.WriteLine("\n Исходный массив: ");

Console.WriteLine(strValue);

// заполнение массива

for (i = 0; i < M; i++)

{

for (j = 0; j < N; j++)

{

temp = rnd.Next(1, 101);

iArray1[i, j] = Convert.ToInt32(temp);

iArray2[i, j] = Convert.ToInt32(temp);

iArray3[j] = iArray1[i, j];

}

foreach (int jj in iArray3)

{

Console.Write("{0, 5}", jj);

}

Console.Write("\n");

}

Console.WriteLine(strValue);

Console.WriteLine("\n Массив после перестановки: ");

Console.WriteLine(strValue);

for (j = 0; j < N; j++)

{

temp = iArray2[2, j];

iArray2[2, j] = iArray2[M - 2, j];

iArray2[M - 1, j] = temp;

}

for (i = 0; i < M; i++)

{

for (j = 0; j < N; j++)

{

iArray3[ j] = iArray2[i, j]

}

foneach (int jj іn iArray3)

{

Console.Write("{0, 5}", jj');

}

Console.Write("\n");

}

Console.WriteLine(strValue);

Console.WriteLine();

Console.WriteLine();

}

}

}

Пример 6.

// Заполнить двухмерный массив A(5, 6) случайными числами.

// Найти в этом массиве строку, сумма элементов которой

// является максимальной, а в ней минимальный по величине элемент

using System;

namespace Example6

{

class Example6\_6

{

static void Main()

{

int im = 0, jm = 0, i, j;

int num, sum = 0;

int max = -100, min = 100;

int[,] iArray = new int[5, 6];

int[ ] iA = new int[6];

Random rnd = new Random();

string strValue = "\n ------------------------------------------------------------------------------";

Console.Write("\n ");

// номера столбцов

for (j = 0; j < 6; j++)

Console.Write("{0, 5}", j);

Console.WriteLine(strValue);

// заполнение массива; сумма строк

for (i = 0; i < 5; i++)

{

sum = 0;

for (j = 0; j < 6; j++)

{

num = rnd.Next(1, 101);

iArray[i, j] = Convert.ToInt32(num);

iA[j] = iArray[i, j];

sum = sum + iArray[i, j];

}

if (sum > max)

{

max = sum;

im = i;

}

}

for (j = 0; j < 5; j++)

{

if (iArray[im, j] < min)

{

min = iArray[im, j];

jm = j;

}

}

// вывод массива

for (i = 0; i < 5; i++)

{

// номер строки

Console.Write("\n " + i);

for (j = 0; j < 6; j++)

{

// элементы строки

Console.Write("{0, 5}", iArray[i, j]);

}

}

Console.WriteLine(strValue);

Console.WriteLine();

Console.WriteLine("\n i = " + im);

Console.WriteLine("\n j = " + jm);

Console.WriteLine();

}

}

}

Сумма элементов строки вычисляется во внутреннем цикле по j . На каждом шаге цикла к сумме элементов строки i добавляется очередной элемент. По окончании цикла переменная sum будет содержать полную сумму всех элементов данной строки. Максимальная сумма определяется во внешнем цикле. Для этого используется следующая группа операторов

if (sum > max)

{

max = sum;

im = i;

}

которая определяет максимальную сумму и номер строки im. Другая группа операторов

if (iArray[im, j] < min)

{

min = iArray[im, j];

jm = j;

}

используется для поиска минимального элемента и его номера jm в найденной строке.

Генерация таблицы умножения достаточно тривиальная задача. Каждый элемент таблицы определяется произведением номера строки на номер столбца k = i \* j. Единственная трудность заключается в организации вывода таблицы на экран.

Конструкция выбора

if (k < 10) Console.Write(" ");

else Console.Write(" ");

вставляет перед числом k < 10 два пробела, а во всех остальных случаях - один пробел.

Пример 7.

// Рассчитать и вывести на экран таблицу умножения.

// Подсчитать общее количество чисел в данной таблице

using System;

namespace Example6

{

class Example6\_7

{

static void Main()

{

int i, j, k, m = 0;

for (i = 1; i < 10; i++)

{

Console.WriteLine("\n");

for (j = 1; j < 10; j++)

{

k = i \* j;

m++;

if (к < 10) Console.Write(" ");

else Console.Write(" ");

Console.Write(k);

}

}

Console.WriteLine(" \n\n всего чисел: " + m);

Console. WriteLine ("\n",);

}

}

}

Программа данного примера выводит на экран верхнюю и нижнюю треугольные матрицы. Для управления выводом используются операторы break и continue. Оператор break прерывает выполнение цикла при выполнении заданного условия. Оператор continue также при выполнении заданного условия приводит к пропуску следующих за ним операторов блока, но без выхода из блока.

Пример 8.

// Рассчитать таблицу (матрицу) умножения.

// Вывести на экран верхнюю треугольную и нижнюю

// треугольную матрицы. Подсчитать общее

// количество чисел (элементов) в каждой из матриц.

using System;

namespace Example6

{

class Example6\_8

{

static void Main()

{

// верхняя треугольная матрица

int i, j, k, m = 0;

for (i = 1; i < 10; i++)

{

Console.WriteLine("\n");

for (j = 1;j < 10; j++)

{

k = i \*j;

if (j >10 - i) break;

m++;

if (k < 10) Console.Write(" ");

else Console.Write(" ");

Console.Write(k);

}

}

Console.WriteLine("\n\n всего чисел: " + m);

Console.WriteLine("\n");

m = 0;

for (i = 1; i < 10; i++)

{

Console.WriteLine("\n");

for (j = 1; j < 10; j++)

{

k = i \* j;

if (k < i \* (10 - i))

Console.Write(" ");

else

{

if (k < 10) Console.Write (" " + k);

else Console.Write(" " + k);

m++;

}

}

}

Console.WriteLine("\n\n всего чисел: " + m);

Console.Write(" \n");

}

}

}

В приведенных выше примерах применения двумерного массива, по существу создавался так называемый прямоугольный массив. Двумерный массив можно представить в виде таблицы, в которой длина каждой строки остается неизменной по всему массиву. Но в С# можно также создавать специальный тип двумерного массива, называемый ступенчатым массивом. Ступенчатый массив представляет собой массив массивов, в котором длина каждого массива может быть разной. Следовательно, ступенчатый массив можно использовать для составления таблицы из строк разной длины.

Ступенчатые массивы объявляются с помощью ряда квадратных скобок, в которых указывается их размерность. Например, для объявления двумерного ступенчатого массива служит следующая общая форма:

тип[ ][ ] имя\_массива = new тип[размер][];

где размер обозначает число строк в массиве. Память для самих строк распределяется индивидуально, и поэтому длина строк может быть разной. Память сначала распределяется для его первого измерения автоматически, а затем для второго измерения вручную. После создания ступенчатого массива доступ к его элементам осуществляется по индексу, указываемому в отдельных квадратных скобках. Например, в следующей строке кода элементу массива jagged, находящемуся на позиции с координатами (2,1), присваивается значение 10:

jagged[2][1] = 10;

Обратите внимание на синтаксические отличия в доступе к элементу ступенчатого и прямоугольного массива.

В приведенном ниже примере программы демонстрируется создание двумерного ступенчатого массива. Здесь следует обратить внимание, что язык C# позволяет использовать в качестве имен переменных и массивов русские имена.

Пример 9.

// Инициализировать и вывести на экран

// ступенчатый массив.

using System;

namespace Example6

{

class Example6\_9

{

static void Main()

{

int j, max = 0;

int[ ] Массив = new int[5];

for (j = 0; j < 5; j++)

{

Массив[j] = j + 2;

if (Массив[j] > max) max = Массив[j];

}

// вспомогательный массив, определяющий

// границы каждой из строк ступенчатого массива

int размер0 = 5;

int размер1 = max;

int размер2 = Convert.ToInt32((4 \* max - 1) / 3);

int[ ] размер = { размер0, размер1, размер2 };

int[ ][ ] стМассив = new int[3][ ];

стМассив[0] = new int[paзмep[0]];

стМассив[1] = new int[pa3Mep[1]];

стМассив[2] = new int[pa3Mep[2]];

Console.WriteLine();

Console.Write(" ");

for (j = 0; j < размер2; j++)

Console.Write("{0,5}", j);

Console.WriteLine("\n\t\t ------------------------------------------------------------------ ");

Console.Write(" стМассив[0][5]");

for (j = 0; j < размер0; j++)

стМассив[0][ j] = j + 1;

foreach (int jj in стМассив[0])

Console.Write("{0,5}", jj);

Console.WriteLine();

Console.Write(" cтМассив [1][6] ");

for (j = 0; j < размер1; j++)

стМассив [1][ j] = j + 11;

foreach (int jj in стМассив[1])

Console.Write("{0,5}", jj);

Console.WriteLine();

Console.Write(" стМассив[2][7]");

for (j = 0; j < размер2; j++)

стМассив[2][ j] = j + 111;

foreach (int jj in стМассив[2])

Console.Write("{0,5}", jj);

Console.WriteLine("\n\t\t -------------------------------------------------------------- ");

Console.WriteLine();

}

}

}

Ступенчатые массивы находят полезное применение не во всех, а лишь в некоторых случаях. Так, если требуется очень длинный двумерный массив, который заполняется не полностью, т.е. такой массив, в котором используются не все, а лишь отдельные его элементы, то для этой цели идеально подходит ступенчатый массив.

Следует отметить, что ступенчатые массивы представляют собой массивы массивов, и поэтому они не обязательно должны состоять из одномерных массивов. Так в приведенной ниже строке кода создается массив двумерных массивов.

int[ ][,] jArray = new int[5][,];

1. Классы и методы

Класс является обобщенным понятием, определяющим характеристики и поведение некоторого множества конкретных объектов этого класса, называемых экземплярами, или объектами, класса.

В целях ускорения работы программиста и упрощения применяемых алгоритмов применяют классы. Применение классов позволяет разбить сложную задачу на ряд простых, взаимосвязанных задач. Проделав эту операцию неоднократно применительно к вновь полученным классам, можно получить алгоритм решения задачи в виде набора простых, понятных, легко осуществимых классов, взаимодействующих друг с другом.

«Классический» класс содержит данные, задающие свойства объектов класса, и функции, определяющие их поведение. В последнее время в класс часто добавляется третья составляющая - события, на которые может реагировать объект класса. Это оправдано для классов, использующихся в программах, построенных на основе событийно-управляемой модели, например, при программировании для Windows.

Процедура объявления и создания экземпляров пользовательского типа не отличается от таковой для предопределенных типов за исключением необходимости использования Спецификатора New (new) для создания экземпляров пользовательских типов значений и ссылочных типов.

**Спецификаторы классов:**

new - используется для вложенных классов. Задает новое описание класса взамен унаследованного от предка. Применяется в иерархиях объектов.

public - доступ не ограничен

protected - используется для вложенных классов. Доступ только из элементов данного и производных классов

internal - доступ только из данной программы (сборки)

protected internal - доступ только из данного и производных классов или из данной программы (сборки)

private - используется для вложенных классов. Доступ только из элементов класса, внутри которого описан данный класс

abstract - абстрактный класс. Применяется в иерархиях объектов.

sealed - бесплодный класс. Применяется в иерархиях объектов.

static - статический класс. Введен в версию языка 2.0.

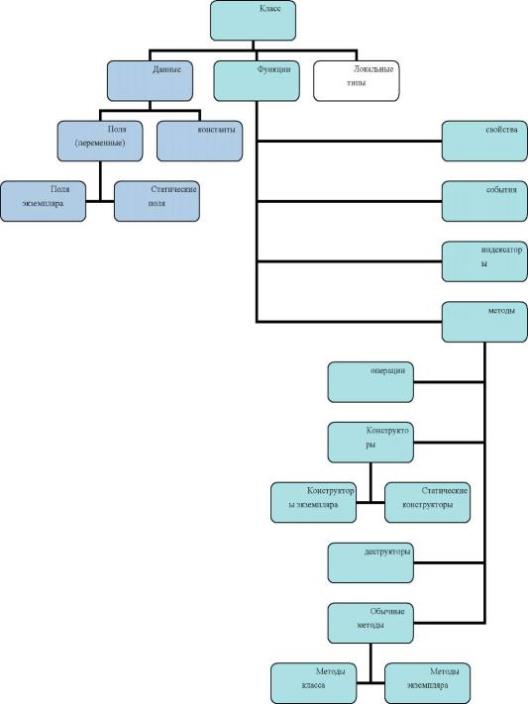
Спецификаторы 2 - 6 называются спецификаторами доступа. Они определяют, откуда можно непосредственно обращаться к данному классу. Спецификаторы доступа могут присутствовать в описании только в вариантах, приведенных в таблице, а также могут комбинироваться с остальными спецификаторами.

* 1. Методы

Методы делают всю работу, для исполнения которой предназначены классы и структуры: вычисляют значения, изменяют данные, получают вводимую информацию - в общем, выполняют все действия, составляющие поведение объекта

* 1. Объявление методов

Один из видов членов, которые добавляются к классам, - это методы, определяющие действия, которые должен выполнять класс. Есть две разновидности методов: те, что возвращают значения, и те, которые никаких значений не возвращают. Следующий пример иллюстрирует обе разновидности методов:



1. Рис. 7.1. Класс

// Этот метод возвращает целые (int) значения

public int Add (int first, int second)

{

int Result;

Result = first + second;

Return result;

}

public void myVoidMethod ()

// void - не возвращает значений

{

MessageBox.Show ("Этот метод не возвращает значения ");

}

Метод является самостоятельной частью кода, которая имеет имя и может содержать аргументы, выполнять последовательность инструкций (операторов) и изменять значения своих аргументов.

* 1. Вызов методов.

Метод не выполняется, пока его не вызовут. Чтобы вызвать метод, следует указать его имя и задать необходимые ему параметры.

// Так вызывают метод Rotate с двумя параметрами Rotate (45, "Degrees");

Метод Main - особый, он вызывается сразу после начала исполнения программы. К особым методам также относятся деструкторы, которые вызывают во время выполнения непосредственно перед уничтожением объекта. Третий вид особых методов - конструкторы; они исполняются во время инициализации объектов.

Метод может иметь один или несколько параметров. Параметр - это аргумент, передаваемый методу вызывающим его методом. При объявлении метода список параметров указывают после его имени в круглых скобках, при этом для каждого параметра необходимо указать его тип.

public void DisplayName (string name, byte age)

{

Console.WriteLine ("Hello " + name + ". You are " + age.ToString () +"years old.");

}

Пример 1. Задан вектор D(5). Создать класс vect осуществляющий ввод данных с клавиатуры, вывод элементов вектора на консоль, и перестановку первого по порядку элемента со k-тым отрицательным элементом вектора. Ограничение доступа в класс (снаружи можно вызывать только public части класса)

using System;

namespace ConsoleApplication1

{

class Program

{

class vect // класс векторов vect

{

double[ ] a; // объявление массива int n;

int n; // размер массива

public vect(int n)

// конструктор создаёт вектор

{

this.a = new double[n];

this.n = n;

}

public int vv()

// заполнение вектора целиком,

// клавиатурный ввод

{

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

{

string buf;

// строка символов

buf = Console.ReadLine();

// заполнение строки

a[i] = Convert.ToDouble(buf);

// преобразование строки в число

}

return (0);

}

public int vyv()

// вывод на консоль

// элементов вектора

{

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

Console.WriteLine("a[" + i + "] = " + a[i]);

return (0);

}

public int perest(int k)

// переставить 1 и k-ое отрицательное число

{

// ключ: 0 – если порядок

// - 1 – если нет

int k1=0;

// k больше размера вектора

if (k >= n) k1= -1;

else

{

// счетчик отрицательных чисел

int cnt = 0;

// указатель на позицию в массиве

int pos = -1;

// подсчёт отрицательных чисел

while (cnt < k)

{

pos++;

if (a[pos] < 0) cnt++;

} // подсчёт

tmp = a[pos] ; //перестановка

a[pos] = a[0];

a[0] = tmp;

k1=0;

}

return k1;

}

//главная программа

static void Main()

{

// создали

vect d = new vect (5);

d.vv (); // заполнили

d.vyv (); // вывели на консоль

d.perest (); // переставили

d.vyv (); // вывели на консоль

}

}

}

}

Пример 2. Задана матрица D(3;3) . Написать метод транспонирования матрицы и метод перемножения квадратных матриц. Получить результат перемножения транспонированной матрицы на соответствующую ей исходную. Отдельный класс для матриц не создаем. Работаем в классе myprog. Поэтому все содержимое класса доступно.

using System;

// моё пространство имён

namespace ConsoleApplication3

{

class myprog

{

static void Main()

{

// объявление матриц

double[,] d = new double[3,3];

double[,] dt = new double[3, 3];

double[,] ddt = new double[3, 3];

// заполнение d c клавиатуры

int i; int j; // строки, колонки

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

d[i, j] = Convert.ToDouble(Console.ReadLine());

}

// траспонирование матрицы

transp(d, 3, dt);

// умножение транпонированной

// матрицы dt на исходную d

mt\_q\_mult(dt, d,ddt,3);

}

// транпонирование

static void transp(double[,] mtr, int p, double[,] mtrt)

// double[,] mtr - так определяют матрицу в формальных параметрах

{

int i; int j;

for (i = 0; i < p; i++)

{

for (j = 0; j < p; j++)

mtrt[j, i] = mtr[i, j];

}

}

// умножение квадратных матриц

static void mt\_q\_mult(double[,] m\_a, double[,] m\_b, double[,] m\_r, int p)

// m\_r=m\_a \* m\_b , int p размер матриц

{

int i; int j; int k; double mel;

for (i = 0; i < p; i++) // строки m\_a

{

// столбцы m\_b

for (j = 0; j < p; j++)

{

mel=0;

// столбцы m\_a и строки m\_b

for (k = 0; k < p; k++)

mel=mel+m\_a[i,k]\*m\_b[k,j];

m\_r[i,j]=mel;

}

}

}

Литература

1. Архипов В. Н., Калядин В. И., Любин А. Н., Макаров А. И. Программирование на Фортране: методические указания к лабораторным работам. - М.; МГТУ «МАМИ», 2007, - 144 с.
2. Калядин В. И., Макаров А. И. Основы работы на персональном компьютере: сборник лабораторных работ. - М.; МГТУ «МАМИ», 2010, - 85 с.
3. Лобанов А. С., Туманова М. Б. Решение задач на языке Visual Basic for Application: учебное пособие. - М.; МГТУ «МАМИ», 2009, - 90 с.
4. Павловская Т. A. C# Программирование на языке высокого уровня. ПИТЕР, 2007, - 432 с.
5. Герберт Шмидт. Полное руководство C# 3.0. М.; Изд. дом «Вильяме», 2010, - 992.