

**Министерство образования и науки Республики Казахстан
Техническое и профессиональное образование**

Центральноазиатский технико-экономический колледж

Жаксыбаева Н.Н.

Основы языка программирования C#. Часть 1.

**Учебное пособие предназначено для учащихся
технического и профессионального образования**

Алматы, 2011

УДК 4857690
ББК длтаодлорлд
Ж.23

Рецензенты:

Баймухамбетова Ж.К. - доктор технических наук, доцент
Кубеков Б.С. заведующий кафедрой «Информационных технологий» университета «Туран», кандидат технических наук, доцент

Ж.23

Жаксыбаева Н.Н. Основы языка программирования C#: Учебное пособие.
Часть 1 - Алматы, 2011 - 205 стр.

ISBN hgo;jo;ifj

Пособие содержит теоретические, практические и справочные материалы по системе визуального объектно-ориентированного программирования C# в среде программирования Microsoft Visual Studio 2008. Рассмотрены особенности создания и использования приложений в консольном режиме. Даются примеры разработки прикладных программ, реализующих возможности C#. В первой главе рассмотрены основные структуры разработки приложений на языке C#. Вторая глава рассматривает вопросы по объектно-ориентированному программированию. В приложениях представлены задачи, а так же тестовые задания.

Данное пособие составлено в соответствии с типовой программой курса «Основы объектно-ориентированного программирования» и предназначено для обучения студентов колледжа, специальности 1304000 «Вычислительная техника и программное обеспечение», квалификации 1304043 «Техник-программист». В пособии кратко изложен теоретический материал, подробно разобраны решения типовых задач и примеров, а также задания по лабораторным работам.

УДК 4857690
ББК длтаодлорлд

Учебное пособие рассмотрено и утверждено на заседании ЦПК «Программное обеспечение и вычислительная техника» протоколом №5 от 08.01.2011 года Центральноазиатского технико-экономического колледжа.

Ж шпрожшщоршш
ISBN hgo;jo;ifj

© Жаксыбаева Н.Н., 2011

ВВЕДЕНИЕ

Начало современной эпохи программирования отмечено созданием языка С. Он был разработан Дэнисом Ритчи (Dennis Ritchie) в 1970-х годах для компьютера PDP-11 компании DEC (Digital Equipment Corporation), в котором использовалась операционная система UNIX. Несмотря на то что некоторые известные языки программирования, в особенности Pascal, достигли к тому времени значительного развития и признания, именно язык С определил направление сегодняшнего программирования.

Язык С вырос из кризиса программного обеспечения 1960-х годов и революционного перехода к *структурному программированию*. До структурного программирования многие программисты испытывали трудности при написании больших программ, поскольку обозначилась тенденция вырождения программной логики и появления так называемого "спагетти-кода" (spaghetti code) с большим размером процедур и интенсивным использованием оператора перехода goto. Такие программы были весьма трудны для изучения и модификаций. В структурных языках программирования эта проблема решалась посредством добавления точно определенных управляющих конструкций, вызова подпрограмм с локальными переменными и других усовершенствований. Структурные языки позволили писать довольно большие программы в приемлемые сроки.

Хотя в то время уже существовали другие структурные языки, С был первым языком, в котором удачно сочетались мощь, элегантность, гибкость и выразительность. Его лаконичный и к тому же простой в применении синтаксис в совокупности с философией, подразумевающей возложение ответственности на программиста (а не на язык), быстро завоевал множество сторонников. С точки зрения сегодняшнего дня, этот язык, возможно, несколько трудноват для понимания, но программистам того времени он показался порывом свежего ветра, которого они так долго ждали. В результате С стал самым популярным структурным языком программирования 1980-х годов.

Но многоуважаемый язык С имел ограничения. Одним из его недостатков была невозможность справиться с большими программами. Если проект достигал определенного размера, то дальнейшая его поддержка и развитие были связаны с определенными трудностями. Местоположение этой "точки насыщения" зависело от конкретной программы, программиста и используемых им средств, но вероятность ее достижения очень возрастала, когда количество строк в программе приближалось к 5 000.

К концу 1970-х размер проектов стал приближаться к критическому, при превышении которого методика структурного программирования и язык С "опускали руки". Поэтому стали появляться новые подходы к программированию, позволяющие решить эту проблему. Один из них получил название *объектно-ориентированного программирования* (ООП).

Используя ООП, программист мог справляться с программами гораздо большего размера, чем прежде. Но вся беда состояла в том, что С, самый популярный на то время язык, не поддерживал ООП. Желание работать с объектно-ориентированной версией языка С в конце концов и привело к созданию С++.

Язык С++ был разработан Бьярни Страуструпом (Bjarne Stroustrup) в компании Bell Laboratories (Муррей Хил, Нью-Джерси), и годом создания считается 1979-й. Первоначально создатель нового языка назвал его "С с классами", но в 1983 году это имя было изменено на С++. С++ полностью включает элементы языка С. Таким образом, С можно считать фундаментом, на котором построен С++. Большинство дополнений, которые Страуструп внес в С, были предназначены для поддержки объектно-ориентированного программирования. По сути, С++ — это объектно-ориентированная версия языка С. Возводя "здание" С++ на фундаменте С, Страуструп обеспечил плавный переход многих программистов на "рельсы" ООП. Вместо необходимости изучать совершенно новый язык, С-программисту достаточно было освоить лишь новые средства, позволяющие использовать преимущества объектно-ориентированной методики.

На протяжении 1980-х годов С++ интенсивно развивался и к началу 1990-х уже был готов для широкого использования. Рост его популярности носил взрывоподобный характер, и к концу этого десятилетия он стал самым широко используемым языком программирования. В наши дни язык С++ по-прежнему имеет неоспоримое превосходство при разработке высокопроизводительных программ системного уровня.

Важно понимать, что создание С++ не было попыткой изобрести совершенно новый язык программирования. Это было своего рода усовершенствование и без того очень успешного языка. Такой подход к разработке языков (взять за основу существующий язык и поднять его на новую ступень развития) дал начало тенденции, которая продолжает жить и сегодня.

Следующей ступенью на лестнице прогресса языков программирования стал язык Java, который первоначально назывался Oak (в переводе с англ. "дуб"). Работа над его созданием началась в 1991 году в компании Sun Microsystems. Основной движущей силой разработки Java был Джеймс Гослинг (James Gosling). В его рабочую группу входили Патрик Нотон (Patrick Naughton), Крис Уортс (Chris Warth), Эд Фрэнк (Ed Frank) и Майк Шеридан (Mike Sheridan).

Java — это структурный объектно-ориентированный язык программирования, синтаксис и основополагающие принципы которого "родом" из С++. Своими новаторскими аспектами Java обязан не столько прогрессу в искусстве программирования (хотя и это имело место), сколько изменениям в компьютерной среде. Еще до наступления эры Internet большинство программ писалось, компилировалось и предназначалось для выполнения с использованием определенного процессора и под управлением

конкретной операционной системы. Несмотря на то что программисты всегда старались делать свои программы так, чтобы их можно было применять неоднократно, возможность легко переносить программу из одной среды в другую не была еще достигнута, к тому же проблема переносимости постоянно отодвигалась, решались же более насущные проблемы. Однако с появлением всемирной сети Internet, в которой оказались связанными различные типы процессоров и операционных систем, старая проблема переносимости заявила о себе уже в полный голос. Для ее решения понадобился новый язык программирования, и им стал Java,

Разработчики Java успешно решили многие проблемы, связанные с переносимостью в среде Internet, но далеко не все. Одна из них — *межъязыковая возможность взаимодействия* (cross-language interoperability) программных и аппаратных изделий разных поставщиков, или *многоязыковое программирование* (mixed-language programming). В случае решения этой проблемы программы, написанные на разных языках, могли бы успешно работать одна с другой. Такое взаимодействие необходимо для создания больших систем с распределенным программным обеспечением (ПО), а также для программирования компонентов ПО, поскольку самым ценным является компонент, который можно использовать в широком диапазоне компьютерных языков и операционных сред.

Windows. Хотя Java-программы могут выполняться в среде Windows (при условии установки виртуальной машины Java), Java и Windows не являются прочно связанными средами. А поскольку Windows — это наиболее широко используемая операционная система в мире, то отсутствие прямой поддержки Windows — серьезный недостаток Java.

Чтобы удовлетворить эти потребности, Microsoft разработала язык C#, C# был создан в конце 1990-х годов и стал частью общей .NET-стратегии Microsoft. Впервые он увидел свет в качестве а-версии в середине 2000 года. Главным архитектором C# был Андерс Хейлсберг (Anders Hejlsberg) — один из ведущих специалистов в области языков программирования, получивший признание во всем мире. Достаточно сказать, что в 1980-х он был автором весьма успешного продукта Turbo Pascal, изящная реализация которого установила стандарт для всех будущих компиляторов.

C# непосредственно связан с C, C++ и Java. И это не случайно. Эти три языка — самые популярные и самые любимые языки программирования в мире. Более того, почти все профессиональные программисты сегодня знают C и C++, и большинство знает Java. Поскольку C# построен на прочном, понятном фундаменте, то переход от этих "фундаментальных" языков к "надстройке" происходит без особых усилий со стороны программистов. Так как Андерс Хейлсберг не собирался изобретать свое "колесо", он сосредоточился на введении усовершенствований и новшеств.

Генеалогическое дерево C# показано на рис. 1.1. "Дедушкой" C# является язык C. От C язык C# унаследовал синтаксис, многие ключевые слова и операторы. Кроме того, C# построен на улучшенной объектной

модели, определенной в C++. Если вы знаете C или C++, то с C# вы сразу станете друзьями.

C# и Java связаны между собой несколько сложнее. Как упоминалось выше, Java также является потомком C и C++. У него тоже общий с ними синтаксис и сходная объектная модель. Подобно Java C# предназначен для создания переносимого кода. Однако C# — не потомок Java. Скорее C# и Java можно считать двоюродными братьями, имеющими общих предков, но получившими от родителей разные наборы "генов". Если вы знаете язык Java, то вам будут знакомы многие понятия C#. И наоборот, если в будущем вам придется изучать Java, то, познакомившись с C#, вам не придется осваивать многие средства Java.

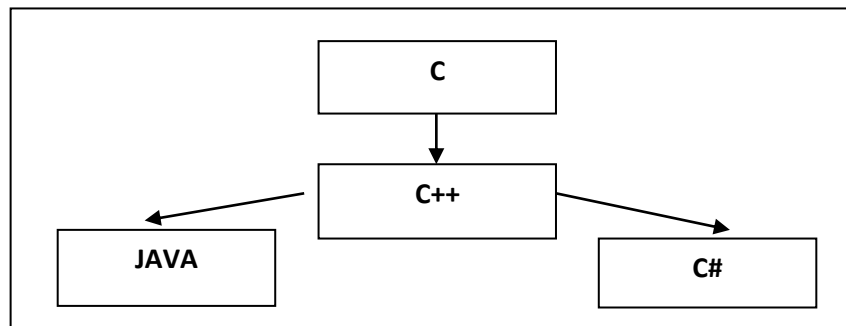


Рисунок 1.1- Генеалогическое дерево

C# содержит множество новых средств, которые описаны в этой книге. Самые важные из них связаны со встроенной поддержкой программных компонентов. Именно наличие встроенных средств написания программных компонентов и позволило C# называться *компонентно-ориентированным языком*. Например, C# включает средства, которые напрямую поддерживают составные части компонентов: свойства, методы и события. Все же самым важным качеством компонентно-ориентированного языка является его способность работать в среде многоязыкового программирования.

ГЛАВА 1.

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C#

1.1 Элементы языка C#

Множество символов языка C# включает прописные и строчные буквы латинского алфавита и цифры. Буквы и цифры используются для формирования идентификаторов, констант, ключевых слов.

▼ C# чувствителен к регистру.

Совокупность ключевых слов (таблица 1.1) составляет словарь языка. Все зарезервированные слова содержат только строчные буквы (символы нижнего регистра) и написаны на английском языке (в том числе с сокращениями).

▼ Ключевые слова запрещено использовать в любом качестве.

Таблица 1.1 – Ключевые слова C#

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	forage	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	volume	virtual
volatile	void	while		

Нелатинский алфавит допускается только в комментариях, внутри символьных строк между двойными кавычками и для символьных переменных между одинарными кавычками.

Пробел используется не только в качестве разделителя лексем языка в тексте программы, но и как символьный знак. Компилятор не реагирует на дополнительные, «лишние», пробелы в тексте программы, поэтому их можно вводить для придания тексту наглядности.

1.2 Переменные

1.2.1 Типы данных

Во всех языках программирования для обозначения типа данных используются специальные ключевые слова, и язык С# не является исключением.

Заметим, что в отличие от других языков программирования все переменные в С# рассматриваются как объекты тех или иных классов. Пока мы еще не приступили к изучению объектно-ориентированного программирования, не будем принимать данное обстоятельство во внимание. Позже мы вернемся к этому вопросу и рассмотрим его во всех подробностях.

В языке С# можно использовать числовые типы данных со знаком, числовые переменные без знака, символьные переменные, логические (булевы) переменные и строки, а также перечисления. Рассмотрим элементарные типы данных С# и их обозначение подробнее.

Каждая переменная имеет имя, тип, размер и значение (таблица 1.2).

Таблица 1.2 – Описание переменных

Имя	Наименование переменной, построенное по правилу языка: содержит комбинацию латинских букв, знака подчеркивания и цифр (не в начале имени). Пример: Error_Int и Div12_TR
Тип	Определяет какие символы и числа записаны в ячейку памяти под этим именем
Размер	Связан с объявлением типа, определяет объем памяти (максимальную величину или точность задания числа)
Значение	Определяет конкретное содержимое ячейки памяти

Переменная должна быть объявлена в программе до первого ее использования. Изменить тип переменной нельзя.

Переменной может присваивается константа (значение) только определенного типа, что фиксируется в объявлении. Нельзя производить действия с переменной, не имеющей конкретного значения.

Числа без знака.

Вы уже знаете, что если все разряды байта используются для представления абсолютного значения числа, то байт хранит только положительные числа и нуль. В таблице 1.3 мы перечислили имена числовых типов данных без знака, используемые в языке программирования С#.

Таблица 1.3 - Числовые типы данных без знака

Тип	Возможные значения	Описание
byte	от 0 до 255	8-разрядное значение без знака, занимает 1 байт памяти
ushort	от 0 до 65 535	16-разрядное значение без знака, занимает 2 байта памяти
uint	от 0 до 4 294 967 295	32-разрядное значение без знака, занимает 4 байта памяти
ulong	от 0 до 18 446 744 073 709 551 615	64-разрядное значение без знака, занимает 8 байт памяти

Чтобы использовать переменные в программе С#, их необходимо объявить, указав имя и тип переменной, например:

```
byte    myByte;
ushort myShort;
uint    myInt;
ulong   myLong;
```

Здесь объявлены 4 переменные с именами **myByte**, **myShort**, **myInt** и **myLong**. Обратите внимание, что записаны объявления каждой переменной на отдельной строке, завершив его символом «точка с запятой».

Если объявляются несколько переменных одного типа, можно использовать список имен переменных, разделенных запятыми, например:

```
ushort xCoord, yCoord, zCoord;
uint height, width;
```

Здесь мы объявили 3 переменные **xCoord**, **yCoord** и **zCoord** типа **ushort**, а также две переменные **height** и **width** типа **uint**.

Транслятор С#, преобразуя исходный текст программы в промежуточный код MSIL, о котором мы упоминали во Введении, игнорирует незначащие пробелы, символы табуляции, символы возврата каретки и перехода на другую строку. Однако если не ставить пробел между обозначением типа переменной и ее именем, компилятор будет считать такую строку ошибочной:

```
byte myByte; // Ошибка!
```

Обратите внимание, что в этой строке после символов «**//**» мы записали так называемую строку **комментария**. Комментарии позволяют вставлять в исходный текст программы произвольные поясняющие текстовые строки. В языке С# существуют комментарии трех типов. На

данный момент вам следует знать, что компилятор игнорирует символы // и все символы, следующие за ними до конца текущей строки:

byte myByte; // Это правильное определение переменной типа byte

Если нужно вставить комментарий в середину строки, используется конструкция /*...*/. Текст комментария помещается между парой символов /*, открывающей комментарий, и парой символов */, закрывающей комментарий.

Например:

byte /* это комментарий*/ myByte;

С помощью конструкции /*...*/ можно создавать многострочные комментарии, включающие в себя при необходимости однострочные комментарии. В частности, данный фрагмент исходного текста программы будет рассматриваться компилятором как многострочный комментарий:

```
/*
Это переменные для хранения чисел без знака
byte    myByte; // это байт
ushort  myShort;
uint     myInt;
ulong    myLong;
*/
```

Использование комментариев делает программу более понятной. Не жалейте времени на комментирование программ, особенно сложных. Впоследствии, если вам придется дорабатывать программу, комментарии помогут вспомнить назначение фрагментов ее исходного текста.

Числа со знаком.

Числовые типы данных со знаком, доступные программисту C#, представлены в таблице 1.4. Как видите, при помощи этих типов данных можно представить числа в довольно широком диапазоне значений.

Таблица 1.4. Числовые типы данных со знаком

Тип	Возможные значения	Описание
sbyte	от -128 до 127	8-разрядное значение без знака, занимает 1 байт памяти
short	от -32 768 до 32 767	16-разрядное значение без знака, занимает 2 байта памяти
int	от -2 147 483 648	32-разрядное значение без

	до 2 147 483 647	знака, занимает 4 байта памяти
long	от -9×10^{18} до 9×10^{18}	64-разрядное значение без знака, занимает 8 байт памяти

Вот примеры объявления переменных для перечисленных в этой таблице типов данных со знаком:

```
sbyte distanceX;
short distanceY;
int height;
long width;
```

Сделаем небольшое замечание относительно выбора имен переменных.

Прежде всего, имена нужно задавать таким образом, чтобы в контексте программы было понятно, для чего эти переменные используются и что они хранят.

Компилятор C# не запрещает указывать такие имена переменных, как x, y, ab, d и т. п. Однако смысл имени переменной будет намного прозрачнее, если составлять его из слов английского языка, обозначающих связанное с этой переменной понятие. Например, по имени **distanceFromHome** сразу можно догадаться, что в соответствующей переменной хранится расстояние от дома.

Кроме того, рекомендуется (хотя это и не обязательно) начинать имя переменной со строчной буквы и применять прописные буквы для выделения слов, входящих в составное имя переменной.

Числа с плавающей точкой.

При проведении научных вычислений часто используются числа в формате с плавающей точкой, такие, например, как $2,1 \times 10^{10}$.

Для представления данных в формате с плавающей точкой в языке C# используется два типа данных— **float** и **double** (таблица 1.5).

Таблица 1.5- Числовые типы данных с плавающей точкой

Тип	Возможные значения	Описание
float	от -1.5×10^{-45} до 3.4×10^{38}	32-разрядное число с плавающей точкой, максимальная точность представления чисел — 7 десятичных цифр
double	от 5.0×10^{-324} до 1.7×10^{308}	64-разрядное число с плавающей точкой, максимальная точность представления чисел — 16 десятичных цифр

Вот примеры объявления переменных, предназначенных для хранения чисел с плавающей точкой, в программе C#:

float piNumber; double eNumber;

Как видите, диапазон значений, представляемых с помощью типов данных **float** и **double**, чрезвычайно широк и вполне подходит для научных расчетов.

Числа для финансистов

Деньги, как известно, любят счет. Если речь идет о компьютерных системах банка, оперирующих с астрономическими денежными суммами, необходимо обеспечить максимально возможную точность вычислений.

Многократное возникновение ошибки округления может привести к бесследному исчезновению (или, наоборот, к появлению из ниоткуда) довольно заметных денежных сумм.

Из прессы нам известен случай, когда недобросовестный программист пользовался данным обстоятельством с выгодой для себя. Мизерные денежные суммы, отбрасываемые при округлении результатов вычислений, он переводил на свой счет, проводя в жизнь известное утверждение о том, что копейка рубль бережет. Когда махинации вскрылись, оказалось, что на счету злоумышленника скопилась изрядная сумма денег.

В языке C# предусмотрен один тип данных специально для работы с деньгами — **decimal** (таблица 1.6). Он обеспечивает колоссальную точность представления денежных сумм — до 29 десятичных цифр!

Таблица 1.6- Числовые типы данных для работы с деньгами

Тип	Возможные значения	Описание
decimal	От 1.0×10^{-28} до 7.9×10^{28}	128-разрядное число с плавающей точкой, максимальная точность представления чисел — 29 десятичных цифр

Вот пример объявления переменной типа **decimal**:

decimal totalAccountValue;

Текстовые символы

Для хранения отдельных символов текста (таких, как буквы, знаки пунктуации и управляющие символы) в языке C# предусмотрен специальный тип данных

char.

Как мы уже говорили, для этого типа данных используется кодировка UNICODE, поэтому символы **char** занимают в памяти 2 байта. Заметим, что переменные типа **char** в языках программирования C и C++ занимают в памяти только 1 байт.

Вот как можно объявить переменную типа **char** в программе, написанной на C#:

char tokenDelemiter;

Текстовые строки

Многие программы работают с текстовыми строками. Язык C# содержит мощные и удобные объектно-ориентированные средства, позволяющие выполнять со строками все необходимые операции.

Про строки мы пока скажем лишь то, что они определяются с помощью ключевого слова `string` и хранятся в кодировке UNICODE. Детальное описание приемов работы со строками мы приведем после того, как рассмотрим технологию объектно-ориентированного программирования.

Вот так можно объявить в программе C# переменную типа `string`:

string firstName;

Логический тип данных

Логические переменные в C# объявляются с помощью ключевого слова **bool** и могут принимать одно из двух значений — `true` (истина) или `false` (ложь).

В этом логические переменные напоминают отдельные разряды байта. Действительно, каждый бит может хранить только одно из двух значений — 1 или 0. Во многих языках программирования (в частности, в языках C и C++) значение 1 сопоставляется с истиной, а значение 0 — с ложью. В языке C# такое сопоставление не используется. Подробнее об этом мы поговорим позже при описании условных операторов.

Перечисления

Для определения некоторых типов данных вместо чисел, текстовых символов и строк больше подходят перечисления. Например, мы можем пронумеровать дни недели и ссылаться в программах на номера, но проще использовать привычные для нас названия — понедельник, вторник, среда и т. д.

В языке C# перечислимый тип данных задается при помощи ключевого слова `enum` и фигурных скобок, например:

```
enum Days  
{ Monday, Tuesday, Wednesday, Thursday,  
Friday, Saturday, Sunday  
}
```

Здесь с помощью ключевого слова `enum` мы создали собственный перечислимый тип данных `Days` и определили составляющие его элементы перечисления, расположив их внутри фигурных скобок.

Так как компилятор C# игнорирует незначачие пробелы и символы новой строки, в тексте программы все элементы перечисления могут находиться и на одной строке. Например, ниже определен перечислимый тип

для представления основных компонентов цвета — красного, зеленого и голубого:

enum BaseColors { Red, Green, Blue }

Компилятор C# автоматически ставит в соответствие элементам перечисления целочисленные значения, однако программист может оперировать именами переменных, что намного удобнее.

Обратите внимание, что оператор объявления переменной нужно завершать точкой с запятой. При объявлении перечисления этот символ не используется.

По умолчанию перечисления создаются на базе типа данных **int**. Однако вы можете создавать перечисления и других типов. Вот полный список типов, которые могут служить базовыми для перечислений: **byte, ushort, uint, ulong, sbyte, short, int, long**.

Примеры описания переменных:

1. описание переменных целого типа:

int k, Kol_Per=23;

sbyte Diam=-34;

uint=124585, i=0;

2. описание переменных вещественного типа:

double a=453.0, Treyg = -0.765;

3. (префиксы **f** и **F** необходимы при описании переменных **float**)

float w=6575.0f, ww=0.56F;

4. (префиксы **m** и **M** необходимы при описании переменных **decimal**)

decimal cost=1467.98m, col=2564M;

bool b=true, b2=false; (описание переменных логического типа).

Литералы

Литералы применяются в исходном тексте программы для обозначения числовых или логических значений, текстовых символов и строк. С помощью литералов программист может присвоить начальные значения переменным.

Целочисленные литералы

Для обозначения целых десятичных чисел в языке C# используются цифры от 0 до 9, а также (при необходимости) дополнительные суффиксы. Вот примеры простейших целочисленных литералов:

25 0 767 6786867867 1233

Целочисленные литералы без дополнительных суффиксов могут представлять значения типов со знаком или без знака: **byte, sbyte, int, uint, long, ulong**.

Чтобы указать, что литерал представляет собой число без знака, он снабжается суффиксом **u** или **U**, например:

256u 1U 0U

В этом случае литерал соответствует типам **byte, uint, ulong**.

Суффиксы **l** или **L** применяются для создания литералов типа **long** и **ulong**:

2564124739362741 11 0L

Комбинируя в произвольном порядке суффиксы **u, U, l** и **L**, можно создавать литералы типа **ulong**:

25UL 145637365UL 76787234711u 7678723471LU 6410193764LU

На экране букву **l** очень легко перепутать с цифрой **1**, поэтому в суффиксах литералов рекомендуется применять вместо нее прописную букву **L**.

Целочисленные литералы могут обозначать и шестнадцатеричные числа. В этом случае они состояются из цифр от 0 до 9 и букв **A, B, C, D, E, F**. Перед шестнадцатеричным числом помещается префикс **0x**, например:

0x1 0x23BF 0xFFFF 0xFE67BCA0001

Для обозначения знака числа допускается использование символов **+** и **-**.

Литералы с плавающей точкой.

Литералы с плавающей точкой бывают типа **float, double** и **decimal**. Если в литерале используется суффикс **f** или **F**, то это литерал типа **float**, а если суффикс **d** или **D** — типа **double**. В том случае, когда суффикс не указан, литерал принимает тип **double**.

Для обозначения литерала типа **decimal** используется суффикс **m** или **M**. Экспоненциальная часть литерала отделяется символом **e** или **E**. Для обозначения знака числа используются символы **+** и **-**.

Вот примеры литералов с плавающей точкой типа **float**:

3F 6.52f 2e10F 1356.4560f -3.7E-10F

Ниже мы привели примеры литералов типа **double**:

.25 3.3 6.52d 2e10 1356.4560D -3.7E-10D

Вот несколько примеров литералов типа **decimal**, применяемых обычно для обозначения денежных сумм:

.25m 3.3M 6.52t 2e10M 13573847420983746.45M

Символьные литералы

Обычно с помощью символьных литералов представляют одиночные символы, заключая их в одинарные кавычки, например:

'С' 'А' 'Щ'

С помощью символа обратного следа (\) можно указать в качестве символьного литерала некоторые специальные и управляющие символы. Например, литерал '\n' задаст символ новой строки.

Последовательность символа \ и идущего вслед за ним символа называют **escape-последовательностью**. В таблице 1.7 мы привели escape-последовательности, допустимые в языке C#, а также соответствующие им 16-разрядные значения в кодировке UNICODE.

Таблица 1.7 - Escape-последовательности

Escape-последовательность	Символ	Код, UNICODE
\'	Одинарная кавычка(')	0x0027
\"	Двойная кавычка(")	0x0022
\\	Обратный слеш (\)	0x005C
\0	Нулевое значение	0x0000
\a	Звуковой сигнал	0x0007
\b	Возврат каретки	0x0008
\f	Новая страница	0x000C
\n	Новая строка	0x000A
\r	Возврат каретки	0x000D
\t	Горизонтальная табуляция	0x0009
\v	Вертикальная табуляция	0x000B
\x	Произвольное значение в шестнадцатеричной кодировке	От 0x0000 до 0xFFFF

Строковые литералы

Для представления текстовых строк в языке программирования C# используют строковые литералы. Строковый литерал представляет собой простую строку символов, заключенную в двойные кавычки, например:

"Hello, C# world!"

Строковые литералы могут включать в себя escape-последовательности, описанные в предыдущем разделе. Например, для разделения слов в строковом литерале может применяться символ табуляции \t:

"Случайные числа:\t657\t3\t4585\t54545"

Для того чтобы включить в литерал символ двойной кавычки, его нужно представить в виде escape-последовательности \", например:

"Гостиница \"Казахстан\""

Если же нужно включить в литерал символ обратного слеша (\), то его нужно повторить дважды. Вот, например, как с помощью текстового литерала можно задать путь к каталогу c:\games\tetris\doc:

"c:\\games\\tetris\\doc"

В языке C# строковый литерал может иметь префикс @. Он задает так называемый **буквальный** или **дословный** (verbatim) **литерал**. В таком литерале все символы интерпретируются буквально, поэтому, например, нет необходимости повторять дважды символ обратного слеша:

@ "c:\games\tetris\doc"

Если же нужно включить в буквальный строковый литерал символ двойной кавычки, его все же придется повторить дважды:

О'Тостиница ""Казахстан""

Обычные строковые литералы C++ должны полностью располагаться на одной строке. Что же касается буквальных строковых литералов, то они могут быть не только однострочными, но и многострочными, например:

**@ "Это
многострочный литерал"**

Такой литерал включает в себя все пробелы, символы новой строки и любые другие символы, располагающиеся между двойными кавычками, ограничивающими литерал.

Логические литералы

Логические литералы предназначены для задания значений логическим переменным. В языке C# используются два логических литерала — true (истина) и false (ложь).

Напомним, что в языке C# логические литералы не сопоставляются с целочисленными значениями, такими, например, как 1 и 0.

Литерал null

В языках программирования C и C++ переменные всегда содержат какие-либо значения. Программист может задать эти значения явно с помощью оператора присваивания (который мы скоро рассмотрим), а может и не задавать. В последнем случае в переменной будет храниться либо нуль, либо случайное значение.

Что же касается языка C#, то в нем с помощью специального литерала null программист может указать, что переменная вообще не содержит никакого значения.

Заметим, что литерал null не эквивалентен нулевому значению. Он предназначен для использования в тех ситуациях, когда в переменной не хранится ни нулевое, ни какое-либо другое значение.

Константы.

Добавление к описанию переменной атрибута const (постоянный) означает, что значение переменной нельзя изменить после ее объявления (с

обязательной инициализацией). При попытке в тексте программы изменить значение константы компилятор фиксирует ошибку.

Например: `const double f=5.456;`

Приоритет выполнения операций: умножение, деление и вычисление остатка, потом вложение и вычитание.

Например: `d=a = r * y / s % a-1;`

Операторы

После ознакомления с типами данных, переменными и литералами необходимо научиться инициализировать переменные при помощи литералов, а также изучить основные операторы языка C#.

Операторы представляют собой символы, с помощью которых программа может выполнять те или иные действия над переменными, литералами и другими объектами. В результате выполнения оператора всегда получается какое-то значение, представляющее собой результат выполнения оператора.

Объекты программы, над которыми выполняются действия, называются **операндами**. Например, оператор сложения чисел принимает два операнда. Результатом сложения будет число, равное сумме значений операндов, т. е. сумме складываемых чисел.

Операторы и операнды формируют **выражения**.

Изучение операторов и выражений языка C# мы начнем с фундаментального оператора присваивания. Далее мы рассмотрим арифметические, логические, условные и другие операторы, а также выражения, составляемые с их применением.

Оператор присваивания

Оператор присваивания обозначается одинарным знаком равенства (=). Его роль в языке C# во многом такая же, как и в других языках программирования. Общая форма записи оператора присваивания имеет следующий вид:

Тип переменная = выражение;

Здесь тип элемента *переменная* должен быть совместим с типом элемента *выражение*. Оператор присваивания интересен тем, что позволяет создавать целую цепочку присвоений. Рассмотрим, например, следующий фрагмент кода.

```
int x, y, z; x = y = z = 100;
```

```
// Устанавливаем переменные x, y и z равными 100.
```

Составные операторы присваивания

В C# предусмотрены специальные составные операторы присваивания, которые упрощают программирование определенных инструкций присваивания. Лучше всего начать с примера.

Рассмотрим следующую инструкцию:

$$x = x + 10;$$

Используя составной оператор присваивания, ее можно переписать в таком виде:

$$x += 10;$$

Пара операторов `+=` служит указанием компилятору присвоить переменной `x` сумму текущего значения переменной `x` и числа 10. А вот еще один пример.

Инструкция

$$x = x - 100;$$

аналогична такой: $x -= 100;$

Обе эти инструкции присваивают переменной `x` ее прежнее значение, уменьшенное на 100.

Составные версии операторов присваивания существуют для всех бинарных операторов (т.е. для всех операторов, которые работают с двумя операндами). Общая форма их записи такова:

переменная op = выражение;

Здесь элемент *op* означает конкретный арифметический или логический оператор, объединяемый с оператором присваивания.

Возможны следующие варианты объединения операторов.

<code>+=</code>	<code>-=</code>	<code>*=</code>	<code>/=</code>
<code>%=</code>	<code>&=</code>	<code> =</code>	

Поскольку составные операторы присваивания выглядят короче своих несоставных эквивалентов, то составные версии часто называют *укороченными операторами присваивания*.

Составные операторы присваивания обладают двумя заметными достоинствами. Во-первых, они компактнее своих "длинных" эквивалентов. Во-вторых, их наличие приводит к созданию более эффективного кода (поскольку операнд в этом случае вычисляется только один раз). Поэтому в профессионально написанных сопрограммах вы часто встретите именно составные операторы присваивания.

Операторы ввода и вывода

Существуют 2 операторов ввода и вывода в консольном приложении на языке C#. Они относятся к классу Console.

Операторы ввода:

Read – считывание символа или числа

ReadLine – считывание строки

Операторы вывода:

Write – вывод строки без переноса курсора на следующую строку

WriteLine - вывод строки с переносом курсора на следующую строку

Для ввода числа необходимо знать функции преобразования строки в число (класс Convert) в таблице 1.8.

Таблица 1.8 – Функции преобразования

№	Название	Назначение
1.	ToDouble	Преобразование строки в вещественное число типа double
2.	ToInt16	Преобразование строки в целое число
3.	ToInt32	Преобразование строки в целое число
4.	ToFloat	Преобразование строки в вещественное число типа float

Примеры:

```
int k = Convert.ToInt32 (Console.Read());
```

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

```
Console.Write(“Введите число”);
```

```
Console.WriteLine(k +”тенге”);
```

```
double m = Convert.ToDouble (Console.ReadLine());
```

1.2.2 Арифметические операции

C# использует операторы, то есть знаки арифметических и логических операций, а также другие знаки (см. табл. 1.9).

Таблица 1.9 - Операции

Операция	Знак	Операция	Знак
Арифметическое сложение	+	Проверка на равенство:	
Унарный плюс	+	равно	==
Арифметическое вычитание	-	не равно	!=
Унарный минус	-	Проверка отношения:	

Умножение	*	больше меньше	>
Деление	/	больше или равно	>=
Вычисление остатка	%	меньше или равно	<=
Присваивание	=	Отрицание (не) not	!
Декремент --k (то же что и k=k- 1)	--	Тернарная	? :
Инкремент k++ (то же что и k=k+1)	++	Проверка условия И (AND) ИЛИ (OR)	&&

Операция, определяемая оператором % (вычисление остатка, целочисленное деление, деление по модулю), вычисляет целый остаток от деления двух чисел.

Например:

$13/3 = 4$ (дробная часть отбрасывается), но $13\%3 = 1$

$13.0 / 3.0 = 4.333333$, но $13.0 \% 3.0 = 1$

Операторы инкремента и декремента

Операторы инкремента (++) и декремента (--) увеличивают и уменьшают значение операнда на единицу, соответственно. Как будет показано ниже, эти операторы обладают специальными свойствами, которые делают их весьма интересными для рассмотрения.

Итак, оператор инкремента выполняет сложение операнда с числом 1, а оператор декремента вычитает 1 из своего операнда. Это значит, что инструкция

$x = x + 1$; аналогична такой инструкции: $x++$;

Точно так же инструкция

$x = x - 1$; аналогична такой инструкции: $x--$;

Операторы инкремента и декремента могут стоять как перед своим операндом, так и после него.

Например, инструкцию

$x = x + 1$;

можно переписать в виде префиксной формы:

$++x$; // Префиксная форма оператора инкремента, или в виде постфиксной формы:

$x++$; // Постфиксная форма оператора инкремента.

В предыдущем примере не имело значения, в какой форме был применен оператор инкремента: префиксной или постфиксной. Но если оператор инкремента или декремента используется как часть большего выражения, то форма его применения имеет важное значение. Если такой оператор применен в префиксной форме, то C# сначала выполнит эту операцию, чтобы операнд получил новое значение, которое затем будет

использовано остальной частью выражения. Если же оператор применен в постфиксной форме, то С# использует в выражении его старое значение, а затем выполнит операцию, в результате которой операнд обретет новое значение.

Рассмотрим следующий фрагмент кода:

```
x = 10; y = ++x;
```

В этом случае переменная у будет установлена равной 11. Но если в этом коде префиксную форму записи заменить постфиксной, переменная у будет установлена равной 10, обоих случаях переменная х получит значение 11. Разница состоит лишь в том, в какой момент она станет равной 11 (до присвоения ее значения переменной у или после).

1.2.3 Логические операции

Операции отношения (сравнения) и логические предполагают проверку некоторого условия с последующим действием, выполнение которого зависит от результата проверки.

В С# на равенство (==) и неравенство (!=) можно проверять все переменные, но **операторы сравнения** можно применять **только к переменным числовых типов и типа char**.

Решение может приниматься в результате анализа не обязательно одного условия. Условия связываются знаками логического сложения && (И) или логического умножения || (ИЛИ).

Например: если i равно j **И** k не равно 5

```
if (i==j && k!=5)
```

если i меньше j **ИЛИ** k больше или равно 5

```
if (i<j || k>=5)
```

1.2.4 Математические операции

В таблице 1.10 представлены функции для расчета математических операций.

Таблица 1.10 – Математические функции

Имя функции, константы	Назначение	Вызов
Abs	Возвращение абсолютной величины числа x, может принимать как вещественное, так и целое значение	double x=-5,y; y=Math.Abs(x); // y=5 int x1,y1;

		<code>y1= Math.Abs(x1);</code>
Acos	Возвращение арккосинуса числа x (в диапазоне от -1 до 1)	<code>double x,y; y= Math.Acos(x);</code>
Asin	Возвращение арксинуса числа x (в диапазоне от -1 до 1)	<code>double x,y; y= Math.Asin(x);</code>
Atan	Возвращение арктангенса числа x	<code>double x,y; y= Math.Atan(x);</code>
Atan2	Возвращение частного y/x	<code>double x,y,z; z= Math.Atan(y,x);</code>
Celling	Возвращение ближайшего целого (y), большего, чем значение аргумента (x) Например: $x=1.04$; $y=2$; $x= -1.04$; $y=-1$;	<code>double x,y; y= Math.Ceiling(x);</code>
Cos	Возвращение косинуса числа x	<code>double x,y; y= Math.Cos(x);</code>
E — константа	Основание натурального логарифма $e=2.71828128459$	<code>double e= Math.E;</code>
Exp	Возвращение основания натур, логарифма E , возведенное в степень x	<code>double x,y; y=Exp(x);</code>
Floor	Возвращение ближайшего целого (y), меньшего, чем значение аргумента (x). Например: $x=1.04$; $y=1$; $x= -1.04$; $y= -2$;	<code>double x,y; y= Math.Floor(x);</code>
Log	Возвращение натурального логарифма x	<code>double x,y; y= Math.Log(x);</code>
	Возвращение логарифма x по основанию a	<code>double x,y,a; y= Math.Log(x,a);</code>
Log 10	Возвращение десятичного логарифма x	<code>double x,y; y=Math.Log10(x);</code>
Max	Возвращает большего из значений x и y	<code>double x,y; y= Math.Max(x,y);</code>
		<code>char x,y; ys(char) Math.Max(x1,y1);</code>
Min	Возвращение меньшего из значений x и y	<code>double x,y; y= Math.Min(x,y);</code>
		<code>char x,y; yss(char) Math.Min(x,y);</code>
PI — константа	Число $\pi = 3.1415926535898$	<code>double pi= Math.PI;</code>

Pow	Возвращение значения “a” в степени “exp”	double a,exp,y; y= Math.Pow(a,exp);
Round	Возвращение значения a, округленного до ближайшего целого (арифметич. округление)	double a,b; b= Math.Round(a);
Sign	Определение знака числа. Возвращает: - 1, если значение аргумента отрицательное, 1, если значение аргумента положительное, 0, если значение аргумента равно нулю	double x,y; y= Math.Sign(x);
Sin	Возвращение синуса x	double x,y; y= Math.Sin(x);
Sqrt	Возвращение квадратного корня x	double x,y; y= Math.Sqrt(x);
Tan	Возвращение значения тангенса числа x	double x,y; y= Math.Tan(x);

Пример: Найти значение $y=\cos(x)$.

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Введите вещественное число = ");
        string s = Console.ReadLine();           // ввод строки
        // преобразование строки s в вещественное число x
        double x = Convert.ToDouble(s);
        double y;                                // описание переменной y
        y = Math.Cos(x);                         // расчет значения y
        Console.WriteLine("Значение y при "+ x +" равно = "+y); // вывод
        Console.ReadKey();                       // задержка экрана
    }
}
```

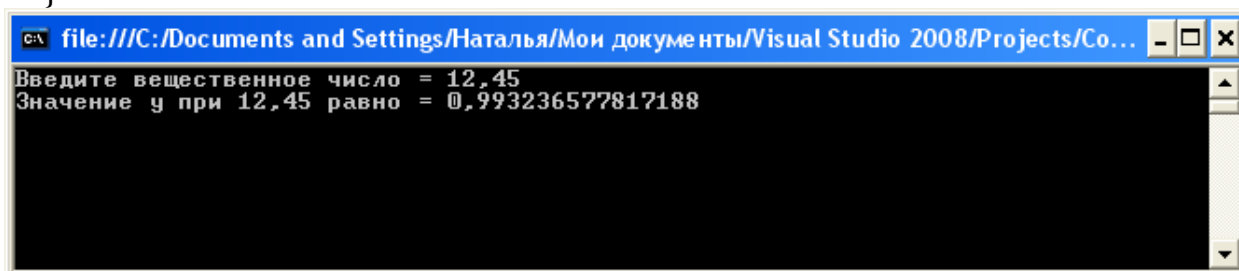


Рисунок 1.2 – Окно выполнения программы

Задания:

Вариант 1.

Даны два ненулевых числа. Найти их сумму, разность, произведение и частное.

Вариант 2.

Даны два числа. Найти среднее арифметическое их квадратов и среднее арифметическое их модулей.

Вариант 3.

Скорость лодки в стоячей воде V км/ч, скорость течения реки U км/ч ($U < V$). Время движения лодки по озеру T_1 ч, а по реке (против течения) — T_2 ч. Определить путь S , пройденный лодкой.

Вариант 4.

Скорость первого автомобиля V_1 км/ч, второго — V_2 км/ч, расстояние между ними S км. Определить расстояние между ними через T часов, если автомобили удаляются друг от друга.

Вариант 5.

Скорость первого автомобиля V_1 км/ч, второго — V_2 км/ч, расстояние между ними S км. Определить расстояние между ними через T часов, если автомобили первоначально движутся навстречу друг другу.

Вариант 6.

Найти периметр и площадь прямоугольного треугольника, если даны длины его катетов a и b .

Вариант 7.

Дана длина ребра куба. Найти площадь грани, площадь полной поверхности и объем этого куба.

Вариант 8.

Найти длину окружности и площадь круга заданного радиуса R . В качестве значения π использовать 3.14.

Вариант 9.

Найти площадь кольца, внутренний радиус которого равен R_1 , а внешний радиус равен R_2 ($R_1 < R_2$). В качестве значения π использовать 3.14.

Вариант 10.

Дана сторона равностороннего треугольника. Найти площадь этого треугольника и радиусы вписанной и описанной окружностей.

Вариант 11.

Дана длина окружности. Найти площадь круга, ограниченного этой окружностью. В качестве значения P_i использовать 3.14.

Вариант 12.

Дана площадь круга. Найти длину окружности, ограничивающей этот круг. В качестве значения P_i использовать 3.14.

Вариант 13.

Найти периметр и площадь равнобедренной трапеции с основаниями a и b ($a > b$) и углом α при большем основании (угол дан в радианах).

Вариант 14.

Найти периметр и площадь прямоугольной трапеции с основаниями a и b ($a > b$) и острым углом α (угол дан в радианах).

Вариант 15.

Найти расстояние между двумя точками с заданными координатами (x_1, y_1) и (x_2, y_2) .

Вариант 16.

Даны координаты трех вершин треугольника (x_1, y_1) , (x_2, y_2) , (x_3, y_3) . Найти его периметр и площадь.

Вариант 17.

Найти корни квадратного уравнения $A \cdot x^2 + B \cdot x + C = 0$, заданного своими коэффициентами A , B , C (коэффициент A не равен 0), если известно, что дискриминант уравнения неотрицателен.

Вариант 18.

Найти решение системы уравнений вида $A_1 \cdot x + B_1 \cdot y = C_1$, $A_2 \cdot x + B_2 \cdot y = C_2$, заданной своими коэффициентами A_1 , B_1 , C_1 , A_2 , B_2 , C_2 , если известно, что данная система имеет единственное решение.

Вариант 19.

Дано целое четырехзначное число. Используя операции `div` и `mod`, найти сумму его цифр.

Вариант 20.

Дано целое четырехзначное число. Используя операции `div` и `mod`, найти произведение его цифр.

1.3 Управляющие структуры языка C#

В языке C# имеется специальный набор операторов, управляющих ходом выполнения программы. С некоторыми из этих операторов мы уже сталкивались в предыдущей главе, а с некоторыми нам еще только предстоит познакомиться.

Чтобы программы могли проверять какие-либо условия и на основании результатов этих проверок выполнять те или иные действия, в любом языке программирования имеются **условные операторы** и **операторы выбора**.

Для выполнения каких-либо повторяющихся действий применяются специальные **итерационные (циклические) операторы**.

И наконец, операторы **безусловного перехода** позволяют программе прервать исполнение текущей последовательности команд, переключившись на исполнение другой последовательности команд.

Все эти команды позволяют управлять ходом исполнения строк программы, поэтому они и называются **управляющими**.

В таблице 1.11 мы перечислили некоторые управляющие операторы C# и кратко описали их применение.

Таблица 1.11 -. Управляющие операторы C#

Оператор	Применение
if	Выполнение строк программы в зависимости от значения логического выражения
switch	Оператор выбора. Используется для исполнения того или иного фрагмента программы в зависимости от значения переменной или выражения
goto	Оператор безусловного перехода к выполнению новой последовательности команд
for	Оператор цикла. Проверка выполнения условия завершения, а также итерация выполняются в начале цикла
while	Оператор цикла с проверкой условия завершения, выполняемой в начале цикла
foreach	Оператор цикла для просмотра всех элементов массива или коллекции
do	Оператор цикла с проверкой условия завершения, выполняемой в конце цикла
continue	Выполнение цикла с начала
break	Прерывание выполнения цикла
return	Возврат управления из метода

После изучения условных и циклических операторов, а также операторов передачи управления мы расскажем вам о **пустом** и **составном операторе** (блоке).

1.3.1 Структура выбора if/else

Любая программа может быть написана с использованием **трех управляющих структур**: структуры следования, структуры выбора и структуры повторения.

Структура следования вписана в С#. Пока не указано иное, компьютер выполняет операции одну за другой в той последовательности, в какой они записаны.

Структура выбора дает возможность создавать ветвящиеся программы с набором условий.

Структура повторения позволяет программисту определить действие, которое должно многократно повторяться при соблюдении заданных условий.

С# содержит три типа **структур выбора** и три типа **структур повторения**.

Условный оператор

Инструкция if. Полный формат ее записи такой:

```
if (условие) инструкция;  
    else инструкция;
```

Здесь под элементом *инструкция* понимается одна инструкция языка С#. Часть **else** необязательна. Вместо элемента *инструкция* может быть использован блок инструкций. В этом случае формат записи if-инструкции принимает такой вид:

```
if (условие)  
{  
    последовательность инструкций  
}  
else  
{  
    последовательность инструкций  
}
```

Если элемент *условие*, который представляет собой условное выражение, при вычислении даст значение ИСТИНА, будет выполнена if-инструкция; в противном случае — else-инструкция (если таковая существует). Обе инструкции никогда не выполняются. Условное выражение, управляющее выполнением if-инструкции, должно иметь тип bool.

Вложенные if-инструкции

Вложенные if-инструкции образуются в том случае, если в качестве элемента *инструкция* (см. полный формат записи) используется другая if-инструкция. Вложенные if-инструкции очень популярны в программировании. Главное здесь — помнить, что else-инструкция всегда относится к ближайшей if-инструкции, которая находится внутри того же программного блока, но еще не связана ни с какой другой else-инструкцией. Например:

```
if (i == 10)
{
  if (j < 20) a = b; if (k > 100) c = d;
  else a = c; // Эта else-инструкция относится к if(k > 100)
}
else a = d; // Эта else-инструкция относится к if(i == 10)
```

Как утверждается в комментариях, последняя else-инструкция не связана с инструкцией if (j < 20), поскольку они не находятся в одном блоке (несмотря на то что эта if-инструкция — ближайшая, которая не имеет при себе "else-пары"). Внутренняя else-инструкция связана с инструкцией if (k > 100), поскольку она — ближайшая и находится внутри того же блока.

Конструкция if-else-if

Очень распространенной в программировании конструкцией, в основе которой лежит вложенная if-инструкция, является "лестница" if-else-if. Ее можно представить в следующем виде:

```
if(условие)
  инструкция; else if{условие}
  инструкция; else if{условие}
  инструкция;

else
  инструкция;
```

Здесь под элементом *условие* понимается условное выражение. Условные выражения вычисляются сверху вниз. Как только в какой-нибудь ветви обнаружится истинный результат, будет выполнена инструкция, связанная с этой ветвью, а вся остальная "лестница" опускается. Если окажется, что ни одно из условий не является истинным, будет выполнена последняя else-инструкция (можно считать, что она выполняет роль условия, которое действует по умолчанию). Если последняя else-инструкция не задана, а все остальные оказались ложными, то вообще никакое действие не будет выполнено.

Преобразование строк.

Для ввода чисел необходимо преобразовывать строки в определенные типы данных. Чтобы преобразовать текстовую строку в число, нужно вызвать один из методов класса **Convert**, определенного в пространстве имен **System**.

Для преобразования текстовой строки в 32-разрядное целое число со знаком используется метод **ToInt32**:

```
int inputNumber;
```

```
inputNumber = System.Convert.ToInt32(inputString);
```

В качестве параметра мы передаем преобразуемую текстовую строку этому методу, а назад получаем искомое число.

Преобразование текстовой строки в числа и данные других типов можно выполнить при помощи методов класса **Convert**, перечисленных в таблице 1.12

Таблица 1.12 - Методы класса Convert для преобразования строк в числа

Метод	Тип числовых данных	Метод	Тип числовых данных
ToBoolean	bool	ToUInt16	ushort
ToChar	char	ToUInt32	uint
ToSbyte	sbyte	ToUInt64	ulong
ToByte	byte	ToSingle	float
ToInt16	short	ToDouble	double
ToInt32	int	ToDecimal	decimal
ToInt64	long		

Применяя эти методы, вы сможете вводить с клавиатуры числовые, логические и символьные значения. В частности, для преобразования строки в число типа **int** нужно использовать метод **ToInt32**.

Пример: Дан прямоугольный треугольник со сторонами *a* и *b*. Найти гипотенузу. Проверить треугольник равносторонний или равнобедренный.

Разработка приложения:

- 1) Запустите: Пуск -> Программы -> Microsoft Visual Studio 2008 (рисунок 1.3).

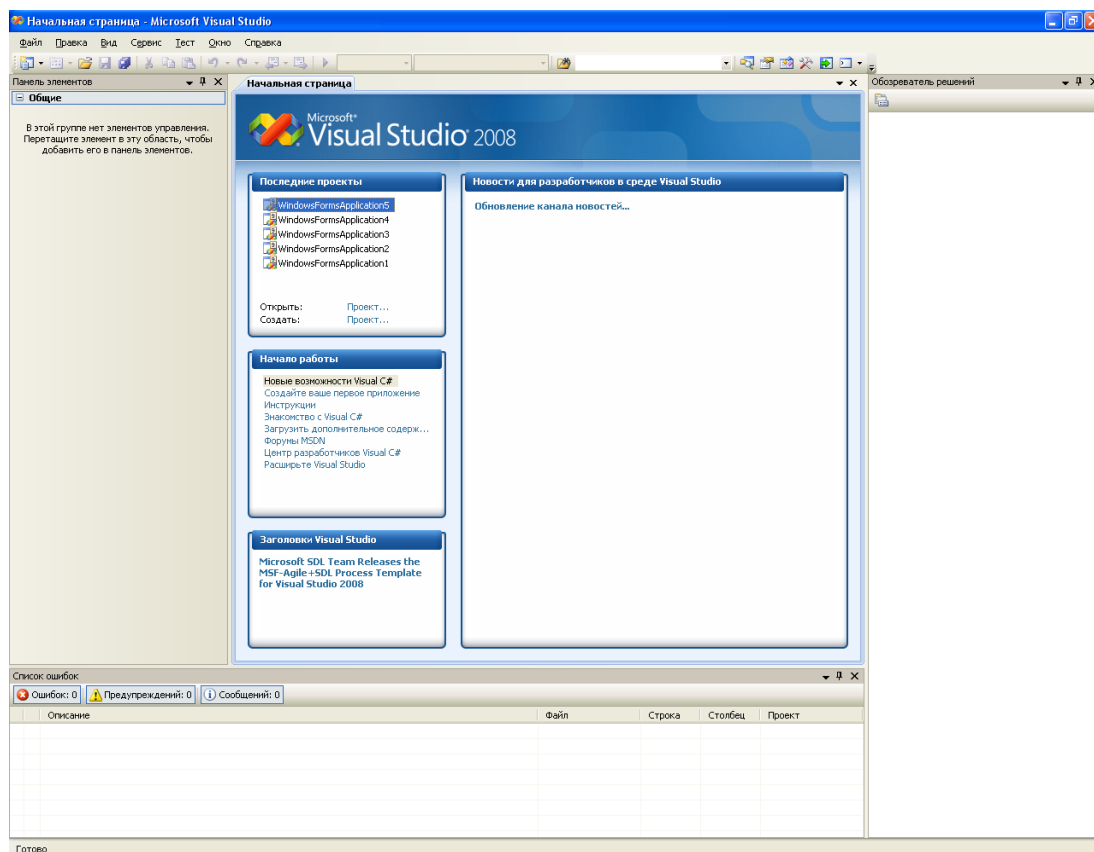


Рисунок 1.3 – Среда Microsoft Visual Studio 2008

- 2) Выполнить команду: **Файл -> Создать->Проект** (рисунок 1.4), в диалоговом окне выбрать **Visual C#**, в установленных шаблонах -> «Консольное приложение» и нажать кнопку «Ok».

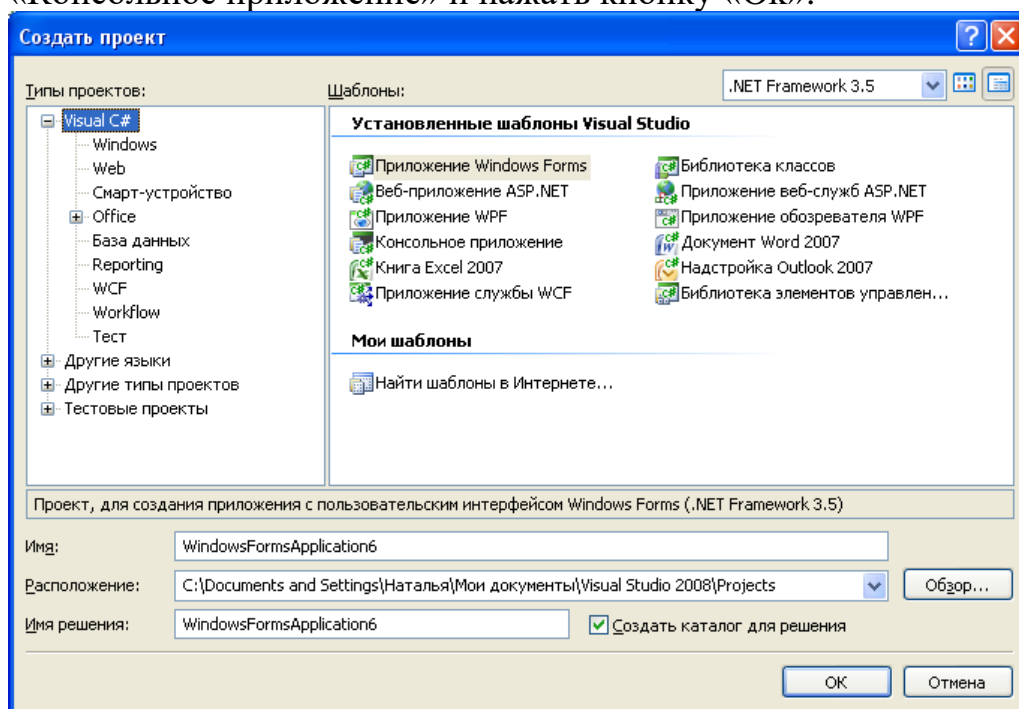


Рисунок 1.4 – Диалоговое окно

- 3) На экране появится среда для консольной разработки приложений (рисунок 1.5).

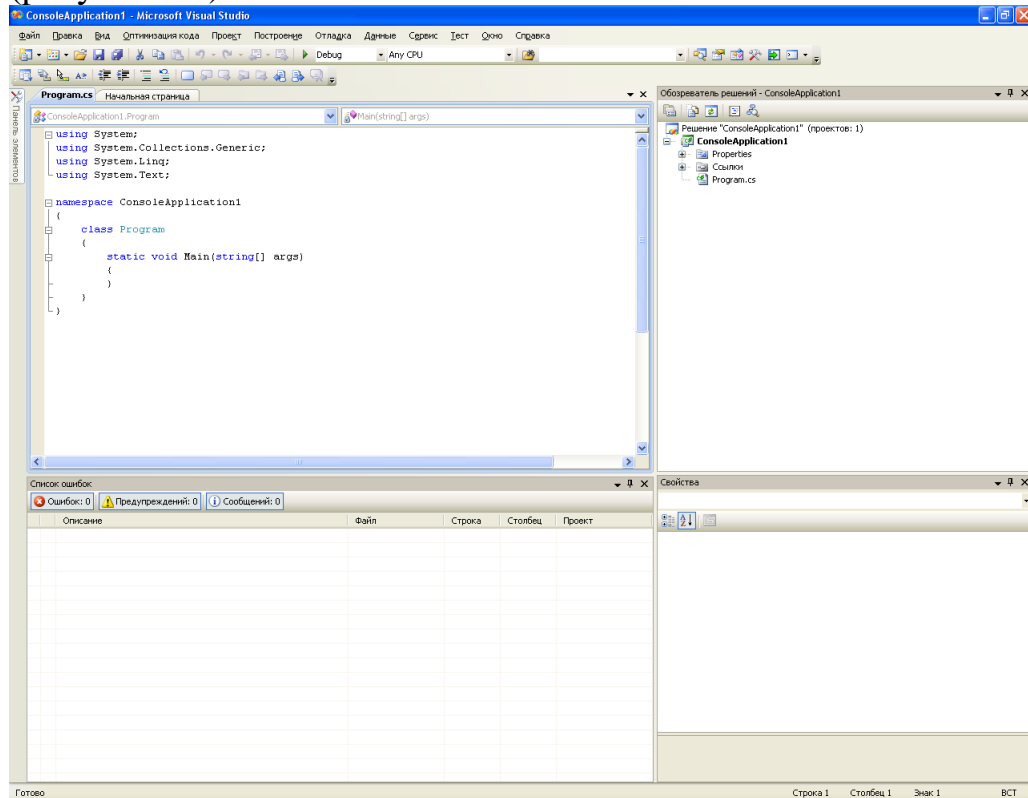


Рисунок 1.5 – Консольное приложение

- 4) В среде программирования необходимо ввести код программы в разделе: **static void Main (string[] args):**

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;
```

```
namespace ConsoleApplication1  
{  
class Program  
{
```

```
    static void Main(string[] args)  
    {  
        Console.WriteLine("Введите значение 1 стороны -");  
        string s = Console.ReadLine();  
        double a = Convert.ToDouble(s); // ввод числа a  
        Console.WriteLine("Введите значение 2 стороны -");  
        double b = Convert.ToDouble(Console.ReadLine()); // ввод числа b  
        double c;
```



```

    c = Math.Sqrt(Math.Pow(a, 2) + Math.Pow(b, 2)); // расчет стороны c
    if ((a == b) && (b == c))
        Console.WriteLine("Равносторонний треугольник");
    if ((a == b) || (b == c) || (c == a))
        Console.WriteLine("Равнобедренный треугольник");
    Console.ReadKey();
}
}

```

- 5) Для запуска приложения на выполнение необходимо выполнить команду: Отладка - > Начать отладку или горячую клавишу F5 (рисунок 1.6).

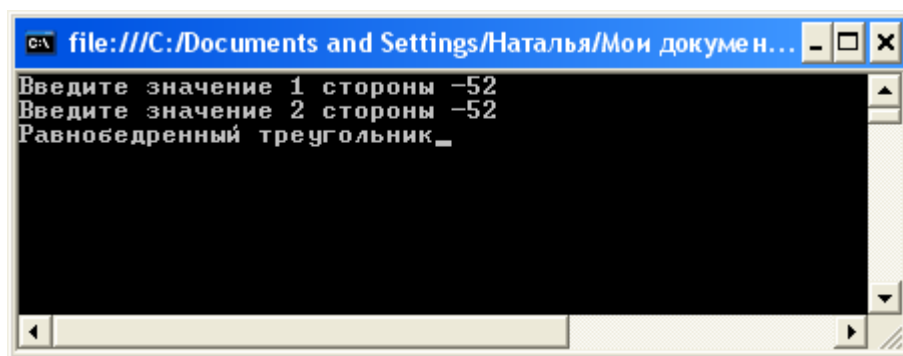


Рисунок 1.6 – Окно выполнения программы

Задания:

Вариант 1.

Даны три целых числа. Возвести в квадрат отрицательные числа и в третью степень — положительные (число 0 не изменять).

Вариант 2.

Из трех данных чисел выбрать наименьшее.

Вариант 3.

Из трех данных чисел выбрать наибольшее.

Вариант 4.

Из трех данных чисел выбрать наименьшее и наибольшее.

Вариант 5.

Перераспределить значения переменных X и Y так, чтобы в X оказалось меньшее из этих значений, а в Y — большее.

Вариант 6.

Значения переменных X , Y , Z поменять местами так, чтобы они оказались упорядоченными по возрастанию.

Вариант 7.

Значения переменных X , Y , Z поменять местами так, чтобы они оказались упорядоченными по убыванию.

Вариант 8.

Даны две переменные целого типа: A и B . Если их значения не равны, то присвоить каждой переменной сумму этих значений, а если равны, то присвоить переменным нулевые значения.

Вариант 9.

Даны две переменные целого типа: A и B . Если их значения не равны, то присвоить каждой переменной максимальное из этих значений, а если равны, то присвоить переменным нулевые значения.

Вариант 10.

Даны три переменные: X , Y , Z . Если их значения упорядочены по убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное.

Вариант 11.

Даны три переменные: X , Y , Z . Если их значения упорядочены по возрастанию или убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное.

Вариант 12.

Даны целочисленные координаты точки на плоскости. Если точка не лежит на координатных осях, то вывести 0. Если точка совпадает с началом координат, то вывести 1. Если точка не совпадает с началом координат, но лежит на оси OX или OY , то вывести соответственно 2 или 3.

Вариант 13.

Даны вещественные координаты точки, не лежащей на координатных осях OX и OY . Вывести номер координатной четверти, в которой находится данная точка.

Вариант 14.

На числовой оси расположены три точки: A , B , C . Определить, какая из двух последних точек (B или C) расположена ближе к A , и вывести эту точку и ее расстояние от точки A .

Вариант 15.

Даны четыре целых числа, одно из которых отлично от трех других, равных между собой. Вывести порядковый номер этого числа.

Вариант 16.

Дан номер некоторого года (положительное целое число). Вывести соответствующий ему номер столетия, учитывая, что, к примеру, началом 20 столетия был 1901 год.

Вариант 17.

Дан номер некоторого года (положительное целое число). Вывести число дней в этом году, учитывая, что обычный год насчитывает 365 дней, а високосный — 366 дней. Високосным считается год, делящийся на 4, за исключением тех годов, которые делятся на 100 и не делятся на 400 (например, годы 300, 1300 и 1900 не являются високосными, а 1200 и 2000 — являются).

Вариант 18.

Для данного x вычислить значение следующей функции f , вещественные значения: -1 если $x \leq 0$, $f(x) = x$ если $0 < x < 2$, 4 , если $x \geq 2$.

Вариант 19.

Для данного x вычислить значение следующей функции f , принимающей значения целого типа: 0, если $x < 0$, $f(x) = 1$, если x принадлежит $[0, 1)$, $f(x) = 2, 3, \dots, -1$ если x принадлежит $[1, 2), [2, 3), \dots$.

Вариант 20.

Дано целое число, лежащее в диапазоне от -999 до 999 . Вывести строку — словесное описание данного числа вида "отрицательное двузначное число", "нулевое число", "положительное однозначное число" и т.д.

1.3.2 Тернарная структура выбора

С# имеет тернарную **условную операцию** (`? :`), близкую по структуре, к **if/else**. Например,

```
int i=11;
```

```
Console.WriteLine (i>9 ? "многозначное число" : "однозначное  
число");
```

Первый операнд $i > 9$ является условием, второй «**многозначное число**» содержит значение условного выражения, если условие истинно, третий «**однозначное число**» — значение условного выражения, если условие ложно.

Недостатком тернарной операции по сравнению с **if/else** является то, что второй и третий операнды могут быть представлены только одним выражением, но не несколькими в фигурных скобках.

В операцию может входить один операнд (унарная операция, например, `i++`;) или два операнда (бинарная операция, например, `a=b+c`;) . Условная операция (`? :`) содержит три операнда и является единственной тернарной операцией в C#.

Пример: Проверить число положительное или отрицательное.

```
Console.WriteLine (m>0 ? “положительное число” : “отрицательное  
число”);
```

1.3.3 Оператор выбора

Переключатель **switch**

Сложную структуру вложенных условных операторов можно выполнить с помощью **структуры множественного выбора switch** (переключатель). Рассмотрим общую форму такой структуры.

```
switch (переменная) //заголовок  
{ //тело переключателя  
    case значение 1 :  
    { . . . } break;  
    case значение 2 :  
    { . . . } break;  
    ...  
    case значение N :  
    { . . . } break;  
    default: { . . . } break;  
} //конец переключателя
```

Переключатель начинается с заголовка, определяющего имя метки. Тело переключателя заключено в фигурные скобки. Текст тела переключателя разделен метками **case**. Двоеточие в данном случае — признак метки.

Действие структуры **switch** начинается с операции, стоящей за меткой, значение которой совпадает с ранее определенным, и **продолжается до ближайшего оператора break** (прерывать), который осуществляет **выход из тела переключателя**.

Фигурные скобки, которыми выделены операции, соответствующие определенной метке, не обязательны. В данном случае они использованы для наглядности, но иногда могут быть полезны для устранения возможных двусмысленностей в программе.

Если заданная перед переключателем метка не совпадает ни с одной из предусмотренных в нем, выполняется операция после метки **default** (умолчание).

Например:

```
int i = 2;
switch(i)
{
    case 1:
        System.Console.WriteLine(" case 1" ) ; break;
    case 2: {
        System.Console.WriteLine(" case 2" ); break;
    }
    case 3: {
        System.Console.WriteLine("case 3"); break;
    }
    default: {
        System.Console.WriteLine(" default"); break;
    }
}
```

Объединение меток case

В некоторых случаях требуется выполнять одну и ту же обработку для разных значений выражения, расположенного после ключевого слова **switch**. В языке C# это можно сделать следующим образом:

```
int i = 2;
switch(i)
{
    case 1: case 2:
        System.Console.WriteLine("case 1 или 2"); break;
    case 3:
        System.Console.WriteLine("case 3 ") ; break;
    default:
        System.Console.WriteLine("default") ; break;
}
```

Здесь мы объединили вместе метки **case 1** и **case 2**, определив одинаковую обработку для значений переменной **i**, равных единице или двум. В любом из этих случаев на консоли будет отображаться строка «**case 1 или 2**».

Пример: По введенному номеру дня недели вывести - выходной или рабочий день (суббота и воскресенье – выходные).

Разработка приложения:

1. Запустите среду разработки Microsoft Visual Studio 2008 в консольном режиме для C#.
2. Внесите следующий код программы в раздел **static void Main(string[] args):**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Введите номер дня недели: ");
            string s = Console.ReadLine();
            int a = Convert.ToInt16(s);
            int x=0;
            if ((a>=1)&&(a<=5)) x=1;
            if ((a>=6)&&(a<=7)) x=2;
            switch(x)
            {
                case 1: Console.WriteLine("Рабочий день"); break;
                case 2: Console.WriteLine("Выходной день"); break;
                default:
                    Console.WriteLine("Не верный ввод данных"); break;
            }
        }
    }
}
```

3. После запуска на выполнение (F5) на экране появится окно приложения:

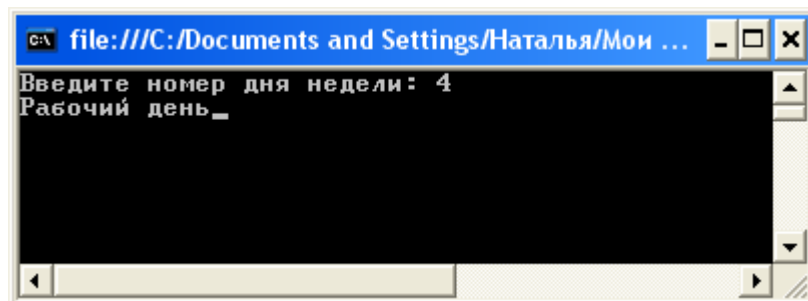


Рисунок 1.7 – Окно выполнения программы

Задания:

Вариант 1.

Дан номер месяца (1 — январь, 2 — февраль, ...). Вывести название соответствующего времени года ("зима", "весна" и т.д.).

Вариант 2.

Дан номер месяца (1 — январь, 2 — февраль, ...). Вывести число дней в этом месяце для невисокосного года.

Вариант 3.

Дано целое число в диапазоне 0 – 9. Вывести строку — название соответствующей цифры на русском языке (0 — "ноль", 1 — "один", 2 — "два", ...).

Вариант 4.

Дано целое число в диапазоне 1 – 5. Вывести строку — словесное описание соответствующей оценки (1 — "плохо", 2 — "неудовлетворительно", 3 — "удовлетворительно", 4 — "хорошо", 5 — "отлично").

Вариант 5.

Арифметические действия над числами пронумерованы следующим образом: 1 — сложение, 2 — вычитание, 3 — умножение, 4 — деление. Дан номер действия и два числа А и В (В не равно нулю). Выполнить над числами указанное действие и вывести результат.

Вариант 6.

Единицы длины пронумерованы следующим образом: 1 — дециметр, 2 — километр, 3 — метр, 4 — миллиметр, 5 — сантиметр. Дан номер единицы длины и длина отрезка L в этих единицах (вещественное число). Вывести длину данного отрезка в метрах.

Вариант 7.

Единицы массы пронумерованы следующим образом: 1 — килограмм, 2 — миллиграмм, 3 — грамм, 4 — тонна, 5 — центнер. Дан номер единицы массы и масса тела M в этих единицах (вещественное число). Вывести массу данного тела в килограммах.

Вариант 8.

Робот может перемещаться в четырех направлениях ("С" — север, "З" — запад, "Ю" — юг, "В" — восток) и принимать три цифровые команды: 0 — продолжать движение, 1 — поворот налево, –1 — поворот направо. Дан символ С — исходное направление робота и число N — посланная ему команда. Вывести направление робота после выполнения полученной команды.

Вариант 9.

Локатор ориентирован на одну из сторон света ("С" — север, "З" — запад, "Ю" — юг, "В" — восток) и может принимать три цифровые команды: 1 — поворот налево, -1 — поворот направо, 2 — поворот на 180 градусов. Дан символ С — исходная ориентация локатора и числа N1 и N2 — две посланные ему команды. Вывести ориентацию локатора после выполнения данных команд.

Вариант 10.

Элементы окружности пронумерованы следующим образом: 1 — радиус (R), 2 — диаметр (D), 3 — длина (L), 4 — площадь круга (S). Дан номер одного из этих элементов и его значение. Вывести значения остальных элементов данной окружности (в том же порядке). В качестве значения P_i использовать 3.14.

Вариант 11.

Элементы равнобедренного прямоугольного треугольника пронумерованы следующим образом: 1 — катет (a), 2 — гипотенуза (c), 3 — высота, опущенная на гипотенузу (h), 4 — площадь (S). Дан номер одного из этих элементов и его значение. Вывести значения остальных элементов данного треугольника (в том же порядке).

Вариант 12.

Элементы равностороннего треугольника пронумерованы следующим образом: 1 — сторона (a), 2 — радиус вписанной окружности (R1), 3 — радиус описанной окружности (R2), 4 — площадь (S). Дан номер одного из этих элементов и его значение. Вывести значения остальных элементов данного треугольника (в том же порядке).

Вариант 13.

Даны два целых числа: D (день) и M (месяц), определяющие правильную дату невисокосного года. Вывести значения D и M для даты, предшествующей указанной.

Вариант 14.

Даны два целых числа: D (день) и M (месяц), определяющие правильную дату невисокосного года. Вывести значения D и M для даты, следующей за указанной.

Вариант 15.

Дано целое число в диапазоне 20 – 69, определяющее возраст (в годах). Вывести строку — словесное описание указанного возраста, обеспечив

правильное согласование числа со словом "год", например: 20 — "двадцать лет", 32 — "тридцать два года", 41 — "сорок один год".

Вариант 16.

Дано целое число в диапазоне 100 – 999. Вывести строку — словесное описание данного числа, например: 256 — "двести пятьдесят шесть", 814 — "восемьсот четырнадцать".

Вариант 17.

В восточном календаре принят 60-летний цикл, состоящий из 12-летних подциклов, обозначаемых названиями цвета: зеленый, красный, желтый, белый и черный. В каждом подцикле годы носят названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, курицы, собаки и свиньи. По номеру года вывести его название, если 1984 год был началом цикла — годом зеленой крысы.

1.3.4 Операторы повторений

Итерационные операторы применяются в программах C# для выполнения каких-либо повторяющихся действий, т. е. для организации циклов. Иногда эти операторы называются **циклическими**.

В этом разделе мы расскажем об использовании итерационных операторов `for`, `while`, `do` и `foreach`. Существует 3 оператора повторений: цикл с параметром, с предусловием и постусловием.

Оператор с параметром.

Оператор с параметром **for** представляет собой удобную конструкцию, позволяющую выполнить некоторое действие заданное количество раз. Он имеет следующий синтаксис:

for(<Инициализация счетчиков>; <Условие >;<Изменение счетчиков>)
 <оператор >

Здесь **for** – зарезервированное слово; <Инициализация счетчиков> - присваивание счетчикам цикла начальных значений; <Условие > - условное выражение, определяющее продолжительность выполнения цикла; <Изменение счетчиков> - выражения, определяющие изменение счетчиков цикла после каждой итерации; <оператор > - тело цикла.

Цикл имеет одну или несколько целочисленных переменных, которые называются счетчиками цикла. Перед началом цикла им присваиваются некоторые начальные значения, затем проверяется истинность условия <Условие>: если условие истинно, выполняется тело цикла, после чего

счетчики изменяют свои значения. Если условие ложно, оператор завершает свою работу.

Таким образом, в отличие от счетных операторов других языков программирования, с циклом можно связать несколько счетчиков. В большинстве случаев имеется единственный счетчик, но возможность использования нескольких счетчиков может оказаться весьма полезной особенностью цикла `for`.

Например,

```
int i; double a=5;
for (i=1;i<5;i++)
    { Console.WriteLine(a);a+=2.0;}
```

Выполнение происходит следующим образом: **управляющей переменной *i*** присваивается начальное значение (**1**), производится проверка условия (***i*<5**) и, если оно истинно, выполняется тело цикла (на экран выводится значение **a**, равное 5, переменная **a** увеличивается на 2.0); значение управляющей переменной **i** изменяется (увеличивается на 1), производится проверка условия (***i*<5**) и, если оно как в нашем случае истинно (***i*=2**), выполняется тело цикла (на экран выводится число 7); значение **i** снова изменяется (увеличивается на 1) и т.д. Процесс продолжается до тех пор, пока выполняется условие (***i*<5**). Когда условие становится ложным (при ***i*=5**) процесс заканчивается. Последнее выведенное число в нашем случае равно **11** (для ***i*=4**).

Если заданное условие всегда ложно, например,

```
(i=1; i<1; i++)
```

тело цикла не исполняется ни разу. Если заданное условие всегда истинно, например,

```
(i=1 ;i>0; 1++)
```

тело цикла будет повторяться бесконечно (зацикливание).

Если тело цикла содержит одну операцию, фигурные скобки можно снять. В числе операций тела цикла могут быть другие структуры повторения, то есть **циклы могут быть вложенными один в другой**. Допустимое количество вложенных циклов зависит от версии компилятора.

Прерывание цикла

С помощью оператора `break` можно в любой момент прервать выполнение цикла. Например, в следующем фрагменте программы мы прерываем работу цикла, когда значение переменной **i** становится больше пяти:

```
for(i = 0; i < 10; i++)
{
    if (i > 5) break;
    System.Console .Write (" {0}  ", i );
}
```

```
}
```

В результате на консоль будут выведены цифры от 0 до 5:

0 1 2 3 4 5

Возобновление цикла

В отличие от оператора **break**, прерывающего цикл, оператор **continue** позволяет возобновить выполнение цикла с самого начала.

Вот как он используется:

```
for(i = 0;; i++) (  
    System.Console.Write("{0} ",i ); if(i < 9)  
        continue; else  
        break;  
)
```

Если в ходе выполнения цикла значение переменной *i* не достигло девяти, цикл возобновляет свою работу с самого начала (т. е. с вывода значения переменной цикла на консоль). Когда указанное значение будет достигнуто, выполнение цикла прервется оператором **break**.

Пример 1: Найти сумму N целых чисел.

Разработка приложения:

1. Запустите среду разработки Microsoft Visual Studio 2008 в консольном режиме для C#.
2. Внесите следующий код программы в раздел **static void Main(string[] args):**

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;
```

```
namespace ConsoleApplication1
```

```
{
```

```
class Program
```

```
{
```

```
    static void Main(string[] args)
```

```
    {
```

```
        Console.Write("Введите последнюю цифру: ");
```

```
        int N = Convert.ToInt16(Console.ReadLine());
```

```
        double Sum = 0.0;
```

```
        for (int i = 1; i <= N; ++i)    // цикл выполнится пока i меньше N
```

```

        {
            Sum += i;
        }
        Console.WriteLine("Сумма " + N + " чисел = " + Sum);
        Console.ReadKey();
    }
}

```

3. На экране после запуска на выполнение (F5) появится следующая информация (рисунок 1.8):

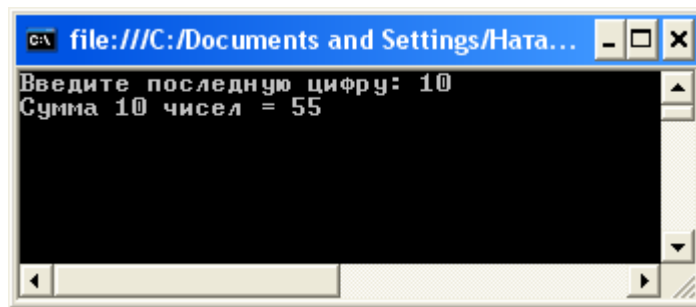


Рисунок 1.8 – Окно выполнения программы

Операторы с предусловием и постусловием.

Оператор `while` используется, когда количество итераций заранее неизвестно. Существуют две формы оператора: префиксная и постфиксная. В префиксной форме его синтаксис таков:

while (<Условие>) <Тело_цикла>

Здесь <while> — зарезервированное слово; <Условие> — выражение типа `bool`; <Тело_цикла> — произвольный оператор или блок операторов.

Постфиксная форма:

do
 <Тело_цикла>
while (<Условие>)

Таким образом, в префиксной форме сначала проверяется истинность условия и только после этого выполняется тело цикла, а в постфиксной сначала выполняется тело цикла и лишь потом проверяется условие: если оно истинно, тело цикла выполняется заново. В префиксной форме тело цикла может не выполниться ни разу, в то время как в постфиксной оно будет выполнено хотя бы один раз.

Очевидно, что в операторе while проверяемое условие должно изменяться в теле цикла, в противном случае цикл никогда не завершится естественным образом, и программа «зациклится».

Пример 1: Найти сумму N целых четных чисел (оператор цикла while).

Разработка приложения:

1. Запустите среду разработки Microsoft Visual Studio 2008 в консольном режиме для C#.
2. Внесите следующий код программы в раздел **static void Main(string[] args):**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Введите последнюю цифру: ");
            string s = Console.ReadLine();
            int N = Convert.ToInt16(s);
            double Sum = 0.0; int i = 2;

            while (i <= N)
            {
                Sum += i;
                i += 2;
            }

            Console.WriteLine("Сумма " + N + " четных чисел = " + Sum);
            Console.ReadKey();
        }
    }
}
```

3. На экране после запуска на выполнение (F5) появится следующая информация:

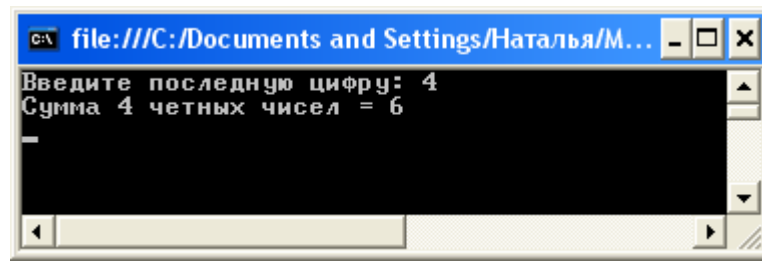


Рисунок 1.9 – Окно выполнения программы

Пример 2: Ввести N чисел и найти количество положительных и сумму отрицательных (оператор цикла do ..while).

Разработка приложения:

1. Запустите среду разработки Microsoft Visual Studio 2008 в консольном режиме для C#.
2. Внесите следующий код программы в раздел **static void Main(string[] args):**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Введите количество цифр: ");
            string s = Console.ReadLine();
            int N = Convert.ToInt16(s);
            double Sum = 0.0;
            int i = 1, k=0, a;
            do
            {
                Console.Write("Введите" + i + "число - ");
                a = Convert.ToInt16(Console.ReadLine());
                if (a > 0)
                    k++;
                if (a < 0)
                    Sum += a;
                ++i;
            }
            while (i <= N);
```

```

    Console.WriteLine("Сумма отрицательных чисел = " + Sum);
    Console.WriteLine("Количество положительных чисел = " + k);
    Console.ReadKey();
}
}

```

3. На экране после запуска на выполнение (F5) появится следующая информация:

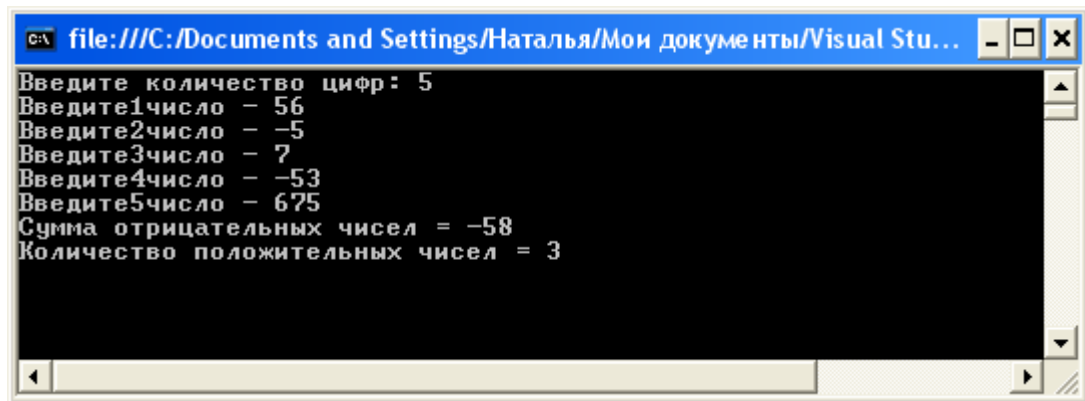


Рисунок 1.10 – Окно выполнения программы

Цикл **foreach**.

Цикл **foreach** предназначен для обработки массивов, коллекций и других контейнеров, рассчитанных на хранение множества данных. Контейнер должен исполнять интерфейсы `System.Collections.IEnumerable` и `System.Collections.IEnumerator`.

Оператор имеет следующий синтаксис:

foreach (<Элемент> in <Контейнер>) <Тело_цикла>;

Здесь **foreach** и **in** — зарезервированные слова; <Элемент> — очередной элемент данных; <Контейнер> — хранилище данных; <Тело_цикла> — произвольный оператор или блок операторов. Разумеется, тип элемента должен совпадать с типом данных в контейнере. Оператор осуществляет последовательный перебор данных в контейнере и на каждой итерации возвращает очередной элемент в переменной <Элемент>. Замечу, что эта переменная в теле цикла доступна только для чтения, то есть с помощью цикла нельзя наполнить хранилище данных.

Использование инструкции **break** для выхода из цикла.

С помощью инструкции **break** можно организовать немедленный выход из цикла, опустив выполнение кода, оставшегося в его теле, и проверку условного выражения. При обнаружении внутри цикла инструкции **break** цикл завершается, а управление передается инструкции, следующей после цикла.

Использование инструкции **continue**.

Помимо средства "досрочного" выхода из цикла, существует средство "досрочного" выхода из текущей его итерации. Этим средством является инструкция **continue**. Она принудительно выполняет переход к следующей итерации, опуская выполнение оставшегося кода в текущей. Инструкцию **continue** можно расценивать как дополнение к более "радикальной" инструкции **break**.

Задания:

Вариант 1.

Даны два целых числа A и B ($A < B$). Вывести все целые числа, расположенные между данными числами (включая сами эти числа), в порядке их возрастания, а также количество N этих чисел.

Вариант 2.

Даны два целых числа A и B ($A < B$). Вывести все целые числа, расположенные между данными числами (не включая сами эти числа), в порядке их убывания, а также количество N этих чисел.

Вариант 3.

Дано вещественное число A и целое число N (> 0). Вывести A в степени N : $A^N = A \cdot A \cdot \dots \cdot A$ (числа A перемножаются N раз).

Вариант 4.

Дано вещественное число A и целое число N (> 0). Вывести все целые степени числа A от 1 до N .

Вариант 5.

Дано вещественное число A и целое число N (> 0). Вывести $1 + A + A^2 + A^3 + \dots + A^N$.

Вариант 6.

Дано вещественное число A и целое число N (> 0). Вывести $1 - A + A^2 - A^3 + \dots + (-1)^N A^N$.

Вариант 7.

Дано целое число N (> 1). Вывести наименьшее целое K , при котором выполняется неравенство $3^K > N$, и само значение 3^K .

Вариант 8.

Дано целое число N (> 1). Вывести наибольшее целое K , при котором выполняется неравенство $3^K < N$, и само значение 3^K .

Вариант 9.

Дано вещественное число $A (> 1)$. Вывести наименьшее из целых чисел N , для которых сумма $1 + 1/2 + \dots + 1/N$ будет больше A , и саму эту сумму.

Вариант 10.

Дано вещественное число $A (> 1)$. Вывести наибольшее из целых чисел N , для которых сумма $1 + 1/2 + \dots + 1/N$ будет меньше A , и саму эту сумму.

Вариант 11.

Дано целое число $N (> 0)$. Вывести произведение $1 \cdot 2 \cdot \dots \cdot N$. Чтобы избежать целочисленного переполнения, вычислять это произведение с помощью вещественной переменной и выводить его как вещественное число.

Вариант 12.

Дано целое число $N (> 0)$. Если N — нечетное, то вывести произведение $1 \cdot 3 \cdot \dots \cdot N$; если N — четное, то вывести произведение $2 \cdot 4 \cdot \dots \cdot N$. Чтобы избежать целочисленного переполнения, вычислять это произведение с помощью вещественной переменной и выводить его как вещественное число.

Вариант 13.

Дано целое число $N (> 0)$. Вывести сумму $2 + 1/(2!) + 1/(3!) + \dots + 1/(N!)$ (выражение $N!$ — " N факториал" — обозначает произведение всех целых чисел от 1 до N : $N! = 1 \cdot 2 \cdot \dots \cdot N$). Полученное число является приближенным значением константы $e = \exp(1) (= 2.71828183\dots)$.

Вариант 14.

Дано вещественное число X и целое число $N (> 0)$. Вывести $1 + X + X^2/2! + \dots + X^N/N!$ ($N! = 1 \cdot 2 \cdot \dots \cdot N$). Полученное число является приближенным значением функции \exp в точке X .

Вариант 15.

Дано вещественное число X и целое число $N (> 0)$. Вывести $X - X^3/3! + X^5/5! - \dots + (-1)^N X^{2N+1}/(2N+1)!$ ($N! = 1 \cdot 2 \cdot \dots \cdot N$). Полученное число является приближенным значением функции \sin в точке X .

Вариант 16.

Дано вещественное число X и целое число $N (> 0)$. Вывести $1 - X^2/2! + X^4/4! - \dots + (-1)^N X^{2N}/(2N)!$ ($N! = 1 \cdot 2 \cdot \dots \cdot N$). Полученное число является приближенным значением функции \cos в точке X .

Вариант 17.

Дано вещественное число X ($|X| < 1$) и целое число N (> 0). Вывести $X - X^2/2 + X^3/3 - \dots + (-1)^{N-1}X^N/N$. Полученное число является приближенным значением функции \ln в точке $1+X$.

Вариант 18.

Дано вещественное число X ($|X| < 1$) и целое число N (> 0). Вывести $X - X^3/3 + X^5/5 - \dots + (-1)^N X^{2N+1}/(2N+1)$. Полученное число является приближенным значением функции \arctg в точке X .

Вариант 19.

Дано целое число N (> 2) и две вещественные точки на числовой оси: A , B ($A < B$). Отрезок $[A, B]$ разбит на равные отрезки длины H с концами в N точках вида $A, A + H, A + 2H, A + 3H, \dots, B$. Вывести значение H и набор из N точек, образующий разбиение отрезка $[A, B]$.

Вариант 20.

Дано целое число N (> 2) и две вещественные точки на числовой оси: A , B ($A < B$). Функция $F(X)$ задана формулой $F(X) = 1 - \sin(X)$. Вывести значения функции F в N равноотстоящих точках, образующих разбиение отрезка $[A, B]$: $F(A), F(A + H), F(A + 2H), \dots, F(B)$.

1.4 Массивы

1.4.1 Одномерные массивы

Массивы представляют собой упорядоченные структуры, содержащие множество данных одного и того же типа. Упорядоченность массива позволяет обращаться к отдельному его элементу с помощью **индекса** — целочисленного выражения, определяющего положение элемента в массиве. Как и в других C-подобных языках, в C# самый первый элемент массива имеет индекс 0.

Массивы относятся к ссылочным типам, поэтому должны инициализироваться при помощи оператора **new**.

Одномерные массивы используются в программах чаще всего.

Варианты описания массива:

тип[] имя;

тип[] имя = new тип [размерность];

тип[] имя = { список инициализаторов };

тип[] имя = new тип [] { список инициализаторов };

тип[] имя = new тип [размерность] { список инициализаторов };

Для описания массива нужно указать квадратные скобки за именем типа данных. При этом он определяет тип хранящихся в массиве данных:

```

int[] arrInt;           // Целочисленный массив
char[] arrChar;       // Массив символов
float[] arrFloat;     // Массив вещественных чисел

```

Объявление массива еще не создает объект, который можно использовать в программе. Для инициализации массива следует указать количество его элементов:

```

arrInt = new int[25];      // Массив содержит 25 целых чисел
errChar = new char[3];    // Массив содержит 3 символа
errFloat = new float[10]; // Массив из 10 вещественных чисел

```

При инициализации массива ему выделяется нужная память в куче, а элементы получают значение 0, которое трактуется в зависимости от их типа: для числовых массивов это число 0, для строковых — пустая строка, для символьных — отсутствие символа и т.д.

После инициализации массив готов к работе. Разумеется, можно объявлять массив и одновременно инициализировать его:

```

int[] arrInt = new int[9]

```

Более того, можно одновременно с созданием массива наполнить его нужными значениями. Например, так:

```

int[] arrInt = {1, 2, 3, 23}

```

Или так:

```

int[] arrInt = new int[4] {1, 2, 3, 23}

```

Пример: Ввести одномерный массив из 6 целых чисел, найти сумму и количество отрицательных, а так же максимальный элемент.

Разработка приложения:

1. Запустите среду разработки Microsoft Visual Studio 2008 в консольном режиме для C#.
2. Внесите следующий код программы в раздел **static void Main(string[] args):**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

```

```

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {

```

```

const int n = 6;
int[] a = new int[n];
Console.WriteLine("Исходный массив:");
for (int i = 0; i < n; ++i)
{
    a[i] = Convert.ToInt16(Console.ReadLine());
}
long sum = 0; // сумма отрицательных элементов
int num = 0; // количество отрицательных элементов
for ( int i =0; i < n; ++i )
    if ( a[i] < 0 )
        sum += a[i]; ++num;
    Console.WriteLine( "Сумма отрицательных = " + sum );
    Console.WriteLine( "Кол-во отрицательных = " + num );
int max = a[0]; // максимальный элемент
for ( int i = 1; i < n; ++i )
    if ( a[i] > max ) max = a[i];
    Console.WriteLine( "Максимальный элемент = " + max );
Console.ReadKey();
}
}

```

3. На экране после запуска на выполнение (F5) появится следующая информация (рисунок 1.11):

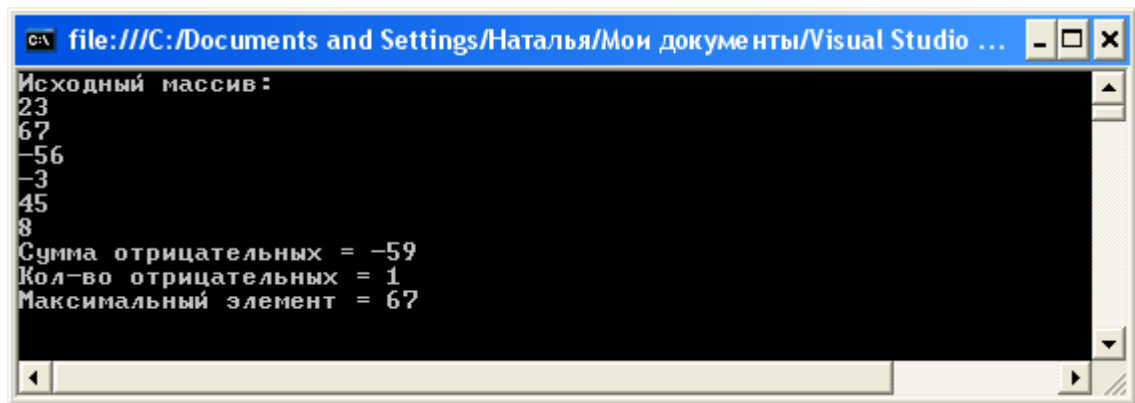


Рисунок 1.11 – Окно выполнения программы

Вариант 1.

Дан массив размера N. Вывести его элементы в обратном порядке.

Вариант 2.

Дан массив размера N. Вывести вначале его элементы с четными (нечетными) индексами, а затем — с нечетными (четными).

Вариант 3.

Дан целочисленный массив A размера 10. Вывести номер первого из тех его элементов $A[i]$, которые удовлетворяют двойному неравенству: $A[1] < A[i] < A[10]$. Если таких элементов нет, то вывести 0.

Вариант 4.

Дан целочисленный массив размера N . Вывести вначале все его четные элементы, а затем — нечетные.

Вариант 5.

Поменять местами минимальный и максимальный элементы массива размера 10.

Вариант 6.

Заменить все положительные элементы целочисленного массива размера 10 на значение минимального.

Вариант 7.

Дан массив размера 10. Переставить в обратном порядке элементы массива, расположенные между его минимальным и максимальным элементами.

Вариант 8.

Дан массив размера N . Осуществить циклический сдвиг элементов массива влево на одну позицию.

Вариант 9.

Дан массив размера N и число k ($0 < k < 5$, $k < N$). Осуществить циклический сдвиг элементов массива влево¹|вправо² на k позиций.

Вариант 10.

Проверить, образуют ли элементы целочисленного массива размера N арифметическую прогрессию. Если да, то вывести разность прогрессии, если нет — вывести 0.

Вариант 11.

Дан массив ненулевых целых чисел размера N . Проверить, чередуются ли в нем [четные и нечетные] числа. Если чередуются, то вывести 0, если нет, то вывести номер первого элемента, нарушающего закономерность.

Вариант 12.

Дан массив размера N . Найти количество его локальных минимумов¹|максимумов².

Вариант 13.

Дан массив размера N. Найти максимальный из его локальных минимумов.

Вариант 14.

Дан массив размера N. Определить количество участков, на которых его элементы монотонно возрастают.

Вариант 15.

Дан массив размера N. Определить количество его промежутков монотонности (то есть участков, на которых его элементы возрастают или убывают).

Вариант 16.

Дано вещественное число R и массив размера N. Найти элемент массива, который наиболее близок к данному числу.

Вариант 17.

Дано вещественное число R и массив размера N. Найти два элемента массива, сумма которых наиболее близка к данному числу.

Вариант 18.

Дан массив размера N. Найти номера двух ближайших чисел из этого массива.

Вариант 19.

Дан целочисленный массив размера N. Определить максимальное количество его одинаковых элементов.

Вариант 20.

Дан целочисленный массив размера N. Удалить из массива все элементы, встречающиеся [менее двух раз]1[[более двух раз]2[[ровно два раза]3[[ровно три раза]4.

1.4.2 Многомерные массивы

Поскольку тип элементов массива может быть любым, им, в частности, может быть и сам массив. Массивы массивов называются **многомерными**. При объявлении многомерного массива в квадратных скобках указываются запятые.

```
тип[,] имя = new тип [ разм_1. разм_2 ]:  
тип[,] иия = { список инициализаторов };  
тип[,] имя = new тип [,] { список инициализаторов };
```

тип[,] имя = new тип [разм_1, разм_2] { список инициализаторов };

Примеры описаний (один пример для каждого варианта описания):

```
int[,] a; //элементов нет
int[,] b = new int[2, 3]; //элементы равны 0
int[,] c = {{1, 2, 3}, {4,5,6}}; // new подразумевается
int[,] c = new int[,] {{1, 2, 3}, {4, 5, 6}}; // размерность вычисляется
int[,] d = new int[2,3] {{1, 2,3}, {4, 5,6}}; // избыточное описание
```

Если список инициализации не задан, размерности могут быть не только константами, но и выражениями типа, приводимого к целому. К элементу двумерного массива обращаются, указывая номера строки и столбца, на пересечении которых он расположен, например:

a[1, 4] b[I, j] b[j, i]

Каждая запятая в объявлении многомерного массива соответствует очередной его **размерности**, то есть добавлению к массиву нового элемента-массива.

Класс System.Array

Все массивы в CLR и, следовательно, в C# являются наследниками базового класса System.Array. Этот класс имеет статические свойства и методы, упрощающие манипулирование массивом как объектом.

Некоторые свойства класса System. Array показаны в табл.1.13.

Таблица 1.13 - Свойства System.Array

Свойство	Описание
public int Length{get:};	Возвращает количество всех элементов массива
public int Rank{get:};	Возвращает количество измерений массива

Некоторые методы класса System.Array описаны в табл. 1.14.

Таблица 1.14 - Методы класса System.Array

Метод	Описание
static int BinarySearch (Array, object, IComparer) ;	Ищет в одномерном отсортированном массиве Array элемент object с помощью интерфейса IComparer и возвращает индекс элемента или отрицательное число, если элемент не найден
public static void Clear (Array, Index, Length);	Помещает в Length элементов одномерного массива Array, начиная с элемента Index, значения 0, false или null в зависимости от типа элементов

<code>public static void CopyTo (Array, Index);</code>	Копирует из текущего одномерного массива все элементы в массив <code>Array</code> , начиная с индекса <code>Index</code>
<code>public static Array CreateInstance (Type ElementsType, int[] Lengths, int [] LowerBounds);</code>	Создает многомерный массив из элементов типа <code>ElementsType</code> с количеством <code>Lengths</code> элементов по каждому измерению и нижними границами индексов <code>LowerBounds</code> . Перегруженные методы позволяют создавать одно- и многомерные массивы с индексами, начинающимися с 0
<code>public virtual IEnumerator GetEnumerator();</code>	Возвращает итератор интерфейса <code>IEnumerator</code> для текущего массива
<code>public int GetLength (Dimension);</code>	Возвращает количество элементов массива по измерению <code>Dimension</code>
<code>public int GetLowerBound (Dimension);</code>	Возвращает минимальное значение индекса по измерению <code>Dimension</code>
<code>public int GetUpperBound (Dimension);</code>	Возвращает максимальное значение индекса по измерению <code>Dimension</code>
<code>X public object GetValue (Index) ;</code>	Возвращает значение элемента <code>Index</code> текущего одномерного массива. Перекрытые методы приспособлены для работы с многомерными массивами
<code>public static void Reverse (Array);</code>	Изменяет порядок следования элементов одномерного массива <code>Array</code> на обратный
<code>public void SetValue(Value, Index);</code>	Устанавливает значение <code>Value</code> элемента <code>Index</code> текущего одномерного массива. Перекрытые методы приспособлены для работы с многомерными массивами
<code>public static void Sort (Array);</code>	Сортирует элементы одномерного массива <code>Array</code>

Пример: Ввести массив `A(3,4)`. Найти количество положительных в каждой строке и среднее арифметическое.

Разработка приложения:

1. Запустите среду разработки Microsoft Visual Studio 2008 в консольном режиме для C#.
2. Внесите следующий код программы в раздел **`static void Main(string[] args):`**

```
using System;
using System.Collections.Generic;
```



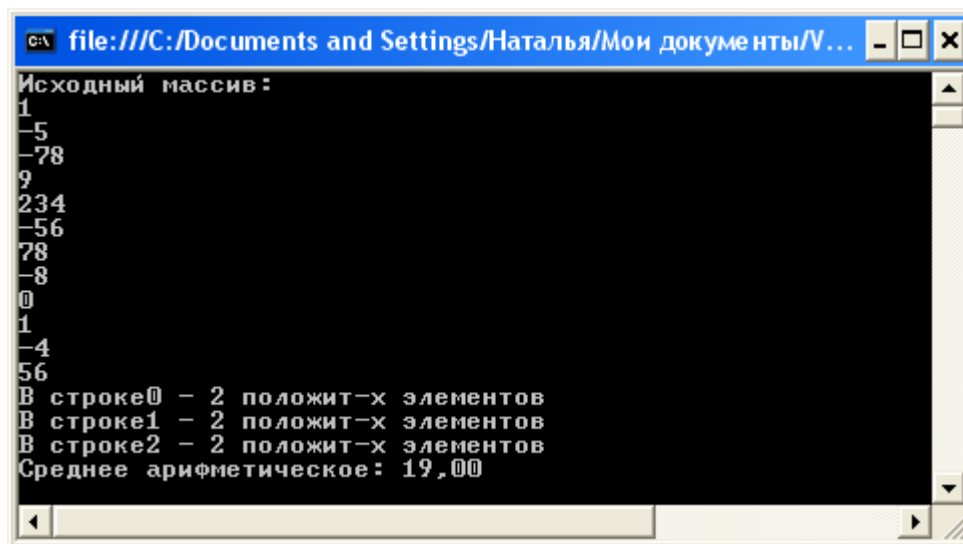
```

using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            const int m = 3, n = 4;
            int[,] a = new int[m, n];
            Console.WriteLine( "Исходный массив:" );
            for ( int i = 0; i < m; ++i )
            {
                for ( int j = 0; j < n; ++j)
                    a[i,j] = Convert.ToInt16(Console.ReadLine());
            }
            double sum = 0; int P;
            for ( int i = 0; i < m; ++i )
            {
                P = 0;
                for (int j = 0; j < n; ++j)
                {
                    sum += a[i, j];
                    if (a[i, j] > 0) ++P;
                }
                Console.WriteLine("В строке{0} - {1} положит-х элементов", i, P);
            }
            Console.WriteLine("Среднее арифметическое: " + sum /m /n );
            Console.ReadKey();
        }
    }
}

```

3. На экране после запуска на выполнение (F5) появится следующая информация (рисунок 1.12):



```
file:///C:/Documents and Settings/Наталья/Мои документы/V...
Исходный массив :
1
-5
-78
9
234
-56
78
-8
0
1
-4
56
В строке0 - 2 положит-х элементов
В строке1 - 2 положит-х элементов
В строке2 - 2 положит-х элементов
Среднее арифметическое: 19,00
```

Рисунок 1.12 – Окно выполнения программы

Вариант 1.

Дана матрица размера 5 x 9. Найти суммы элементов всех ее четных1|нечетных2 строк3|столбцов4.

Вариант 2.

Дана матрица размера 5 x 10. Найти минимальное1|максимальное2 значение в каждой строке3|столбце4.

Вариант 3.

Дана матрица размера 5 x 10. В каждой строке1|столбце2 найти количество элементов, больших3|меньших4 среднего арифметического всех элементов этой строки1|столбца2.

Вариант 4.

Дана матрица размера 5 x 10. Преобразовать матрицу, поменяв местами минимальный и максимальный элемент в каждой строке1|столбце2.

Вариант 5.

Дана матрица размера 5 x 10. Найти минимальное1|максимальное2 значение среди сумм элементов всех ее строк3|столбцов4 и номер строки3|столбца4 с этим минимальным1|максимальным2 значением.

Вариант 6.

Дана матрица размера 5 x 10. Найти минимальный1|максимальный2 среди максимальных1|минимальных2 элементов каждой строки3|столбца4.

Вариант 7.

Дана целочисленная матрица размера 5 x 10. Вывести номер ее оerbn1|последней2 строки3|столбца4, содержащего равное количество

положительных и отрицательных элементов (нулевые элементы не учитываются). Если таких строк₃|столбцов₄ нет, то вывести 0.

Вариант 8.

Дана матрица размера 5 x 10. Вывести номер ее первой₁|последней₂ строки₃|столбца₄, содержащего только положительные элементы. Если таких строк₃|столбцов₄ нет, то вывести 0.

Вариант 9.

Дана целочисленная матрица размера M x N. Различные строки (столбцы) матрицы назовем похожими, если совпадают множества чисел, встречающихся в этих строках (столбцах). Найти количество строк₁|столбцов₂, похожих на первую₃|последнюю₄ строку₁|столбец₂.

Вариант 10.

Дана целочисленная матрица размера M x N. Найти количество ее строк₁|столбцов₂, все элементы которых различны.

Вариант 11.

Дана целочисленная матрица размера M x N. Вывести номер ее первой₁|последней₂ строки₃|столбца₄, содержащего максимальное количество одинаковых элементов.

Вариант 12.

Дана квадратная матрица порядка M. Найти сумму элементов ее главной₁|побочной₂ диагонали.

Вариант 13.

Дана квадратная матрица порядка M. Найти суммы элементов ее диагоналей, параллельных главной₁|побочной₂ (начиная с одноэлементной диагонали A[1,M]₁|A[1,1]₂).

Вариант 14.

Дана квадратная матрица порядка M. Вывести минимальные₁|максимальные₂ из элементов каждой ее диагонали, параллельной главной₃|побочной₄ (начиная с одноэлементной диагонали A[1,M]₃|A[1,1]₄).

Вариант 15.

Дана квадратная матрица порядка M. Заменить нулями элементы, лежащие одновременно выше₁|ниже₂ главной диагонали (включая эту диагональ) и выше₃|ниже₄ побочной диагонали (также включая эту диагональ).

Вариант 16.

Дана квадратная матрица порядка M . Зеркально отразить ее элементы относительно [горизонтальной оси симметрии]¹[вертикальной оси симметрии]²[главной диагонали]³[побочной диагонали]⁴ матрицы.

Вариант 17.

Дана квадратная матрица порядка M . Повернуть ее на 90°|180°|270° градусов в положительном направлении.

Вариант 18.

Дана матрица размера 5×10 . Вывести количество строк¹|столбцов², элементы которых монотонно возрастают³|убывают⁴.

Вариант 19.

Дана матрица размера 5×10 . Найти минимальный¹|максимальный² среди элементов тех строк³|столбцов⁴, которые упорядочены либо по возрастанию, либо по убыванию. Если такие строки³|столбцы⁴ отсутствуют, то вывести 0.

Вариант 20.

Даны два числа k_1 и k_2 и матрица размера 4×10 . Поменять местами строки¹|столбцы² матрицы с номерами k_1 и k_2 .

1.5 Функции

Первым способом структурирования программ в языках программирования высокого уровня было использование процедур и функций — относительно самостоятельных фрагментов программ, оформленных особым образом и снабженных именем. Упоминание этого имени в программе называется *вызовом* процедуры (функции). Отличие функции от процедуры заключается в том, что результатом выполнения операторов, образующих тело функции, всегда является значение некоторого типа, поэтому функция может участвовать в образовании выражений наряду с переменными и константами. Условимся в дальнейшем называть процедуры и функции общим словом подпрограммы.

Структурирование программ необходимо по двум причинам. Во-первых, невозможно написать более-менее сложную программу как единое целое, не расчленив ее на относительно самостоятельные части — подпрограммы. Во-вторых, подпрограммы открывают возможность повторного использования однажды созданного кода и способствуют появлению библиотек подпрограмм. В языке C# подпрограммы не имеют самостоятельного значения — они могут быть только методами класса.

Описание подпрограмм

В C# нет специальных зарезервированных слов `procedure` и `function` для описания подпрограмм. Поскольку они являются методами класса, эти слова избыточны. Синтаксис описания таков:

**[модификаторы] <Тип> <Имя> ([<Формальные_параметры>])
{<Тело>}**

В квадратных скобках указаны необязательные элементы. Модификаторы определяют область видимости подпрограммы. Сейчас лишь поясню два модификатора — **private** и **public**. Любые члены класса (в том числе методы-подпрограммы), объявленные с модификатором `private`, доступны только в методах данного класса. Модификатор `public` - делает метод (подпрограмму) доступным в любом месте программы. Если модификатор не указан, считается, что данный член класса помечен как закрытый (с модификатором `private`).

Формальные параметры могут отсутствовать, но и в этом случае круглые скобки за именем подпрограммы обязательны.

Тип подпрограммы может быть любым типом данных. В этом случае подпрограмма представляет собой функцию, которая возвращает результат указанного типа. В теле функции обязательно указывается оператор `return`, который присваивает функции нужное значение. В качестве типа можно указать зарезервированное слово `void`, которое означает отсутствие типа. В этом случае подпрограмма представляет собой процедуру, и использование в ней оператора `return` означает принудительное завершение ее работы.

Имя подпрограммы должно быть уникальным в текущей области видимости идентификатором.

Тело подпрограммы обязательно реализуется в виде блока операторов, поэтому за закрывающей круглой скобкой должна следовать открывающая фигурная, даже если тело подпрограммы отсутствует Или содержит единственный оператор.

Примеры описаний:

`int A() {...}`

void `В (...)` {...}

public `string C ()` {...}

Здесь `A` и `В` — закрытые члены класса, которые доступны только в методах этого же класса, причем `A` — целочисленная функция, а `В` — процедура. `C` — открытая функция строкового типа, доступная в любой точке программы.

В таблице 1.15 представлены примеры описания функций.

Таблица 1.15

Описание функции	Комментарий
<code>private int fun(int i,double a) {int k=15; . . .</code>	Функция с двумя параметрами, с именем <code>fun</code>

<code>return k; }</code>	
<code>void fun2 (int i,double a) {...;}</code>	Процедура fun2 С 2 параметрами Тип не определен (void)
<code>public double fun3 () {..double b=a; . . . return b; }</code>	Открытая функция fun3 Без параметра
<code>public void fun4 () {double b=a; . . . ;}</code>	Открытая процедура без параметров
<code>void fun5() (double b=a; . . . ;</code>	Ошибка

Формальные параметры

Формальные параметры являются средством настройки подпрограммы на выполнение нужной работы. Например, обычная математическая функция $\sin(x)$ имеет формальный параметр x , трактуемый в теле функции как угол, для которого следует вычислить значение синуса.

В С# различают три разновидности (статуса) формальных параметров: входные, выходные и ссылочные. С помощью входных параметров программа передает данные в тело подпрограммы. Однако фактически в подпрограмму передаются не данные, а их копии, поэтому изменение входных параметров в подпрограмме никак не передается вызывающей программе. Выходные параметры объявляются с зарезервированным словом **out**. Они предназначены для передачи данных из подпрограммы в вызывающую программу. В теле подпрограммы обязательно должен присутствовать оператор присваивания параметрам нового значения — за этим следит компилятор. Ссылочные параметры передаются подпрограмме по ссылке, то в тело подпрограммы передается адрес параметра в памяти компьютера. Такие параметры помечаются зарезервированным словом **ref**. Они позволяют, как передать данные в подпрограмму, так и получить из нее новые данные.

В подпрограмме можно объявить не один, а несколько параметров. В этом случае соседние параметры разделяются запятыми используя список параметров. Последним (или единственным) параметром списка может объявляться массив любого типа с зарезервированным словом **params**. В этом случае на месте такого параметра при **вызове** подпрограммы может стоять сколько угодно параметров данного типа.

Вызов подпрограмм

Для вызова подпрограммы нужно указать ее имя и список фактических параметров (если в описании есть непустой список формальных параметров). Такой список должен по типу, количеству, порядку следования и статусу соответствовать списку формальных параметров.

Если в списке n формальных параметров, то фактических параметров должно быть не меньше n (соответствие по количеству). Каждому i -му формальному параметру ставится в соответствие i -й фактический параметр. Последнему формальному параметру при условии, что он объявлен с зарезервированным словом `params`, ставятся в соответствие все оставшиеся фактические параметры (соответствие по порядку).

Пример 1: Написать **процедуру** `Koren` для расчета корней квадратного уравнения.

Разработка приложения:

1. Запустите среду разработки Microsoft Visual Studio 2008 в консольном режиме для C#.
2. Создайте процедуру в разделе класса `Program` с именем `Koren`:

```
class Program
{
    static void Koren (double a,double b,double d)
    {
        double x1, x2;
        x1 = (-b + Math.Sqrt(d)) / (2 * a);
        x2 = (-b - Math.Sqrt(d)) / (2 * a);
        Console.Write("1 корень -" + x1.ToString("N") + ", 2 корень - " +
x2.ToString("N"));
    }
}
```

3. Внесите следующий код программы в раздел **`static void Main(string[] args)`**:

```
static void Main(string[] args)
{
    double a,b,c,d;
    a=Convert.ToDouble(Console.ReadLine());
    b =Convert.ToDouble(Console.ReadLine());
    c=Convert.ToDouble(Console.ReadLine());
    d=Math.Pow(b,2)-4*a*c;
    if (d>0)
        Koren(a,b,d); // вызов процедуры Koren
    else
        Console.Write ("Корней нет");
    Console.ReadKey();
}
```

4. На экране после запуска на выполнение (F5) появится следующая информация (рисунок 1.13):

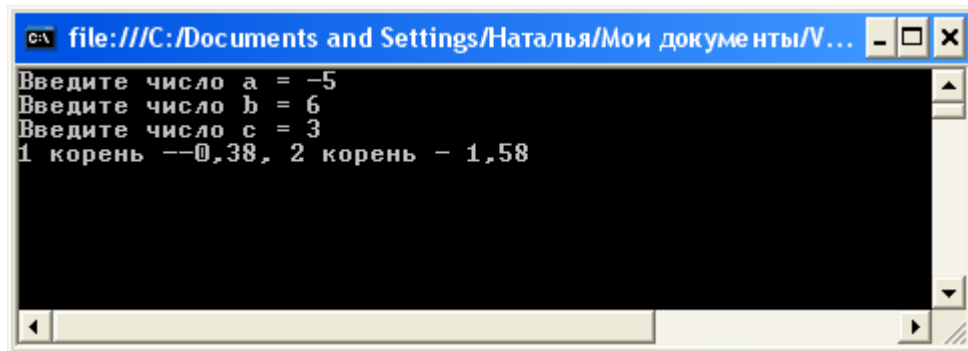


Рисунок 1.13 – Окно выполнения программы

Пример 2: Написать функцию для нахождения среднего арифметического из 3 вещественных чисел.

Разработка приложения:

1. Запустите среду разработки Microsoft Visual Studio 2008 в консольном режиме для C#.
2. Создайте функцию в разделе класса Program с именем Sred:

class Program

```
{  
    static double Sred(double a, double b, double c)  
    {  
        double x1;  
        x1 = (a+b+c)/3;  
        return x1;  
    }  
}
```

3. Внесите следующий код программы в раздел **static void Main(string[] args):**

```
static void Main(string[] args)  
{  
    double a, b, c, d;  
    Console.Write("Введите число a = ");  
    a = Convert.ToDouble(Console.ReadLine());  
    Console.Write("Введите число b = ");  
    b = Convert.ToDouble(Console.ReadLine());  
    Console.Write("Введите число c = ");  
    c = Convert.ToDouble(Console.ReadLine());  
    double s = Sred(a,b,c); // вызов функции Sred с параметрами  
    Console.Write("Среднее арифметическое = "+ s.ToString("N"));  
}
```



```

    Console.ReadKey();
}

```

4. На экране после запуска на выполнение (F5) появится следующая информация (рисунок 1.14):

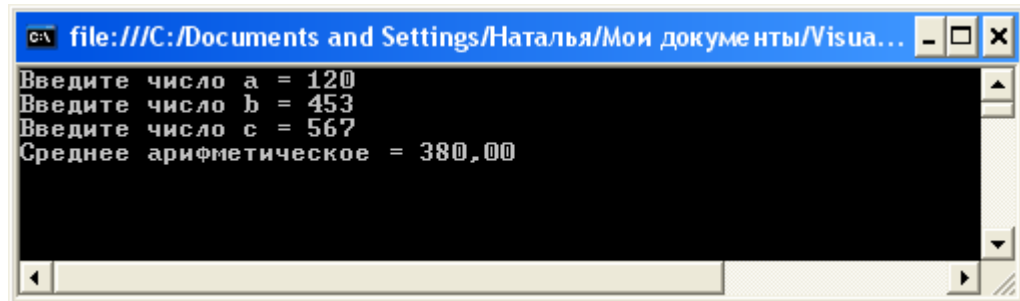


Рисунок 1.14 – Окно выполнения программы

Задание: В заданиях данной подгруппы требуется реализовать процедуры или функции с числовыми параметрами типа `int` и `double`. Входные параметры этих типов обычно описываются как параметры-значения.

Вариант 1.

Описать функцию `Min2(A,B)1|Max2(A,B)2` вещественного типа, находящую минимальное1|максимальное2 из двух вещественных чисел `A` и `B`. С помощью этой функции найти минимальные1|максимальные2 из пар чисел `A` и `B`, `A` и `C`, `A` и `D`, если даны числа `A`, `B`, `C`, `D`.

Вариант 2.

Описать процедуру `Minmax(A,B)`, записывающую в переменную `A` минимальное из значений `A` и `B`, а в переменную `B` — максимальное из этих значений (`A` и `B` — вещественные параметры, являющиеся одновременно входными и выходными). Используя четыре вызова этой процедуры, найти минимальное и максимальное из чисел `A`, `B`, `C`, `D`.

Вариант 3.

Используя процедуру `Minmax` из задания `Proc2`, описать функцию `Min3(A,B,C)1|Max3(A,B,C)2` вещественного типа, находящую минимальное1|максимальное2 из трех вещественных чисел `A`, `B` и `C`. С помощью этой функции найти минимальные1|максимальные2 из наборов `(A,B,C)`, `(A,B,D)`, `(A,C,D)`, если даны числа `A`, `B`, `C`, `D`.

Вариант 4.

Используя функцию `Min21|Max22` из задания `Proc1`, описать функцию `Min4(A,B,C,D)1|Max4(A,B,C,D)2` вещественного типа, находящую

минимальное1|максимальное2 из четырех вещественных чисел A, B, C и D. С помощью этой функции найти минимальные1|максимальные2 из наборов (A,B,C,D), (A,B,C,E), (A,C,D,E), если даны числа A, B, C, D, E.

Вариант 5.

Описать функцию Fact(N) целого типа, вычисляющую значение факториала $N! = 1 \cdot 2 \cdot \dots \cdot N$ ($N > 0$ — параметр целого типа). С помощью этой функции вычислить факториалы 10 данных чисел.

Вариант 6.

Описать функцию FactR(N) вещественного типа, позволяющую вычислять приближенное значение факториала $N! = 1 \cdot 2 \cdot \dots \cdot N$ для целых N (> 0). С помощью этой функции вычислить факториалы пяти данных чисел.

Вариант 7.

Описать функцию Fact2(N) целого типа, вычисляющую значение "двойного факториала": $N!! = 1 \cdot 3 \cdot 5 \cdot \dots \cdot N$, если N — нечетное, $N!! = 2 \cdot 4 \cdot 6 \cdot \dots \cdot N$, если N — четное ($N > 0$ — параметр целого типа). С помощью этой функции вычислить двойные факториалы десяти данных чисел.

Вариант 8.

Описать нерекурсивную функцию Fib(N) целого типа, вычисляющую N -е число Фибоначчи $F(N)$ по формуле: $F(1) = F(2) = 1$, $F(k) = F(k-2) + F(k-1)$, $k = 3, 4, \dots$. С помощью этой функции вычислить 10 чисел Фибоначчи с указанными номерами.

Вариант 9.

Описать процедуру SumDigit(N,S), находящую сумму цифр S целого числа N (N — входной, S — выходной параметр). Используя эту процедуру, найти суммы цифр пяти данных чисел.

Вариант 10.

Описать нерекурсивную функцию NOD2(A,B) целого типа, находящую наибольший общий делитель (НОД) двух натуральных чисел A и B , используя алгоритм Евклида: $\text{НОД}(A,B) = \text{НОД}(B \bmod A, A)$, если $A \neq 0$; $\text{НОД}(0,B) = B$. С помощью этой функции найти наибольшие общие делители пар A и B , A и C , A и D , если даны числа A, B, C, D .

Вариант 11.

Используя функцию NOD2 из задания Proc10, описать процедуру Frac(a,b,p,q), преобразующую дробь a/b к несократимому виду p/q (все параметры процедуры — целого типа). Знак результирующей дроби p/q приписывается числителю (т.е. $q > 0$). С помощью этой процедуры найти

несократимые дроби, равные $a/b + c/d$, $a/b + e/f$, $a/b + g/h$ (числа a, b, c, d, e, f, g, h даны).

Вариант 12.

Описать функцию $\text{Exp1}(x, \text{eps})$ вещественного типа (параметры x, eps — вещественные, $\text{eps} > 0$), находящую приближенное значение функции $\exp(x)$: $\exp(x) = 1 + x + x^2 / 2! + x^3 / 3! + \dots + x^n / n! + \dots$. В сумме учитывать все слагаемые, большие eps . С помощью Exp1 найти приближенное значение экспоненты для данного x при шести данных eps .

Вариант 13.

Описать функцию $\text{Sin1}(x, \text{eps})$ | $\text{Cos1}(x, \text{eps})$ вещественного типа (параметры x, eps — вещественные, $\text{eps} > 0$), находящую приближенное значение функции $\sin(x)$ | $\cos(x)$: $[\sin(x) = x - x^3 / 3! + x^5 / 5! - \dots + (-1)^n x^{2n+1} / (2n+1)! + \dots]$ | $[\cos(x) = 1 - x^2 / 2! + x^4 / 4! - \dots + (-1)^n x^{2n} / (2n)! + \dots]$. В сумме учитывать все слагаемые, большие по модулю eps . С помощью Sin1 | Cos1 найти приближенное значение синуса | косинуса для данного x при шести данных значениях eps .

Вариант 14.

Описать функцию $\text{Ln1}(x, n)$ | $\text{Arctg1}(x, n)$ вещественного типа (параметры x, eps — вещественные, $|x| < 1$, $\text{eps} > 0$), находящую приближенное значение функции $\ln(1+x)$ | $\arctg(x)$: $[\ln(1+x) = x - x^2 / 2 + x^3 / 3 - \dots + (-1)^n x^{n+1} / (n+1) + \dots]$ | $[\arctg(x) = x - x^3 / 3 + x^5 / 5 - \dots + (-1)^n x^{2n+1} / (2n+1) + \dots]$. В сумме учитывать все слагаемые, большие по модулю eps . С помощью Ln1 | Arctg1 найти приближенное значение $\ln(1+x)$ | $\arctg(x)$ для данного x при шести данных значениях eps .

Вариант 15.

Описать функцию $\text{PowerA}(x, a, \text{eps})$ вещественного типа (параметры x, a, eps — вещественные, $|x| < 1$, $a > 0$, $\text{eps} > 0$), находящую приближенное значение функции $(1+x)^a$: $(1+x)^a = 1 + a \cdot x + a \cdot (a-1) \cdot x^2 / 2! + \dots + a \cdot (a-1) \cdot \dots \cdot (a-n+1) \cdot x^n / n! + \dots$. В сумме учитывать все слагаемые, большие по модулю eps . С помощью PowerA найти приближенное значение $(1+x)^a$ для данных x и a при шести различных значениях eps .

Вариант 16.

Описать функцию $\text{Otr}(Ax, Ay, Bx, By)$ вещественного типа, находящую длину отрезка AB на плоскости по координатам его концов: $|AB| = \sqrt{(Ax - Bx)^2 + (Ay - By)^2}$ (Ax, Ay, Bx, By — вещественные параметры). С помощью этой функции найти длины отрезков AB, AC, AD , если даны координаты точек A, B, C, D .

Вариант 17.

Используя функцию `Otr` из задания `Proc16`, описать функцию `Perim(Ax,Ay,Bx,Bu,Cx,Cy)` вещественного типа, находящую периметр треугольника ABC по координатам его вершин ($A_x, A_y, B_x, B_y, C_x, C_y$ — вещественные параметры). С помощью этой функции найти периметры треугольников ABC, ABD, ACD, если даны координаты точек A, B, C, D.

Вариант 18.

Используя функции `Otr` и `Perim` из заданий `Proc16` и `Proc17`, описать функцию `Area(Ax,Ay,Bx,Bu,Cx,Cy)` вещественного типа, находящую `okny`d` треугольника ABC по формуле Герона: $S_{ABC} = \sqrt{p \cdot (p - |AB|) \cdot (p - |AC|) \cdot (p - |BC|)}$, где p — полупериметр. С помощью этой функции найти площади треугольников ABC, ABD, ACD, если даны координаты точек A, B, C, D.

Вариант 19.

Используя функции `Otr` и `Area` из заданий `Proc16` и `Proc18`, описать процедуру `Dist(Px,Py,Ax,Ay,Bx,Bu,D)`, находящую расстояние D от точки P до прямой AB по формуле $D = 2S_{PAB} / |AB|$, где S_{PAB} — площадь треугольника PAB . С помощью этой процедуры найти расстояния от точки P до прямых AB, AC, BC , если даны координаты точек P, A, B, C .

Вариант 20.

Используя процедуру `Dist` из задания `Proc19`, описать процедуру `Heights(Ax,Ay,Bx,Bu,Cx,Cy,hA,hB,hC)`, находящую высоты h_A, h_B, h_C треугольника ABC, проведенные соответственно из вершин A, B, C. С помощью этой процедуры найти высоты треугольников ABC, ABD, ACD, если даны координаты точек A, B, C, D.

1.6 Строки

Как и остальные типы в `C#`, строковый тип `String` является классом, то есть относится к ссылочным типам. Он предназначен для хранения строк переменной длины. Последнее обстоятельство определяет особенность значений типа: как и у всех других ссылочных типов, эти значения хранятся в куче, но при изменении значения строки в куче создается ее новый экземпляр, что не влияет на значения других переменных, содержащих прежнюю строку.

Для совместимости с `C (C++)` в `C#` определен псевдоним `string` типа `String`. Класс `String` имеет свойства и методы, представленные в табл.1.16 и 1.17.

Таблица 1.16 - Свойства класса `String`

Свойство	Описание
<code>public char this [int Index] {get:};</code>	Это свойство позволяет получить нужный символ из строки, рассматривая ее как 0 -базированный

	массив символов (только для чтения)
<pre>public int Length() {get:};</pre>	<p>Возвращает длину строки (только для чтения)</p> <p>Пример: S= “Текст программы”; int k = s.Length;</p>

Таблица 1.17 - Методы класса String

Метод	Описание
<pre>public static int Compare (string S1, string S2) ;</pre>	<p>Сравнивает 2 строки если S1 < S2, то отрицательное число если S1 = S2, то ноль если S1 > S2 ,то положительное число, Пример: if (String.Compare(s,s2)==0) Console.WriteLine("Строки одинаковые");</p>
<pre>public static int CompareOrdinal (string S1, string S2) ;</pre>	<p>Сравнивает две строки путем сравнения числовых значений их символов</p>
<pre>public static string Concat (params object[]);</pre>	<p>Объединяет строки Пример: s2 = String.Concat(s,s2);</p>
<pre>public void CopyTo (int SourceInd, char[] Dest, int DestInd, int Count) ;</pre>	<p>Копирует не более Count символов строки, начиная с символа SourceInd, в массив Dest Unicode-символов, начиная с символа DestInd Пример: s.CopyTo(0,4,s,1);</p>
<pre>public bool EndsWith (string S) ;</pre>	<p>Возвращает true, если строка заканчивается подстрокой S Пример: bool b=s.EndsWith(“catec”);</p>
<pre>public bool StartsWith (string S) ;</pre>	<p>Возвращает true, если текущая строка начинается подстрокой S</p>
<pre>public static string Format (strings, params object[]);</pre>	<p>Форматирует строку S</p>
<pre>public int IndexOf (string S) ;</pre>	<p>Возвращает индекс первого вхождения подстроки или символа S в текущую строку или отрицательное число, если S в строке не найден Пример: s=’подлесок в лесу’; int k = s.IndexOf("лес"); {k=3}</p>
<pre>public static int LastIndexOf (strings);</pre>	<p>Возвращает индекс последнего вхождения подстроки или символа S в текущую строку или отрицательное число, если S в строке не найден</p>

	Пример: s='подлесок в лесу'; int k = s.LastIndexOf("лес"); {k=11}
public string Insert (int Ind, string S);	Вставляет в текущую строку подстроку S, начиная с индекса Ind Пример: s=s.Insert(0, "привет");
public static string Join (strings, params string[] ss) ;	Объединяет разделителем S подстроки ss в одну длинную строку Пример: s = String.Join(" ", s, s1);
public string PaddLeft (int TotalWidth, char C) ;	Дополняет текущую строку слева символами C до тех пор, пока длина строки не станет равной TotalWidth
public static string Remove (int StartIndex, int count) ;	Удаляет из текущей строки не более Count символов, начиная с символа StartIndex Пример: s=s.Remove(0, 4); // удаление 4 букв с начала
public string Replace (strings, string ss);	Изменяет в текущей строке подстроку S на Ss Пример: s=s.Replace("f", "r"); // замена букв f на r
public string[] Split (params char [] delimiters);	Расщепляет текущую строку на лексемы в соответствии с ограничителями delimiters
public string Substring (int Beglnd, int Endlnt) ;	Возвращает из текущей строки подстроку, ограниченную символами Beglnd и Endlnt
public string ToLower () ;	Возвращает текущую строку с символами в виде строчных букв Пример: s='ПАПА'; s=s.ToLower(); {s='папа'}
public string ToUpper () ;	Возвращает текущую строку с символами в виде прописных букв Пример: s='папа'; s=s.ToUpper(); {s='ПАПА'}
public string Trim ();	Возвращает текущую строку, в которой удалены ведущие и ведомые пробелы Пример: s=s.Trim();
public string TrimEnd () ;	Возвращает исходную строку, в которой удалены ведомые пробелы Пример: s=s.TrimEnd();
public string TrimStart ();	Возвращает исходную строку, в которой удалены ведущие пробелы

Пример: s=s.TrimStart();

Работа с текстовыми строками

В то время как первые компьютеры создавались главным образом для выполнения математических вычислений, сейчас они активно применяются для работы с текстами, графикой, звуком и видео. При этом задача обработки текстовых символов и строк является одной из важнейших. Сегодня вы не найдете ни одной программы, в которой так или иначе не использовались бы текстовые строки.

Практически в любом языке программирования предусмотрен богатый набор функций, предназначенных для работы с символами и текстовыми строками. Эти функции позволяют объединять строки, вырезать из строк фрагменты, менять строчные буквы на прописные и обратно, искать в строках другие строки и произвольные последовательности символов и т. д.

В языках C и C++ стандартные функции, обрабатывающие символы и строки, являются частью библиотеки транслятора. Фактически они уже стали частью этих языков и потому описываются во всех учебниках, посвященных C и C++.

Что же касается языка программирования C#, то, как мы уже говорили, для представления текстовых строк в библиотеке классов Microsoft .NET Framework имеется класс System.String. Вместе с набором операций, методов и индексаторов этот класс представляет собой мощное средство обработки строк.

Помимо этого класса, существуют и другие, предназначенные для построения строк. Это, например, класс StringBuilder, а также класс Regex. Последний из них предназначен для использования так называемых регулярных выражений (regular expressions), представляющих собой мощнейшее средство обработки текста. Перед тем как мы приступим к изучению способов построения текстовых строк в C#, напомним, чем строки C# отличаются от строк, с которыми имеют дело программисты C, C++ и Pascal.

Прежде всего, программы, составленные на языках C, C++ и Pascal, имеют непосредственный доступ к представлению текстовых строк, хранящихся в оперативной памяти компьютера. Такие программы могут, например, перезаписывать или изменять содержимое строк по месту их расположения.

Что же касается программ C#, то все операции с текстовыми строками выполняются только при помощи методов, свойств, интерфейсов и индексов, предусмотренных в соответствующих классах библиотеки Microsoft .NET Framework.

Далее, программы C, C++ и Pascal могут работать с различными представлениями текстовых строк, такими, как, например ASCII и UNICODE. В зависимости от представления (или, как еще говорят, кодировки) для хранения каждого символа строки может использоваться 1, 2 или несколько байт памяти.

В языке C# текстовые строки представлены только в кодировке UNICODE, предполагающей представление одного текстового символа 2 байтами памяти.

Применение класса System.String

Методы и свойства класса System.String позволяют выполнять над строками все самые необходимые операции, такие, например, как сравнение, поиск подстроки, объединение строк, копирование символов из одной строки в другую, извлечение подстроки и т. д.

Ниже мы рассмотрим приемы выполнения этих операций и приведем примеры программ.

Создание строк

В предыдущей главе мы рассказывали вам о текстовых строках и литералах, а также приводили некоторые примеры их использования в программах C#.

До сих пор в наших программах встречались главным образом строки, созданные при помощи литералов или полученные при работе тех или иных методов (например, текстовые строки сообщений об ошибках). В классе System. String имеется набор перегруженных конструкторов, позволяющих создавать текстовые строки и другими способами.

Преобразование массива символов в строку

Если у вас есть массив символов UNICODE типа char, то его можно преобразовать в строку string, воспользовавшись для этого конструктором, предусмотренным специально для этой цели в классе System. String.

Рассмотрим пример программы, исходный текст которой приведен ниже.

Пример:


```

using System; namespace CreateString {
class CreateStringApp
{
    static void Main(string[] args)
    {
        char[] ch = { 'H', 'e', 'l', 'l', 'o', '!' };
        string s = new String(ch);
        Console.WriteLine(s);
        Console.ReadLine();
    }
}

```

Здесь мы объявили массив символов `ch` и проинициализировали его при помощи символьных литералов:

```
char[] ch = { 'H', 'e', 'l', 'l', 'o', '!' };
```

Чтобы преобразовать этот массив символов в текстовую строку, мы воспользовались конструктором класса `System.String`:

```
string s = new String(ch);
```

Отобразив полученную строку методом `WriteLine`, мы получим ожидаемый результат, а именно увидим на консоли строку «Hello!».

Создание строки на базе фрагмента массива

В предыдущей программе мы преобразовали в строку целиком весь массив символов. Существует способ создания текстовой строки на базе любого фрагмента такого массива.

Пример:

```

using System; namespace SubArray
{
class SubArrayApp
{
    static void Main(string[] args)
    {
        char[] ch= {'H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '! '};
        string s = new String(ch, 7, 6);
    }
}

```

```

        Console.WriteLine(s) ;
        Console.ReadLine();
    }
}

```

Здесь мы воспользовались конструктором класса `System.String`, имеющим три параметра:

```
string s = new String(ch, 7, 6);
```

Первый параметр задает ссылку на массив символов `UNICODE`, из которых нужно сделать строку. Второй и третий параметры определяют соответственно индекс первого элемента массива и количество символов, из которых будет создана строка.

В результате работы приведенной выше программы на консоль выводится только последние 6 символов массива, образующие строку «World!».

Заполнение строки символом

Иногда программисту может потребоваться создать строку, содержащую большое количество одинаковых символов. Типичный пример — создание разделителей для заголовков в листингах.

В программе, исходный текст которой представлен в примере, демонстрируются два способа решения этой проблемы.

Пример:

```

using System; namespace Fill
{
    class FillApp
    {
        static void Main(string[] args)
        {
            string header = "=====";
            string title= "Оформление заголовков";
            Console.WriteLine(header);
            Console.WriteLine(title);
            Console.WriteLine(header);
            string header1 = new String('-', 40);

```

```

        Console.WriteLine(header1);
        Console.WriteLine(title);
        Console.WriteLine(header1);
        Console.ReadLine();
    }
}

```

Первый способ предполагает «лобовое» решение. Мы используем для инициализации строки длинный текстовый литерал:

```
string header = "=====
```

Этот способ имеет, по крайней мере, два недостатка.

Во-первых, такие литералы могут загромождать листинг программы. Во-вторых, создавая литерал из повторяющихся символов, приходится подсчитывать количество этих символов вручную, что очень неудобно.

Другой способ основан на использовании одного из конструкторов, предусмотренных в классе `System.String` специально для инициализации подобных строк:

```
string header1 = new String('=', 40);
```

В качестве первого параметра этому конструктору нужно передать повторяющийся символ, а в качестве второго — количество повторов символа.

Копирование и клонирование строк

Если вам нужно создать копию текстовой строки, то это можно сделать при помощи обыкновенной операции присваивания. В этом случае в памяти будет находиться два одинаковых объекта.

Например, ниже мы объявили текстовые строки `src` и `si`, после чего скопировали первую строку во вторую:

```
string src= "Hello, World!"; string si  = src;
```

Теперь у нас есть два различных объекта, содержащих одинаковые текстовые строки. Для копирования строк можно также воспользоваться статическим методом `Copy`, определенным в классе `String`:

```
string s2 = String.Copy(src);
```

Результат будет такой же, как и при использовании оператора присваивания, — будет создан новый объект, содержащий исходную строку. В переменную s2 будет записана ссылка на этот объект.

Операция клонирования строк очень похожа на операцию копирования, однако она не приводит к созданию объекта-близнеца. Вместо этого создается еще одна ссылка на уже существующий объект:

```
string s3 = (string)src.Clone();
```

После выполнения этого кода в переменную s3 будет записана ссылка на строку src. Так как метод Clone возвращает данные базового типа object, нам здесь пришлось выполнить явное приведение типов.

Описанные выше операции копирования и клонирования демонстрируются в программе, исходный текст которой приведен в примере.

Пример:

```
using System;
namespace StringCopy
{
    class StringCopyApp
    {
        static void Main(string[] args)
        {
            string src = "Hello, World!";
            string si = src;
            string s2 = String.Copy(src);
            string s3 = (string)src.Clone();
            Console.WriteLine("Исходная строка: {0}", src);
            Console.WriteLine("Копия 1: (0)", si);
            Console.WriteLine("Копия 2: {0}", s2);
            Console.WriteLine("Клон: {0}", s2);
            Console.ReadLine();
        }
    }
}
```

Конкатенация строк

Иногда перед программистом встает задача конкатенации (объединения) текстовых строк. Эта операция выполняется следующим образом: вначале нужно объявить новую строку, а затем записать в нее объединяемые строки при помощи оператора + или метода String.Concat.

Вот как оператор + объединяет строки si и s2:

```
string si    = "Hello, string s2    = "World!";  
string res1 = si + s2;
```

В результате res 1 будет содержать строку «Hello, World!». Заметим, что таким образом можно объединять произвольное количество текстовых строк.

Другой способ объединения текстовых строк — применение метода String.Concat:

```
string res2 = String.Concat(si, s2);
```

Объединяемые строки передаются этому методу в качестве параметров. Метод String.Concat возвращает ссылку на новую строку, полученную в результате объединения исходных строк.

Существует несколько перегруженных методов String.Concat, позволяющих объединять до четырех текстовых строк:

```
string res3 = String.Concat(s1, s2, res1, res2);
```

Кроме того, метод String.Concat в состоянии объединить все строки, хранящиеся в массиве:

```
string[] array = { "Это ", "массив ", "строка" };  
string res4 = String.Concat(array);
```

После выполнения этого фрагмента кода в переменной res4 будет находиться строка «Это массив строка».

Извлечение подстроки

Чтобы создать новую строку на базе фрагмента существующей строки, можно воспользоваться методом Substring из примера ниже.

Пример:

```

using System;
namespace StringSubstring
{
    class StringSubstringApp
    {
        static void Main(string[] args)
        {
            string src = "Hello, World!";
            string res = src.Substring(7, 6);
            Console.WriteLine(res);
            Console.ReadLine();
        }
    }
}

```

Первый параметр метода `Substring` задает начальный индекс извлекаемой подстроки, а второй, необязательный, размер подстроки. Ниже в строку `res` будет записан фрагмент строки `src` длиной 6 символов и начинающийся с 7-й позиции:

```
string res = src.Substring(7, 6);
```

Если второй параметр метода `Substring` (определяющий размер подстроки) не задан, копируются все символы, начиная с исходной позиции и до конца строки.

Вставка подстроки

Одна из особенностей строк `C#` заключается в том, что прямая модификация существующей строки невозможна. Таким образом, если программа создала текстовую строку, то, чтобы изменить ее, придется создавать копию и уже потом переписывать туда символы исходной строки (в неизменном или модифицированном виде).

Специально для выполнения операции вставки одной строки в заданное место другой строки был предусмотрен метод `Insert`. Его использование демонстрируется в программе, исходный текст которой приведен в листинге 12.8.

Пример:

```

using System; namespace InsertString
{
class InsertStringApp
    {
        static void Main(string[] args)
            {
                string si = "Hello!";
                string res= si.Insert(5, ", World");
                Console.WriteLine(res);
                Console.ReadLineO ;
            }
    }
}

```

В качестве первого параметра методу Insert нужно передать индекс позиции исходной строки, в которую нужно вставить другую строку. Через второй параметр передается ссылка на вставляемую строку.

Здесь мы вставили запятую и слово «World» между словом «Hello» и восклицательным знаком:

```

string si = "Hello!" ;
string res= si.Insert(5, ", World");

```

В результате, как нетрудно догадаться, мы получим бессмертную фразу «HeHo, World!».

Замена символов и строк

Если вам нужно заменить в строке какой-либо символ другим или выполнить такую замену для последовательности символов, используйте метод Replace.

В качестве первого параметра этому методу нужно передать заменяемый символ или заменяемую последовательность символов. Вторым параметром задает новые символы, на которые необходимо заменить исходный символ или последовательность символов.

Пример использования этого метода приведен в листинге 12.9.

Пример:

```

using System; namespace Replace
{

```

```

class ReplaceApp
{
    static void Main(string[] args)
    {
        string str = "Hello, World!";
        string res = str.Replace("World", "C# World");
        string res1 = str.Replace("o", "0");
        Console.WriteLine(res);
        Console.WriteLine(res1);
        Console.ReadLine();
    }
}

```

Прежде всего мы меняем в исходной строке «Hello, World!» слово «World» на слова «C# World»:

```

string str = "Hello, World!";
string res = str.Replace("World", "C# World");

```

В результате получится фраза «Hello, C# World!». Далее мы меняем в полученной строке все букву «o» на цифру 0:

```

string res1 = str.Replace("o", "0");

```

В итоге получим фразу в «хакерском» стиле: «He110, W0rld!».

Удаление символов из строки

Теперь решим другую задачу — удалим из текстовой строки фрагмент с заданным начальным индексом и длиной.

Для решения этой задачи нам потребуется метод `Remove`).

Пример:

```

using System; namespace Remove
{
    class RemoveApp
    {
        static void Main(string[] args)
        {

```



```

        string str = "Hello, Kazakhstan!";
        string res = str.Remove(5, 7);
        Console.WriteLine(res);
        Console.ReadLine();
    }
}

```

В качестве первого параметра мы передаем методу Remove индекс запятой, а в качестве второго — число 7:

```
string str = "Hello, World!"; string res = str.Remove(5, 7);
```

В результате мы получим из строки «Hello, Kazakhstan!» строку «Hello !».

Удаление незначащих пробелов

При обработке строк, введенных пользователем, часто возникает задача удаления из полученной в результате ввода строки различных незначащих символов, или, как их еще называют, «белых пробелов» (white spaces). Это обычные пробелы, символы табуляции, символы возврата каретки, перевода строки и т. п.

Метод Trim позволяет вам удалить незначащие пробелы, расположенные как в начале, так и в конце текстовой строки. Ее применение демонстрируется в примере, исходный текст которой приведен ниже.

Пример:

```

using System; namespace Trim
{
    class TrimApp
    {
        static void Main(string[] args)
        {
            string str = "\t Hello,\tKazakhstan! ";
            string res = str.Trim();
            Console.WriteLine(res);
            Console.ReadLine();
        }
    }
}

```

Здесь метод Trim удалит символ табуляции и пробел, расположенный в начале строки `str`, а также пробел, находящийся в конце этой строки:

```
string str = "\t Hello,\tKazachstan! ";  
string res = str.Trim();
```

При этом символ табуляции, разделяющий слова фразы и расположенный в середине строки, останется нетронутым.

Помимо только что описанного метода Trim удаление незначащих пробелов можно выполнять методами TrimEnd и TrimStart. Они используются аналогично методу Trim. Первый из этих методов удаляет незначащие пробелы, расположенные в конце строки, а второй — в ее начале.

Преобразование к верхнему и нижнему регистру

Для преобразования всех символов строки к верхнему или нижнему регистру вы можете использовать методы ToUpper и ToLower соответственно.

Работа с этими методами демонстрируется в примере, представленной в листинге.

Пример:

```
using System;  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace ToLowerToUpper  
{  
    class ToLowerToUpperApp  
    {  
        static void Main(string[] args)  
        {  
            string str = "Республика Казахстан...";  
            string res1 = str.ToLower();  
            string res2 = str.ToUpper();  
            Console.WriteLine(res1);  
            Console.WriteLine(res2); Console.ReadLine();  
        }  
    }  
}
```

```
}  
}
```

Если запустить эту программу на выполнение, то на экране консоли появятся две строки, первая из которых состоит из букв нижнего регистра, а вторая — из букв верхнего регистра:

Так как в языке C# текстовые символы хранятся в универсальной кодировке UNICODE, преобразование регистра будет выполняться правильно не только для латинских символов, но и для символов других алфавитов.

Выравнивание по левому и правому краю поля

При форматном выводе текста иногда требуется выровнять текстовую строку по левому или правому краю поля заданной ширины.

Это можно сделать при помощи методов `PadLeft` и `PadRight` соответственно. В качестве первого параметра этим методам передается ширина поля (в символах), внутри которого необходимо выполнить выравнивание, а в качестве второго — символ-заполнитель (например, пробел).

Применение этих методов демонстрируется в примере, исходный текст которой представлен в листинге ниже.

Пример:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace ToLowerToUpper  
{  
    class PaddingApp  
    {  
        static void Main(string[] args)  
        {  
            string str = "Однажды, в студеную зимнюю пору...";  
            string res1 = str.PadLeft(80, '_');  
            string res2 = str.PadRight(80, '_');  
            Console.WriteLine(res1);  
            Console.WriteLine(res2);  
            Console.ReadLine();  
        }  
    }  
}
```

```
}
```

В качестве символа заполнителя мы применили символ подчеркивания. Вот что наша программа выведет на консоль после запуска:

Однажды, в студеную зимнюю пору...
Однажды, в студеную зимнюю пору...

Объединение массива строк

Для объединения строк, хранящихся в массиве, удобно использовать метод `String.Join`. В качестве первого параметра этому методу передается символ-разделитель, который будет добавлен после вставки каждой строки массива. Через второй параметр передается ссылка на объединяемый массив строк.

Третий и четвертый параметры необязательные. Они задают соответственно индекс первого элемента массива, с которого должно начинаться объединение, и количество объединяемых элементов. Если эти параметры не заданы, объединяется весь массив.

Пример использования этого метода вы найдете в примере, исходный текст которой представлен в листинге ниже.

Пример:

```
using System;
namespace Join
{
    class JoinApp
    {
        static void Main(string[] args)
        {
            string[] array = { "Это", "массив", "строка" };
            string res = String.Join(" ", array, 0, 3);
            Console.WriteLine(res);
            Console.ReadLine();
        }
    }
}
```

Массив строк определен в этой программе статически:

```
string[] array = { "Это", "массив", "строка" };
```

В качестве символа-заполнителя мы используем пробел:

```
string res = String.Join(" ", array, 0, 3);
```

Хотя мы объединяем все строки массива `array`, для примера мы указали необязательные параметры метода `String.Join`, определяющие первую ячейку массива и количество объединяемых ячеек.

Разбор строки

Метод `Split`, определенный в классе `String`, позволяет выполнять разбор строк, содержащих ключевые слова, отделенные друг от друга символами-разделителями.

Рассмотрим программу, предназначенную для разбора строки запуска утилит с параметрами вида «имя_параметра=значение».

Пример:

```
using System;
namespace Split
{
    class SplitApp
    {
        static void Main(string[] args)
        {
            string str = "myprg x=1,y=4,z=5";
            string[] resArray = str.Split(new Char[] { ',', ' ' });
            foreach (string res in resArray)
            {
                Console.WriteLine(res);
            }
            Console.ReadLine();
        }
    }
}
```

Исходная строка, подлежащая разбору, представлена ниже:

```
string str = "myprg x=1,y=4,z=5";
```

Здесь мы объявили строку, предназначенную для запуска программы с именем `myprg`, в которой этой программе передаются 3 параметра с именами `x`, `y`, и `z`. Разбор выполняется при помощи метода `Split`:

```
string[] resArray = str.Split(new Char() {',', ' '});
```

В качестве первого параметра этому методу нужно передать ссылку на массив символов-разделителей. В качестве таких символов мы указали здесь запятую и пробел. Метод `Split` разбирает строку, создавая массив из отдельных ключевых слов, обнаруженных в строке. При необходимости вы можете ограничить размер создаваемого массива, указав максимально допустимое значение верхней границы при помощи второго параметра метода `Split`.

После запуска описанной выше программы на консоли появится результат разбора строки:

```
myprg x=1 y=4 z = 5
```

Сравнение строк

Вы можете сравнивать строки таким же образом, как и числа. Об этом мы подробно рассказывали в главе, посвященной условным операторам.

Однако есть и еще одна возможность— использование статического метода `Compare`. Сравниваемые строки передаются этому методу в качестве параметров. Дополнительно методу `Compare` можно указывать, каким именно образом выполнять сравнение.

Если строки равны, метод `Compare` возвращает нулевое значение. Если первая строка меньше второй (используется лексикографическое сравнение), возвращается отрицательное значение, а если больше — положительное.

В программе, исходный текст которой приведен в примере, передается методу `Compare` - 3 параметра. Первые два из них представляют собой ссылки на сравниваемые строки, а третий, имеющий значение `true`, указывает, что при сравнении не следует учитывать регистр букв.

Пример:

```
using System;
namespace Compare
{
    class CompareApp
    {
        static void Main(string[] args)
        {
            for (; ; )
            {
                Console.Write("введите строку: ");
                string s = Console.ReadLine();
            }
        }
    }
}
```

```

        if (String.Compare(s, "exit", true) == 0) break;
    }
}
}
}

```

В нашей программе имеется бесконечный цикл, который прерывает свою работу, если ввести с консоли слово `exit`. Заметим, что это слово может быть набрано на любом регистре (так как при сравнении регистр не учитывается).

Класс `String` содержит несколько перегруженных методов `Compare`. Ниже мы привели прототип такого метода, предназначенного для лексикографического сравнения текстовых строк с учетом особенностей национального алфавита:

```

public static int Compare( String string1, Int32 index1, String string2, Int32
index2, Int32 length, Boolean ignoreCase, CultureInfo culture);

```

Параметры `string1` и `index1` задают соответственно первую сравниваемую строку и позицию внутри ее. Параметры `string2` и `index2` имеют то же назначение, относятся ко второй сравниваемой строке. Заметим, что параметры `index1` и `index2` можно не указывать. В этом случае происходит сравнение полных строк. С помощью необязательного параметра `length` можно задать максимальную длину сравниваемых строк.

Если значение параметра `ignoreCase` равно `true`, то метод `Compare` не будет учитывать регистр сравниваемых строк.

Однако самый интересный параметр— `culture`. С его помощью можно указать функции национальный алфавит, который должен применяться для лексикографического сравнения строк.

В примере представлено использование метода `Compare` для сравнения строк с учетом национального алфавита.

Пример:

```

using System;
namespace Culture
{
    class CultureApp
    {
        static void Main(string[] args)
        {

```

```

        string str1 = "Я из лесу вышел";
        string str2 = "Был сильный мороз";
        string str3 = "Hello, World!";
        string str4 = "Learning C# Now";
        Console.WriteLine(String.Compare(str1, str2, true, new
System.Globalization.CultureInfo("ru")));
        Console.WriteLine(String.Compare(str3, str4, true, new
System.Globalization.CultureInfo("en")));
        Console.ReadLine();
    }
}
}

```

Чтобы лексикографическое сравнение строк выполнялось правильно, необходимо учитывать особенности национальных алфавитов, использованных для представления строк. Библиотека классов .Net Framework содержит специальный класс System.Globalization.CultureInfo, позволяющий задать национальный язык, или, в терминологии Microsoft.NET Framework, культуру (culture).

Для обозначения культуры программист может использовать символическое имя или код. В таблице 1.18 мы привели имена и коды некоторых культур.

Таблица 1.18 - Имена и коды некоторых культур

Национальный язык, страна или район	Имя культуры	Код культуры
не учитывается	"" (пустая строка)	0x007F
Английский	en	0x0009
Английский (Канада)	en-CA	0x1009
Английский (Объединенное Королевство)	en-GB	0x0809
Английский (Соединенные Штаты Америки)	en-US	0x0409
Арабский	ar	0x0001
Белорусский	be	0x0023
Белорусский (Беларусь)	be-BY	0x0423
Испанский	es	0x000A
Испанский (Испания)	es-ES	0x0C0A
Итальянский	it	0x0010

Итальянский (Италия)	it-IT	0x0410
Казахский (Казахстан)	kz	0x0510
Китайский (Китай)	zh-CN	0x0804

Обратите внимание, что при сравнении строк `str1` и `str2`, содержащих символы кириллицы, мы указали русский алфавит. Для этого мы передали обозначение русского алфавита `ru` конструктору класса `System.Globalization.CultureInfo`. Латинский алфавит, использованный для сравнения строк `str3` и `str4`, обозначается как `en`.

Форматирование текстовых строк

Одна из важнейших задач, которую приходится решать программисту при разработке приложений любого типа, — форматирование текстовых строк. С помощью текстовых строк обычно представляется числовая информация, такая, как номера заказов, количество каких-либо предметов, цены, дата, время и т. д.

Если вы программировали раньше на языке C или C++, то вам знакомы такие средства форматирования, как функции `printf` и `sprintf`, а также управляющие символы в потоках вывода. Эти средства, ставшие стандартными, позволяют представить числа различных типов практически в любом формате.

Что же касается C#, то сам по себе этот язык не содержит средств форматирования строк. Однако богатейшие возможности такого форматирования предоставляются программисту в рамках библиотеки классов Microsoft .NET Framework. Мы уже использовали некоторые средства форматирования, предоставляемые методом `Console.WriteLine`, когда выводили в консольное окно шестнадцатеричные числа. Аналогичное форматирование доступно при формировании текстовых строк методом `String.Format`, не имеющим никакого отношения к консольному выводу.

Так как данные любого типа в C# являются объектами некоторых классов, это открывает новые возможности для форматного представления этих объектов в виде текстовых строк. В этом разделе мы рассмотрим этот вопрос применительно к форматному представлению чисел.

Заметим, что библиотека классов .Net Framework позволяет легко решить и обратную задачу — преобразование текстовых строк с числами в числовые значения. Это необходимо, например, для обработки чисел, введенных пользователями при помощи клавиатуры.

Представление целых чисел

Чтобы преобразовать целочисленное значение в текстовую строку с форматированием при помощи метода `String.Format`, необходимо задать этому методу так называемую строку формата (`format string`), а также передать в качестве параметров одно или несколько преобразуемых значений. В ответ данный метод возвратит отформатированную строку.

По принципу использования метод `String.Format` больше всего похож на функцию `sprintf`, знакомую всем программистам С.

Строка формата задается методу `String.Format` в следующем виде:

{ N [, M][: formatstring]}

Здесь число `N` задает номер преобразуемого аргумента, передаваемого методу `String.Format` в качестве параметра.

Необязательное число `M` задает ширину области текстовой строки (в символах), внутри которой необходимо поместить цифры преобразуемого значения. Если это число отрицательное, цифры числа выравниваются по левой границе данной области, а если положительное — по правой границе области.

И наконец, строка `formatstring` задает коды форматирования, которые мы скоро рассмотрим. Для форматирования целых чисел используются эквивалентные коды форматирования `d` и `D`.

Рассмотрим программу, исходный текст которой представлен в листинге. Эта программа демонстрирует применение различных способов форматирования для представления целых чисел в виде текстовых строк.

Пример:

```
using System;
namespace StringFormat
{
    class StringFormatApp
    {
        static void Main(string[] args)
        {
            int iSignedNumber = 777;
            string result;
            result = String.Format("{0}", iSignedNumber);
            Console.WriteLine(result);
            result = String.Format("{0:x}", 0x23fab);
            Console.WriteLine(result);
            result = String.Format("{0:X}", 0x23fab);
            Console.WriteLine(result);
            result = String.Format("{0:d2}", iSignedNumber);
```

```

        Console.WriteLine(result);
        result = String.Format("{0:d8}", iSignedNumber);
    Console.WriteLine(result);
        result = String.Format("{0,5:d}", iSignedNumber);
    Console.WriteLine(result);
        result = String.Format("{0,-5:d}3То счастливое число",
        iSignedNumber); Console.WriteLine(result); Console.ReadLine();
    }
}
}

```

Автоматическое форматирование

В самом простейшем случае можно вообще не указывать код форматирования, при этом будет использован формат по умолчанию:

```

int iSignedNumber = 777;
string results-result = String.Format("(0)", iSignedNumber);
Console.WriteLine (result);

```

В данном случае на консоль будет выведена строка 777. Аналогичного результата можно было бы добиться и следующим более простым образом:

```

Console.WriteLine("{0}", iSignedNumber);

```

Именно так мы выводили на консоль данные в примерах программ, приведенных в нашей книге. Заметим, однако, что этот способ подходит только для консольных программ. Что же касается программ с оконным пользовательским интерфейсом, то там, как правило, необходимо перед выводом преобразовывать числовые значения в текстовые строки. Эта задача и решается при помощи метода `String.Format`.

Представление чисел в шестнадцатеричном формате

Если вам нужно представить целочисленное значение в шестнадцатеричном формате, необходимо указать код форматирования `x` или `X`:

```

result = String.Format("{0:x}", 0x23fab);
Console.WriteLine(result);

```

В первом случае шестнадцатеричное число `0x23fab` будет отображаться с использованием нижнего регистра, а во втором — верхнего:

23fabC
23 FABC

При отображении шестнадцатеричных чисел префикс 0x не добавляется, так что вы можете выделять шестнадцатеричные числа любым способом по вашему усмотрению или не выделять их вовсе.

Определение ширины поля вывода

Непосредственно после кода форматирования вы можете указать ширину поля вывода в символах. При этом если в числе меньше цифр, чем ширина поля вывода, то при формировании строки это число будет дополнено слева нулями. Если же цифр больше, чем значение ширины вывода, то будут выведены все цифры числа.

Рассмотрим следующий пример, где мы форматируем трехзначное число:

```
int iSignedNumber = 777; string result;  
result = String.Format("{0:d2}", iSignedNumber);  
Console.WriteLine(result);  
result = String.Format("{0:d8} ", iSignedNumber);  
Console.WriteLine(result);
```

В результате работы этого фрагмента программы на консоли появятся следующие две строки:

```
777  
00000777
```

Несмотря на то что в первом случае мы указали в строке формата ширину поля, равную двум, в полученной текстовой строке присутствуют все цифры исходного числа. Таким образом, форматирование не может исказить значение числа.

Во втором случае ширина поля вывода составляет 8 символов, но в числе только 3 значащих цифры. Поэтому при форматировании это число было дополнено слева пятью нулями.

Ширина поля вывода может задаваться и при отображении числе в шестнадцатеричной системе счисления.

Выравнивание числа внутри поля вывода

Указав ширину поля вывода после номера аргумента (через запятую), можно выполнить выравнивание числа внутри этого поля.

Ширина поля вывода может задаваться как положительными числами, так и отрицательными:

```
result = String.Format("{0,5:d}", iSignedNumber);  
Console.WriteLine(result);  
result = String.Format("{0,-5:d}Это счастливое число", iSignedNumber);
```

В первом и втором случаях мы указали ширину поля вывода, равную пяти символам. Когда это число положительное, символы пробела добавляются слева, а когда отрицательное — справа:

```
777  
777 Это счастливое число
```

Таким образом, указывая ширину поля вывода, вы можете выравнивать отображаемые числа по правой или левой границе поля.

Представление чисел с фиксированной десятичной точкой.

Для представления чисел с фиксированной десятичной точкой используются равнозначные коды формата d и D. Вслед за кодом формата обычно указывают необходимое количество знаков после десятичной точки.

Пример программы форматного вывода чисел с фиксированной десятичной точкой приведен в листинге.

Пример:

```
using System;  
namespace Decimal  
{  
    class DecimalApp  
    {  
        static void Main(string[] args)  
        {  
            double pi = 3.1415926;  
            string result;  
            result = String.Format("{0}", pi);  
            Console.WriteLine(result);  
            result = String.Format("{0:f}", pi);  
            Console.WriteLine(result);  
        }  
    }  
}
```

```

        result = String.Format("{0:f3}", pi);
        Console.WriteLine(result);
        result = String.Format("{0,5:f3}", pi);
        Console.WriteLine(result);
        result = String.Format("{0:f10}", pi);
        Console.WriteLine(result);
        Console.ReadLine();
    }
}

```

Формат по умолчанию

Прежде всего наша программа выводит значение числа π , используя для этого форматирование по умолчанию:

```

double pi = 3.1415926; string result;
result = String.Format("{0}", pi); Console.WriteLine(result);

```

Данное значение будет выведено на консоль в следующем виде:
3,1415926

Можно использовать только код формата, не указывая количество знаков после десятичной точки:

```

result = String.Format("{0:f}", pi); Console.WriteLine(result);

```

В этом случае, однако, значение может быть округлено:

3, 14

Как результат, точность отображаемого значения может оказаться недостаточной.

Указание количества знаков после десятичной точки

Для получения предсказуемого результата мы рекомендуем всегда задавать необходимое количество знаков после десятичной точки:

```

result = String.Format("{0:f3}", pi);
Console.WriteLine(result);

```

В этом примере после запятой будет отображено 3 цифры:

3 , 142

Ограничение ширины поля вывода

Для ограничения ширины поля вывода можно задать необходимое значение через запятую после номера аргумента метода `String.Format`, как мы это делали для целых чисел:

```
result = String.Format("{0,5:f3}", pi);  
Console.WriteLine(result);
```

В результате на консоль будет выведено только 5 символов нашего числа (включая десятичную запятую): 3,142

Извлечение целой части числа

Если вам нужно отформатировать при выводе число с плавающей десятичной точкой таким образом, чтобы отобразить только целую часть числа, укажите количество цифр после десятичной точки, равное нулю:

```
result = String.Format("{0:f0}", pi);  
Console.WriteLine(result);
```

Этот фрагмент кода выведет на консоль число 3.

Избыточная точность при выводе

Если ширина поля вывода превышает количество значащих цифр в числе с фиксированной десятичной точкой, то такое число будет дополнено справа необходимым количеством нулей.

Например, здесь мы форматируем наше число для вывода в поле шириной 10 символов:

```
result = String.Format("{0:f10}", pi);  
Console.WriteLine(result);
```

При этом к числу будет дописано справа 3 нуля: 3,1415926000

Обратите внимание, что таким способом мы не увеличили точность представления числа π , а только добавили к числу дополнительные нули.

Представление чисел в научном формате

Научный, или экспоненциальный, формат обычно используется в научных расчетах для представления чисел, лежащих в большом диапазоне

значений. Для форматирования таких чисел применяются коды форматов `e` и `E`.

Пример, отображающей на консоли отформатированные числа в научном формате, представлен в листинге.

Пример:

```
using System;
namespace Science
{
    class ScienceApp
    {
        static void Main(string[] args)
        {
            double pi = 0.31415926E1;
            string result;
            result = String.Format("{0}", pi);
            Console.WriteLine(result);
            result = String.Format("{0:e}", pi);
            Console.WriteLine(result);
            result = String.Format("{0:e3}", pi);
            Console.WriteLine(result);
            Console.ReadLine();
        }
    }
}
```

Здесь мы будем выводить значение числа `pi`, представленного в научном формате:

```
double pi = 0.31415926E1;
```

Такие числа можно выводить с использованием формата по умолчанию:

```
result = String.Format("{0}", pi); Console.WriteLine(result);
```

Если при этом результат можно будет представить в формате с фиксированной десятичной точкой, то он и будет использован. Вот что вы увидите на консоли, запустив нашу программу:

```
3,1415926
```


3,141593e+000

3,142e+000

Первое число выводится с использованием формата по умолчанию. Во втором случае мы указали формат вывода явным образом:

```
result = String.Format("{0:e}", pi);  
Console.WriteLine(result);
```

И наконец, в третьем случае мы ограничили тремя количество цифр, отображаемых после запятой:

```
result = String.Format("{0:e3}", pi);  
Console.WriteLine(result);
```

Выделение тысяч при отображении больших чисел

При отображении больших многозначных чисел для большей наглядности часто отделяют друг от друга разряды тысяч при помощи пробела (по 3 разряда). Для подобного форматирования применяется код формата n или N. Рассмотрим пример.

Пример:

```
using System;  
namespace Delemiter  
{  
    class DelemiterApp  
    {  
        static void Main(string[] args)  
        {  
            double oneMByte = 1048576;  
            string result;  
            result = String.Format("{0:n}", oneMByte);  
            Console.WriteLine(result);  
            result = String.Format("{0:n4}", oneMByte);  
            Console.WriteLine(result);  
            result = String.Format("{0:n3}", oneMByte);  
            Console.WriteLine(result);  
            Console.ReadLine();  
        }  
    }  
}
```

Здесь мы сначала указываем код формата `n`, не дополняя его количеством цифр дробной части:

```
double oneMByte = 1048576; string result;  
result = String.Format("{0:n}", oneMByte);  
Console.WriteLine(result);
```

В результате наше число (количество байтов в одном мегабайте) будет отображено следующим образом: 1 048 576,00

При необходимости мы можем указать нужное количество цифр после запятой:

```
result = String.Format("{0:n4}", oneMByte);  
Console.WriteLine(result);  
result = String.Format("{0:n3}", oneMByte);  
Console.WriteLine(result);
```

В первом случае мы указали 4 цифры после запятой, а во втором — 3. Вот что получилось после запуска программы: 1 048 576,0000 1 048 576,000

Формат для представления денежных сумм

Специально для вывода значений денежных сумм предусмотрены форматы `c` и `C`. При использовании этих форматов числа выводятся с разделением разрядов на тысячи, а также с добавлением обозначения денежной единицы. Символ-разделитель, а также обозначение денежной единицы зависят от настройки локализации в управляющей панели ОС Microsoft Windows.

Пример программы, отображающей на консоли денежную сумму в различных форматах, приведен в листинге.

Пример:

```
using System;  
namespace Currency  
{  
    class CurrencyApp  
    {  
        static void Main(string[] args)  
        {  
            double someMoney = 1048576.25;
```

```

        string result;
        result = String.Format("{0:c}", someMoney);
        Console.WriteLine(result);
        result = String.Format("{0:C3}", someMoney);
        Console.WriteLine(result);
        Console.ReadLine();
    }
}

```

Вот что эта программа выводит на консоль:

1 048 576,25p. 1 048 576,25p. 1 048 576,250p.

Использование шаблонов при форматировании

Шаблоны открывают перед программистами дополнительные возможности форматирования чисел в процессе их преобразования в текстовые строки.

Шаблоны создаются с помощью символов 0, #, % и точки с запятой. Использование этих символов зависит от способа форматирования.

Форматирование целых чисел

При форматировании целых чисел символ 0 соответствует символу преобразуемой строки или нулевому значению (если данная позиция не используется).

Позиция, в которой стоит символ #, будет игнорироваться, если в шаблоне перед ней не стоит 0 или другой символ. В первом случае будет выведено нулевое значение или символ преобразуемой строки, а во втором — данная позиция будет пропущена или в ней будет символ преобразованной строки.

Рассмотрим пример, отображающий на консоли 7-значный телефонный номер.

Пример:

```

using System;
namespace Templatel
{
    class TemplatelApp
    {
        static void Main(string[] args)
        {
            uint number = 9367733; // только для примера!

```

```

        string result;
        result = String.Format("{0:#000-00-00}", number);
        Console.WriteLine(result);
        result = String.Format("{0:#0000-00-00}", number);
        Console.WriteLine(result);
        result = String.Format("{0:##00-00-00}", number);
        Console.WriteLine(result);
        Console.ReadLine();
    }
}

```

В первом случае мы выделяем в телефонном номере группы цифр, разделяя их дефисом:

```

result = String.Format("{0:#000-00-00}", number);
Console.WriteLine(result);

```

Вот что появится на консоли в результате выполнения этих строк:
936-77-33

Если нужно дополнить отображаемое целое число нулями с левой стороны, укажите необходимое количество нулей в левой позиции:

```

result = String.Format("{0:#0000-00-00}", number);
Console.WriteLine(result);

```

Теперь наш номер будет дополнен слева одним нулем: 0936-77-33

Если указать меньше нулей, чем надо, число все равно будет выведено полностью:

```

result = String.Format("{0:##00-00-00}", number);
Console.WriteLine(result);

```

Вот результат работы этого фрагмента программы: 936-77-33

Форматирование чисел с плавающей десятичной точкой

При форматировании чисел с плавающей десятичной точкой символы шаблона 0 и # применяются аналогично тому, как они применяются при форматировании целых чисел. Дополнительно для выделения тысяч в шаблоне применяется запятая.

Мы приведем пример использования нестандартного форматирования для решения такой распространенной задачи, как выделение тысяч при выводе чисел.

Пример:

```
using System;
namespace Template2
{
    class Template2App
    {
        static void Main(string[] args)
        {
            double number = 478903208236.567956;
            string result;
            Console.WriteLine(number);
            result = String.Format("(0:##,#.000000}", number);
            Console.WriteLine(result);
            Console.ReadLine();
        }
    }
}
```

Здесь мы указали в шаблоне между символами # символ запятой, в результате чего при выводе числа разряды тысяч будут выделены. Наша программа выводит число в исходном и отформатированном виде:

```
478903208236,568
478 903 208 236,568000
```

Обратите внимание, что мы задали отображение шести цифр после запятой. В результате наше число после округления было дополнено тремя незначащими нулями.

Форматирование чисел с процентами

Если к строке шаблона добавить символ процента (%), то перед выводом значение числа будет увеличено в 100 раз. Кроме того, будет выведен и сам символ %.

Этот прием форматирования демонстрируется в программе, исходный текст которой приведен в примере.

Пример:

```
using System;
```

```

namespace TemplatePercent
{
    class TemplatePercentApp
    {
        static void Main(string[] args)
        {
            double number = 0.4756; string result;
            Console.ReadLine();
        }
    }
}

```

Вот что эта программа выведет на консоль: 47, 6%

Заметим, что в одном шаблоне может встречаться несколько символов %. Каждый из них будет действовать описанным выше образом.

Форматирование с учетом знака чисел

Если для положительных, отрицательных и нулевых чисел нужно использовать разное форматирование, вы можете указать 3 шаблона, разделив их точкой с запятой. Первый из этих шаблонов будет использоваться для вывода положительных чисел, второй — отрицательных, а третий — равных нулю.

В примере мы привели исходный текст программы, демонстрирующий использование описанного выше форматирования.

Пример:

```

using System;
namespace TemplateSign
{
    class TemplateSignApp
    {
        static void Main(string[] args)
        {
            string result;
            result = String.Format("{0:плюс 0;минус 0;0}", 10);
            Console.WriteLine(result);
            result = String.Format("{0:плюс 0;минус 0;0}", 0);
            Console.WriteLine(result);
            result = String.Format("{0:плюс 0;минус 0;0}", -10);
            Console.WriteLine(result);
            Console.ReadLine();
        }
    }
}

```

```

    }
}

```

Вот что наша программа вывела на консоль после запуска:

плюс 10 0

минус 10

Как видите, мы смогли заменить символы + и - на слова плюс и минус соответственно.

Пример: Данный пример реализует возможности использования строк.

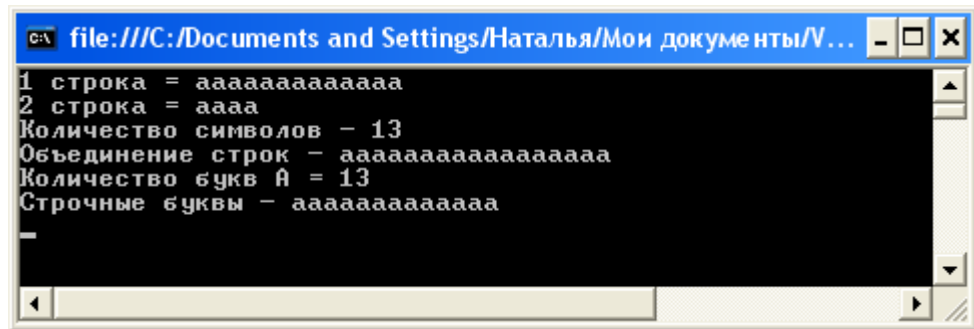
```

using System;
namespace TemplateSign
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("1 строка = ");
            string s = Console.ReadLine();
            Console.Write("2 строка = ");
            string s2 = Console.ReadLine();
            int k = s.Length;
            Console.WriteLine("Количество символов - " + k);
            if (String.Compare(s, s2) == 0)
                Console.WriteLine("Строки одинаковые");
            s2 = String.Concat(s, s2);
            Console.WriteLine("Объединение строк - " + s2);
            int l = 0;
            for (int i = 0; i < k; ++i)
            {
                if (s[i] == 'a')
                    ++l;
            }
            Console.WriteLine("Количество букв А = " + l);
            s.Replace("f", "r");           // замена букв f на r
            s.Remove(0, 4);               // удаление 4 букв с начала
            k = s.LastIndexOf("catec");
            s.Insert(0, "привет");
            s.ToLower();                  // преобразование в строчные буквы
            Console.WriteLine("Строчные буквы - " + s);
            Console.ReadKey();
        }
    }
}

```

}

На рисунке 1.15 представлен ход выполнения программы.



```
file:///C:/Documents and Settings/Наталья/Мои документы/V...
1 строка = аaaaaaaaaaaaaа
2 строка = аааа
Количество символов - 13
Объединение строк - аaaaaaaaaaaaaaaaaааа
Количество букв А = 13
Строчные буквы - аaaaaaaaaaaaaа
-

```

Рисунок 1.15 – Окно выполнения программы

Задания:

Вариант 1.

Вывести строку длины N (N — четное), которая состоит из чередующихся символов C1 и C2, начиная с C1.

Вариант 2.

Дана строка. Вывести строку, содержащую те же символы, но расположенные в обратном порядке.

Вариант 3.

Дана строка. Вывести коды ее первого и последнего символа.

Вариант 4.

Дана строка. Подсчитать количество содержащихся в ней цифр1[[прописных букв]2[[строчных букв]3.

Вариант 5.

Дана строка. Преобразовать все строчные1|прописные2 латинские3|русские4 буквы в прописные1|строчные2.

Вариант 6.

Дана строка. Если она представляет собой запись целого числа, то вывести 1; если вещественного (с дробной частью), то вывести 2; если строку нельзя преобразовать в число, то вывести 0.

Вариант 7.

Дано целое число. Вывести набор символов, содержащий цифры этого числа в исходном1|обратном2 порядке.

Вариант 8.

Дана строка S , изображающая вещественное число в формате с плавающей точкой, и целое число $N (> 0)$. Вывести набор символов, изображающих первые N цифр дробной части этого вещественного числа (без округления).

Вариант 9.

Дана строка, изображающая двоичную¹|десятичную² запись целого числа. Вывести строку, изображающую десятичную¹|двоичную² запись этого же числа.

Вариант 10.

Дана строка, изображающая целое число. Вывести сумму цифр этого числа.

Вариант 11.

Дана строка S и число N . Преобразовать строку S в строку длины N следующим образом: если длина строки S больше N , то отбросить первые символы, если длина строки S меньше N , то в ее начало добавить символы "." (точка).

Вариант 12.

Даны два числа: N_1 и N_2 , и две строки: S_1 и S_2 . Получить из этих строк новую строку, объединив N_1 первых символов строки S_1 и N_2 последних символов строки S_2 .

Вариант 13.

Даны две строки: S_1 и S_2 . Проверить, содержится ли строка S_2 в строке S_1 . Если да, то вывести номер позиции, начиная с которой S_2 содержится в S_1 , если нет, то вывести 0.

Вариант 14.

Даны две строки: S_1 и S_2 . Определить количество вхождений строки S_2 в строку S_1 .

Вариант 15.

Дана строка S и символ C . Удвоить каждое вхождение C в строку S .

Вариант 16.

Даны строки S_1 , S_2 и символ C . Перед¹|после² каждого вхождения символа C в строку S_1 вставить строку S_2 .

Вариант 17.

Даны две строки: S1 и S2. Удалить из строки S1 первую¹|последнюю²|все³ подстроки, совпадающие с S2. Если таких подстрок нет, то вывести S1 без изменений.

Вариант 18.

Даны три строки: S1, S2, S3. Заменить в строке S1 первое¹|последнее²|все³ вхождения строки S2 на S3.

Вариант 19.

Дана строка. Вывести подстроку, расположенную между первой и второй¹|последней² точками исходной строки. Если в строке менее двух точек, то вывести всю исходную строку.

Вариант 20.

Дана строка, состоящая из русских слов, разделенных пробелами (одним или несколькими). Определить количество слов в строке.

Тема 1.7 Препроцессорные средства

Главное, что следует помнить о препроцессоре C# — то, что он практически не существует. Препроцессорные средства C/C++ либо полностью отсутствуют в C#, либо реализуются в ограниченном виде. В частности, исчезли возможности включения файлов (директива `#include`) и макрозамены текста (`#define`). Директива `#ifdef` и сопутствующие директивы сохранились, они управляют условной компиляцией программы.

Избавление от макросов `#define` позволяет программисту лучше понять, что же происходит в программе. Любое незнакомое имя должно относиться к одному из пространств имен, и вам уже не придется устраивать длительные поиски во всех включаемых файлах.

Одна из основных причин подобного изменения состоит в том, что избавление от препроцессорной обработки и `#include` упрощает логику компиляции, и таким образом, заметно ускоряет компиляцию. Кроме того, программисту не приходится создавать отдельный заголовочный файл и обеспечивать его синхронизацию с файлом реализации.

При компиляции файлов с исходными текстами C# порядок компиляции отдельных файлов не важен. Результат будет точно таким же, как если бы все файлы были объединены в один большой файл. Вам не придется создавать опережающие объявления или следить за порядком размещения директив `#include`.

Препроцессорные директивы

Препроцессорные директивы C# перечислены в следующей таблице 1.19.

Таблица 1.19 - Препроцессорные директивы

Директива	Описание
<code>#define</code> <i>идентификатор</i>	Определяет идентификатор. Обратите внимание: идентификатору нельзя присвоить значение, можно лишь определить его. Идентификаторы также могут определяться в командной строке
<code>#undef</code> <i>идентификатор</i>	Отменяет определение идентификатора
<code>#if выражение</code>	Код следующей секции компилируется, если выражение истинно
<code>#elif выражение</code>	Конструкция else-if. Если условие предыдущей директивы не было выполнено и выражение истинно, компилируется код следующей секции
<code>#else</code>	Если условие предыдущей директивы не было выполнено, компилируется код следующей секции
<code>#endif</code>	Отмечает конец секции

Ниже приведен пример использования препроцессорных директив:

```
#define DEBUGLOG
using System;
class Test
{
    public static void Main()
    {
        #if DEBUGLOG
        Console.WriteLine("In Main - Debug Enabled");
        #else
        Console.WriteLine("In Main - No Debug");
        #endif
    }
}
```

Директивы `#define` и `#undef` должны предшествовать «полноценному» программному коду в файле, или произойдет ошибка. Предыдущий пример нельзя записать в следующем виде:

```
// Ошибка
using System;
class Test
{
    #define DEBUGLOG
    public static void Main()
    {
```

```

    #if DEBUGLOG
    Console.WriteLine("In Main - Debug Enabled");
    #else
    Console.WriteLine("In Main - No Debug");
    #endif
}
}

```

Препроцессорные выражения

Операторы, используемые в препроцессорных выражениях, перечислены в следующей таблице 1.20.

Таблица 1.20 - Препроцессорные выражения

Оператор	Описание
<i>! операнд</i>	Выражение истинно, если операнд ложен
<i>операнд == значение</i>	Выражение истинно, если операнд равен указанному значению
<i>операнд != значение</i>	Выражение истинно, если операнд не равен указанному значению
<i>операнд1 && операнд2</i>	Выражение истинно, если оба операнда истинны
<i>операнд1 операнд2</i>	Выражение истинно, если истинен хотя бы один из операндов

Выражения группируются при помощи круглых скобок:

```
#if !(DEBUGLOG && (TESTLOG || USERLOG))
```

Выражение во внешних скобках истинно, если определен один из идентификаторов TESTLOG или USERLOG, а также определен идентификатор DEBUGLOG. Затем значение этого выражения инвертируется оператором .

Дополнительные препроцессорные директивы

Помимо директив #if и #define, существуют и другие препроцессорные директивы.

#warning и #error

Директивы #warning и #error предназначены для выдачи предупреждений или сообщений об ошибках в процессе компиляции. При достижении компилятором строки, содержащей директиву #warning или #error, выводится весь текст, указанный за директивой.

Например, в программе может присутствовать следующая директива:

```
#warning Check algorithm with John
```

При компиляции этой строки будет выведен текст «Check algorithm with John».

```
#line
```

В директиве `#line` программист указывает имя исходного файла и номер строки, используемые компилятором при выводе сообщений об ошибках. Обычно эта директива применяется в автоматически сгенерированном коде для синхронизации выводимых номеров с другой системой нумерации или формирования имен.

ГЛАВА 2

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

2.1 Введение в объектно-ориентированное программирование

2.1.1 Классы и объекты

Класс — это шаблон, который определяет форму объекта. Он задает как данные, так и код, который оперирует этими данными. С# использует спецификацию класса для создания *объекта*.

Объекты — это *экземпляры* класса. Таким образом, класс — это множество намерений (планов), определяющих, как должен быть построен объект. Важно четко понимать следующее: класс — это *логическая абстракция*. О ее реализации нет смысла говорить до тех пор, пока не создан объект класса, и в памяти не появилось физическое его представление.

Методы и переменные, составляющие класс, называются *членами* класса.

Общая форма определения класса

Определяя класс, вы определяете данные, которые он содержит, и код, манипулирующий этими данными. Несмотря на то что очень простые классы могут включать только код или только данные, большинство реальных классов содержат и то, и другое.

Данные содержатся в переменных экземпляров, определяемых классом, а код — в методах. Однако важно с самого начала отметить, что класс определяет также ряд специальных членов данных и методов-членов, например статические переменные, константы, конструкторы, деструкторы, индексаторы, события, операторы и свойства. Пока мы ограничимся рассмотрением переменных экземпляров и методов класса, а к концу главы познакомимся с конструкторами и деструкторами. Остальные типы членов класса описаны в последующих главах.

Класс создается с помощью ключевого слова `class`. Общая форма определения класса, который содержит только переменные экземпляров и методы, имеет следующий вид:

```
class имя_класса {  
// Объявление переменных экземпляров.  
    модификатор тип переменная1;  
    модификатор тип переменная2;  
    //..  
    модификатор тип переменная N;  
  
// Объявление методов
```

```

модификатор тип_возврата метод1 {параметры)
    { // тело метода }
модификатор тип_возврата метод2(параметры)
    { // тело метода }
//..
модификатор тип_возврата методN(параметры)
    { // тело метода }
} // конец описания класса

// описание переменной объектного типа
имя_ КЛАССА имя _ОБЪЕКТА = new имя_КЛАССА;

```

Здесь:

Class – ключевое слово;

имя_класса – идентификатор, имя нового типа;

модификатор – уровень доступа;

имя_объекта – переменная объектного типа.

Обратите внимание на то, что объявление каждой переменной и каждого метода предваряется элементом *модификатор*. Здесь элемент *доступ* означает *спецификатор доступа*.

Доступ необходим для уточнения объявления класса. С его помощью определяется область видимости класса, возможность его наследования, готовность класса и отдельных его членов к работе. Модификаторы задаются зарезервированными словами, перечисленными в табл. 2.1.

Таблица 2.1 - Модификаторы класса и его членов

Модификатор	Пояснение
public	Класс или его член доступны из любой точки программы
internal	Класс (член) доступен в сборке, в которой он определен
protected	Класс (член) доступен потомкам и только им
private	Члены класса доступны только методам этого же класса
abstract	Абстрактный класс; должен обязательно перекрываться в потомках
sealed	Класс не может иметь наследников
static	Определяет статический член класса

Четыре первых модификатора последовательно сужают область видимости класса или его члена от полностью видимого (**public**) до закрытого (**private**). Если класс объявлен без модификатора доступа, считается, что он имеет область видимости **internal**. Но если член класса

объявлен вообще без модификатора, компилятор считает его закрытым (**private**).

Модификатор **abstract** сигнализирует компилятору, что класс не предназначен для непосредственного использования — попытка создать экземпляр абстрактного класса блокируется на этапе компиляции. Модификатор **sealed**, наоборот, предупреждает компилятор о невозможности создания потомка класса. Эти модификаторы широко используются в CTS. Абстрактными обычно объявляются классы, являющиеся родоначальниками родственных классов со схожей функциональностью. В этом случае абстрактный родитель инкапсулирует в себе общие для всех потомков поля, методы и события. С помощью модификатора **sealed** объявляются «полностью гоовые» классы (**sealed** — опечатанный, закрытый на ключ). Таким модификатором, кстати, снабжаются все примитивные классы — это исключает возможность перегрузки для них операций.

Модификатор **static** объявляет член класса статическим. Для доступа к статическому члену можно не создавать объект класса — статические члены принадлежат не отдельному объекту, а классу как новому. Если членом является поле или свойство, оно имеет одинаковое значение для всех экземпляров класса.

Пример описания класса:

```
class name
{
    private int i,j;           //данные
    private double a,b;       //данные
    private void funl(int k){. . .} //функция без возвр.знач.
    public name < ){. . . }    //конструктор
    ~name( ){. . . }          //деструктор
    public char cc,ccl;       //данные
    public int fun2( ){. . .}  //функция с возвр.значением
    public void fun3(int m){. . .} //функция без воэвр.знач.
} //конец описания класса
```

name ob=new name();//объявление объекта ob типа name

Классы, которые мы использовали в этой книге до сих пор, содержали только один метод — **Main ()**. Вскоре мы узнаем, как создавать и другие методы. Однако заметьте, что в общей форме определения класса метод **Main()** не задан. Он нужен только в том случае, если определяемый класс является отправной точкой программы.

Пример: Создать объект **ob** для возведения вещественного числа в определенную степень.


```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication17
{
    class math
    {
        public double power(double ai, int mn)
        {
            double a;
            if (mn == 0) a = 1.0;
            else
            {
                a = ai;
                for (int j = 1; j < mn; j++) a *= ai;
            }
            return a;
        }
    } // КОНЕЦ ОПИСАНИЯ КЛАССА

    class Program
    {
        public static void Main(string[] args)
        {
            math ob = new math(); // переменная объектного типа
            Console.Write("введите число - ");
            double D = Convert.ToDouble(Console.ReadLine()), rez;
            Console.Write("введите степень числа - ");
            int j = Convert.ToInt16(Console.ReadLine());
            rez = ob.power(D, j);
            Console.WriteLine("Число в степени = "+rez);
            Console.ReadKey();
        }
    }
}

```

На рисунке 2.1 представлен ход выполнения программы.

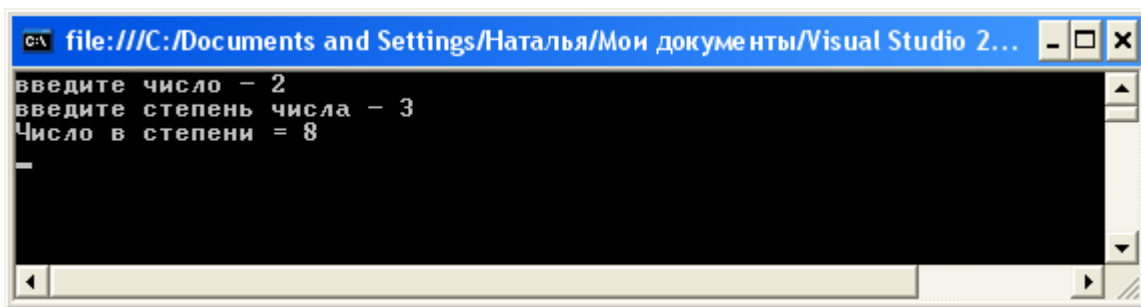


Рисунок 2.1 – Окно выполнения программы

Задание: Описать функцию в классе Name и создать объект, вызывающий данную функцию. Во всех заданиях данного пункта требуется вывести логическое значение True, если приведенное высказывание для предложенных исходных данных является истинным, и значение False в противном случае. Все числа, для которых указано количество цифр (двухзначное число, трехзначное число и т.д.), считаются целыми.

Вариант 1.

Проверить истинность высказывания: "Квадратное уравнение $A \cdot x^2 + B \cdot x + C = 0$ с данными коэффициентами A, B, C имеет вещественные корни".

Вариант 2.

Проверить истинность высказывания: "Данные числа x, y являются координатами точки, лежащей во второй координатной четверти".

Вариант 3.

Проверить истинность высказывания: "Данные числа x, y являются координатами точки, лежащей в первой или третьей координатной четверти".

Вариант 4.

Проверить истинность высказывания: "Точка с координатами (x, y) лежит внутри прямоугольника, левая верхняя вершина которого имеет координаты (x1, y1), правая нижняя — (x2, y2), а стороны параллельны координатным осям".

Вариант 5.

Проверить истинность высказывания: "Данное целое число является четным двухзначным числом".

Вариант 6.

Проверить истинность высказывания: "Данное целое число является нечетным трехзначным числом".

Вариант 7.

Проверить истинность высказывания: "Среди трех данных целых чисел есть хотя бы одна пара совпадающих".

Вариант 8.

Проверить истинность высказывания: "Среди трех данных целых чисел есть хотя бы одна пара взаимно противоположных".

Вариант 9.

Проверить истинность высказывания: "Сумма цифр данного трехзначного числа является четным числом".

Вариант 10.

Проверить истинность высказывания: "Сумма двух первых цифр данного четырехзначного числа равна сумме двух его последних цифр".

Вариант 11.

Проверить истинность высказывания: "Данное четырехзначное число читается одинаково слева направо и справа налево".

Вариант 12.

Проверить истинность высказывания: "Все цифры данного трехзначного числа различны".

Вариант 13.

Проверить истинность высказывания: "Цифры данного трехзначного числа образуют возрастающую последовательность".

Вариант 14.

Проверить истинность высказывания: "Цифры данного трехзначного числа образуют возрастающую или убывающую последовательность".

Вариант 15.

Проверить истинность высказывания: "Цифры данного трехзначного числа образуют арифметическую прогрессию".

Вариант 16.

Проверить истинность высказывания: "Цифры данного трехзначного числа образуют геометрическую прогрессию".

Вариант 17.

Даны координаты (как целые от 1 до 8) двух различных полей шахматной доски. Если ладья за один ход может перейти с одного поля на

другое, вывести логическое значение True, в противном случае вывести значение False.

Вариант 18.

Даны координаты (как целые от 1 до 8) двух различных полей шахматной доски. Если король за один ход может перейти с одного поля на другое, вывести логическое значение True, в противном случае вывести значение False.

Вариант 19.

Даны координаты (как целые от 1 до 8) двух различных полей шахматной доски. Если слон за один ход может перейти с одного поля на другое, вывести логическое значение True, в противном случае вывести значение False.

Вариант 20.

Даны координаты (как целые от 1 до 8) двух различных полей доски. Если ферзь за один ход может перейти с одного поля на другое, вывести логическое значение True, в противном случае вывести значение False.

2.1.2 Конструкторы и деструкторы

Конструктором называется специальный метод класса, занимающийся созданием объекта класса. В подавляющем большинстве случаев:

- имена конструкторов совпадают с именами классов;
- тип конструктора не указывается, хотя по сути своей конструктор является функцией;
- конструкторы объявляются с модификатором public, так как обычно они вызываются вне тела класса.

Если в классе не объявлен конструктор, компилятор снабжает такой класс умалчиваемым конструктором без параметров, но если конструктор указан, умалчиваемый конструктор не создается.

В следующем примере создаются и используются объекты е и ее класса Employee.

Иллюстрация конструкторов

```
using System;
```

```
class Employee
```

```
{
```

```
    public string fam;
```

```
    public int ID;
```

```
    public Employee(){} // Конструктор без параметров
```

```
    public Employee (string Fam, int id) // Конструктор с параметрами
```

```

    {
        fam = Fam;
        ID=Id;
    }
}
class ConstructorDemo
{
static void Main ()
    {
        // Создаем два объекта: один - конструктором без
        // параметров, второй - конструктором с параметрами:

        Employee e = new Employee(),
            ee = new Employee("Иванов", 1234);
        Console.WriteLine("Конструктор без параметров:");
        Console.WriteLine("fam = (0)", e.fam);
        Console.WriteLine("ID = {0}\n", e.ID);
        Console.WriteLine("Конструктор с параметрами:");
        Console.WriteLine("fam = (0)", ee.fam);
        Console.WriteLine("ID = (0)\n", ee.ID);
        Console.ReadLine ();
    }
}

```

Как правило, сложные многофункциональные классы получают несколько перегруженных конструкторов с различными наборами параметров. Вызов конструктора происходит в операторе **new**, где следом за зарезервированным словом указывается нужный конструктор.

В C# определены также деструкторы. В функциональном плане они должны осуществлять действия, обратные тем, что реализуют конструкторы. Однако особенностью деструкторов C# является то, что они не занимаются освобождением памяти, занятой соответствующим объектом, — за этим следит сборщик мусора. Деструкторы освобождают иные ресурсы, выделенные объекту, например, закрывают связанные с ним файлы.

Имя деструктора совпадает с именем класса, перед которым стоит знак ~ (тильда):

```
~Employee() {тело_деструктора}
```

Деструктор нельзя вызвать непосредственно в программе — он автоматически вызывается сборщиком мусора при удалении объекта из кучи. В теле деструктора невозможно освободить связанную с объектом память (в C# нет средств освобождения памяти), но следует освободить другие

ресурсы: файлы, связь с сервером базы данных, с другим компьютером и т.п. На практике освобождение такого рода ресурсов обычно осуществляется, как только надобность в них отпадает, поэтому деструкторы, как правило, создаются.

Вызов базового конструктора

В соответствии с принципом наследования классы могут создавать ветвящиеся иерархии типов. Например, добавим к обсуждавшемуся ранее классу Employee класс-наследник Manager:

```
class Manager: Employee
{
    public int numberOfOptions;
}
```

При создании экземпляра нового класса есть несколько возможностей. Одна из них — воспользоваться умалчиваемым конструктором:

```
Manager Man = new Manager();
Man.numberOfOptions = 25;
Man.fam = "Петров";
Man.ID = 222
```

Очевидно, этот метод не годится, если одно или несколько полей, нуждающихся в инициализации, недоступны. Если, например, поле numberOfOptions объявлено закрытым, придется изменять класс следующим образом:

```
class Manager: Employee
{
    int numberOfOptions;
    public Manager(string Fam, int id, int NO)
    {
        fam = Fam;
        ID = id;
        numberOfOptions = NO;
    }
}
```

В любом случае, множество параметров инициализации отношение ко вновь созданному, а к родительскому (базовому) классу, инициализировать поля базового класса можно следующим образом:

```
class Manager: Employee
{
```

```

int numberOfOptions;
public Manager(string Fam, int id, int NO): base(Fam, id)
{
    numberOfOptions = NO;
}
}

```

Таким образом, конструктор потомка принимает все параметры инициализации, но с помощью зарезервированного слова `base` передает часть параметров конструктору базового класса.

Виртуальные методы

Виртуализация методов — очень интересная возможность класса, позволяющая родительскому классу обращаться к методам своих наследников.

Чтобы продемонстрировать этот механизм, обратимся к примеру, изминавшемуся в предыдущей теме. Класс `GeomethricShape` является родоначальником классов `Line`, `Circle`, `Rectangle`. В родительском классе `GeomethricShape` инкапсулированы общие для потомков поля и методы. Целочисленные поля `x` и `y` хранят координаты реперной точки, однозначно определяющей положение геометрической фигуры на экране. Методы `Hide()` и `Show()` соответственно прячут и показывают фигуру, метод `Draw()` отрисовывает ее, а метод `MoveTo()` перемещает в новую точку. Анализ примера показывает, что каждый потомок должен переопределить конструктор для расширения полей и метод `Draw()`, специфичный для каждой фигуры. Все остальные методы родителя переопределять не надо: чтобы спрятать или показать фигуру, достаточно обратиться к методу `Draw()` с цветом фона (спрятать) или текущим цветом (показать); чтобы переместить фигуру, ее нужно сначала спрятать, а затем показать в новом месте.

Но обычное статическое переопределение метода `Draw()` в данном случае не решает проблему, так как остальные методы статически связаны с `Draw()`, определенным в родительском классе. Вот тут на помощь приходит динамическое замещение, которое называется виртуализацией метода.

Для этого замещаемый метод родителя объявляется с квалификатором `virtual`. Встретив это слово, компилятор создает таблицу виртуальных методов, в которую помещает их названия и адреса точек входа. Класс-наследник объявляет замещающий метод с квалификатором `override`. На этапе выполнения программы при обращении любого родительского класса к замещенному в таблицу виртуальных методов помещается ссылка на соответствующий метод наследника, который работает так, как если бы он был изначальной частью родительского класса.

I

Для примера показана программа, в которой объявляется родительский класс `Ancestor` и его наследник `Descendant`. В первом есть виртуальный

метод Get Str(), во втором этот метод замещается. Родительский метод Test() обращается к замещающему методу наследника как к своему собственному.

Иллюстрация виртуального метода

```
using System;
class Ancestor
{
    public virtual string  GetStr(){return "Родитель";}
    public void Test()
    {
        Console.WriteLine(GetStr());
    }
}

class Descendant: Ancestor
{
    public override string  GetStr(){return "Наследник";}
}
class VirtualDerao
{
    static void Mainf)
    {
        Ancestor A = new Ancestor*;
        Descendant D = new Descendant();
        A.Test();      // Вывод: Родитель
        D.Test(<);     // Вывод: Наследник
    }
}
```

Если убрать квалификаторы **virtual** и **override** из объявления методов GetStr(), на экран будут выведены две строки «Родитель».

Пример: Создание объекта об класса name

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication15
{
    class name
    { //ЗАКРЫТЫЕ ЭЛЕМЕНТЫ
        private double[] array;
        public name() { Console.WriteLine("конструктор"); } // конструктор
        void assign() //private по умолчанию
    }
}
```



```

    {array = new double[M];
    for (int m = 0, i = -3; m < M; m++, i++)
        array[m] = i * m * m * 0.21 * M;}
    ~name() {Console.WriteLine("деструктор"); }
// ОТКРЫТЫЕ ЭЛЕМЕНТЫ
    public int M;
    public void start()
    { assign(); } // вызов закрытого элемента}
    public void output()
    { for(int m=0; m<M; m++)
        Console.Write(" "+array[m]); }
    public double outputE(int m)
    {return array [m];}
} // КОНЕЦ ОПИСАНИЯ КЛАССА

class Program
{
    public static void Main(string[] args)
    {
        name ob = new name(); // объявление объекта ob
        ob.M=5; //доступ к данным
        ob.start();// вызов функции без возвр занчения
        ob.output();
        Console.WriteLine(ob.M);
        int i=2;
        double a=ob.outputE(i);
        Console.WriteLine (a);
        Console.ReadKey();
    }
}

```

На рисунке 2.2 представлен ход выполнения программы.

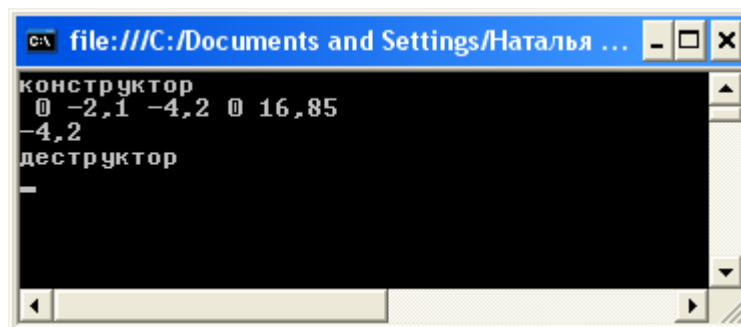


Рисунок 2.2 – Окно выполнения программы

Задание: Создать объект с конструктором и деструктором. Во всех заданиях вариантов предполагается, что исходный набор содержит ненулевое число элементов (в частности, число N всегда больше нуля).

Вариант 1.

Даны десять чисел. Вывести их среднее арифметическое.

Вариант 2.

Дано целое число N и набор из N вещественных чисел. Вывести сумму и произведение чисел из данного набора.

Вариант 3.

Дано целое число N и набор из N ненулевых целых чисел. Вывести в том же порядке все четные числа из данного набора и K таких чисел.

Вариант 4.

Дано целое число N и набор из N ненулевых целых чисел. Вывести в том же порядке номера всех нечетных чисел из данного набора и количество K таких чисел.

Вариант 5.

Даны целые числа K , N и набор из N целых чисел. Если в наборе присутствует число, меньшее K , то вывести True; в противном случае вывести False.

Вариант 6.

Дан набор ненулевых целых чисел; признак его завершения — число 0. Вывести количество элементов в наборе.

Вариант 7.

Дан набор ненулевых целых чисел; признак его завершения — число 0. Вывести сумму всех положительных четных чисел из данного набора. Если требуемые числа в наборе отсутствуют, то вывести 0.

Вариант 8.

Дано целое число K и набор ненулевых целых чисел; признак его завершения — число 0. Вывести количество чисел в наборе, меньших K .

Вариант 9.

Дано целое число K и набор ненулевых целых чисел; признак его завершения — число 0. Вывести номер первого числа в наборе, большего K . Если таких чисел в наборе нет, то вывести 0.

Вариант 10.

Дано целое число K и набор ненулевых целых чисел; признак его завершения — число 0. Вывести номер последнего числа в наборе, меньшего K . Если таких чисел в наборе нет, то вывести 0.

Вариант 11.

Дано целое число N и набор из N целых чисел. Вывести номера тех чисел в наборе, которые меньше своего левого соседа, и количество K таких чисел.

Вариант 12.

Дано целое число N и набор из N целых чисел. Вывести номера тех чисел в наборе, которые больше своего правого соседа, и количество K таких чисел.

Вариант 13.

Дано целое число N и набор из N вещественных чисел. Проверить, образует ли данный набор возрастающую последовательность. Если образует, то вывести True, если нет — вывести False.

Вариант 14.

Дано целое число N и набор из N вещественных чисел. Если данный набор образует убывающую последовательность, то вывести 0; в противном случае вывести номер первого числа, нарушающего закономерность.

Вариант 15.

Дано целое число N и набор из N целых чисел, содержащий по крайней мере два нуля. Вывести сумму чисел из данного набора, расположенных между первыми двумя нулями (если первые нули идут подряд, то вывести 0).

Вариант 16.

Дано целое число N и набор из N целых чисел, содержащий по крайней мере два нуля. Вывести сумму чисел из данного набора, расположенных между последними двумя нулями (если последние нули идут подряд, то вывести 0).

Вариант 17.

Даны целые числа K , N и набор из N вещественных чисел: A_1, A_2, \dots, A_N . Вывести K -е степени чисел из данного набора: $A_1^K, A_2^K, \dots, A_N^K$.

Вариант 18.

Дано целое число N и набор из N вещественных чисел: A_1, A_2, \dots, A_N . Вывести следующие числа: $A_1, A_2^2, \dots, A_{N-1}^{N-1}, A_N^N$.

Вариант 19.

Дано целое число N и набор из N вещественных чисел: A_1, A_2, \dots, A_N . Вывести следующие числа: $A_{1N}, A_{2N-1}, \dots, A_{N-12}, A_N$.

2.1.3 Наследование

Классы позволяют объединить вместе данные и методы, предназначенные для их обработки. Создав объекты того или иного класса, программа может вызывать методы и обращаться напрямую к полям объекта (если это разрешено при объявлении класса). Работая с объектами, созданными на базе классов, программисту нет нужды вникать во внутренние детали устройства класса— достаточно знать, как и какие методы можно вызывать, какие передавать им параметры и какие значения эти методы возвращают.

Механизм сокрытия, или, как говорят, инкапсуляции, данных и методов класса облегчает создание программ, особенно сложных, так как позволяет программисту не вникать во все детали реализации всех программных компонентов.

В этом разделе мы рассмотрим другое базовое понятие ООП, а именно *наследование*. С этим понятием неразрывно связаны два других понятия — *базовый класс* и *производный класс*.

Базовый класс

Наследование позволяет создавать новые классы на базе уже существующих классов. Создавая новый тип данных на базе типа данных, определенного ранее, можно упростить работу за счет использования разработанных ранее методов базового класса.

Базовым (base), или *родительским* (parent), классом называется класс, на основе которого создаются другие классы.

Чтобы это определение было понятнее, рассмотрим следующий пример.

Пусть нам нужно работать в программе с такими объектами, как прямоугольники. Для этого мы создаем класс **Rectangle**, инкапсулирующий в себе данные и методы, необходимые для размещения прямоугольника на плоскости:

```
class Rectangle {  
    int xPos; int yPos; int width; int height;  
    public Rectangle() {  
        xPos = yPos = width = height = 0;  
    }  
    public void SetPosition(int x, int y) {  
        xPos = x; yPos = y;  
    }  
    public void SetSize(int w, int h) {
```

```

width = w; height = h;
}
}

```

Поля класса **xPos** и **yPos** хранят координаты левого нижнего угла прямоугольника (в прямоугольной системе координат), а поля **width** и **height** — соответственно ширину и высоту прямоугольника. Таким образом, содержимое полей класса **Rectangle** однозначно определяет расположение прямоугольника на плоскости и его размеры (рисунок 2.3).

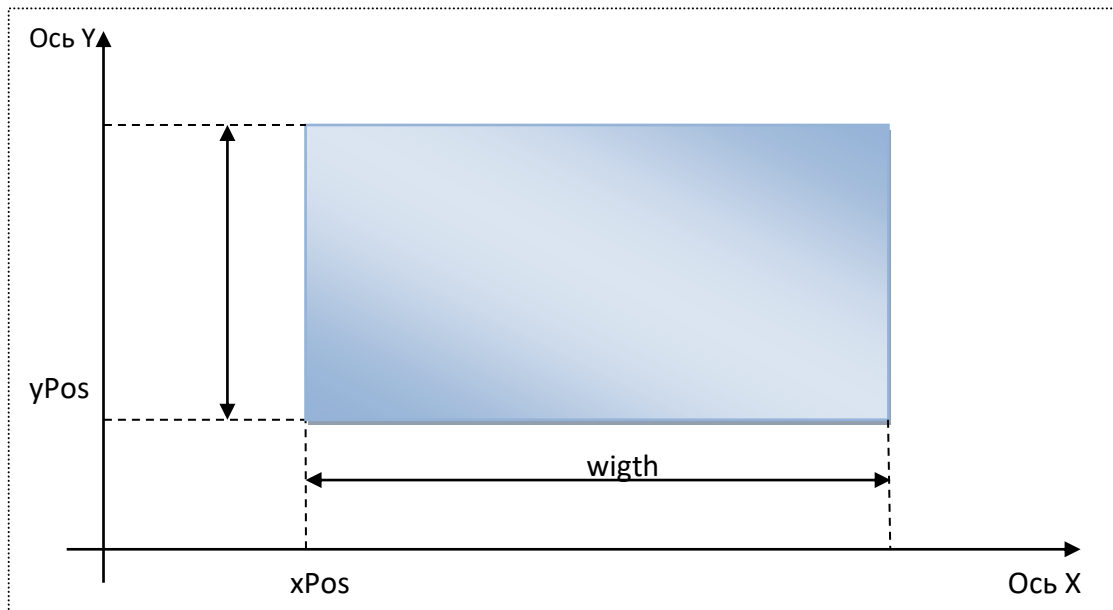


Рисунок 2.3 - Расположение прямоугольника на плоскости

Конструктор класса **Rectangle** записывает в поля класса нулевые значения, в результате чего все новые прямоугольники создаются в центре системы координат с размерами, равными нулю.

В классе **Rectangle** мы также определили методы **SetPosition** и **SetSize**, позволяющие установить соответственно расположение прямоугольника на плоскости и его размеры.

Работать с этим классом очень просто — нужно сначала создать прямоугольник, а затем установить его расположение и размеры:

```

Rectangle r;
r = new ColorRectangleO; r.SetPosition(0, 10); r.SetSize(20, 30);

```

Производный класс

Итак, мы создали базовый класс, описывающий прямоугольники. А теперь представьте себе, что вам понадобились не простые прямоугольники, а раскрашенные в разные цвета.

Вы, конечно, можете изменить класс **Rectangle**, добавив в него поля и методы для работы с цветом. Однако более красивое решение заключается в

создании на базе класса **Rectangle** нового класса **ColorRectangle**, дополненного средствами представления цвета:

```
class ColorRectangle : Rectangle
{
    byte colorR; byte colorG; byte colorB;
    public void SetColor(byte r, byte g, byte b)
    {
        colorR = r; colorG = g; colorB = b;
    }
}
```

Обратите внимание, что при объявлении класса **ColorRectangle** после двоеточия мы указали имя базового класса **Rectangle**. В результате класс **ColorRectangle** наследует все поля и методы своего базового класса, а также добавляет к ним еще 3 поля: **colorR**, **colorG**, **colorB** — и один метод — **SetColor**. Новые поля предназначены для хранения трех основных компонентов цвета (красный, голубой и зеленый), а метод **SetColor** позволяет установить значения этих полей, раскрасив наш прямоугольник.

Таким образом, мы очень легко добавили новую функциональность — теперь наши прямоугольники стали цветными. При добавлении новой сущности (цвета) нам были несущественны детали реализации базового класса **Rectangle**, такие, как способ размещения прямоугольника на плоскости и способ установки его размеров.

Более того, создавая свой собственный класс на базе готового класса, программист может даже и не подозревать о существовании в этом классе каких-то еще полей и методов. Например, в базовом классе **Rectangle** могло существовать поле для хранения запаха прямоугольника, а также метод для установки этого запаха.

Пока мы имеем дело с простыми классами и простыми программами, эффект от использования наследования может показаться небольшим. Однако в реальности наследование открывает перед программистом возможность создания своих классов на базе огромной библиотеки классов C#.

Взяв за основу один из десятков тысяч классов библиотеки, вы можете создать на его базе собственный класс, наделив его необходимыми вам свойствами. При этом вам не потребуется реализовывать полную функциональность, так как она уже реализована в базовом классе. Достаточно только добавить свои поля и методы (а также, возможно, *переопределить* существующие поля и методы базового класса, но об этом мы поговорим позже).

Класс **ColorRectangle**, созданный на базе класса **Rectangle**, называется *производным* или *дочерним* (derived, child). В свою очередь, класс **Rectangle** играет роль *базового* (base), или *родительского* (parent), класса.

Множественное наследование

Заметим, что в языке C# дочерний класс может наследовать свойства только одного базового класса. В других языках программирования

(например, в C++) допускается так называемое *множественное наследование*.

Множественное наследование позволяет создавать один производный класс на основе нескольких базовых классов, что бывает удобно в некоторых случаях, когда производный класс нужно наделить свойствами сразу нескольких базовых классов.

Язык C# не допускает множественного наследования, однако аналогичная функциональность может быть достигнута при использовании механизма так называемых *интерфейсов*. Интерфейсы C# мы рассмотрим в гл. 8.

Представление иерархии классов

Взаимосвязи между родительскими и дочерними (т. е. между базовыми и производными) классами могут быть достаточно сложными. Чтобы сделать эти взаимосвязи нагляднее, их часто иллюстрируют графическими диаграммами.

На рисунке 2.4 показана схему взаимосвязей базового класса **Rectangle** и производного класса **ColorRectangle**.

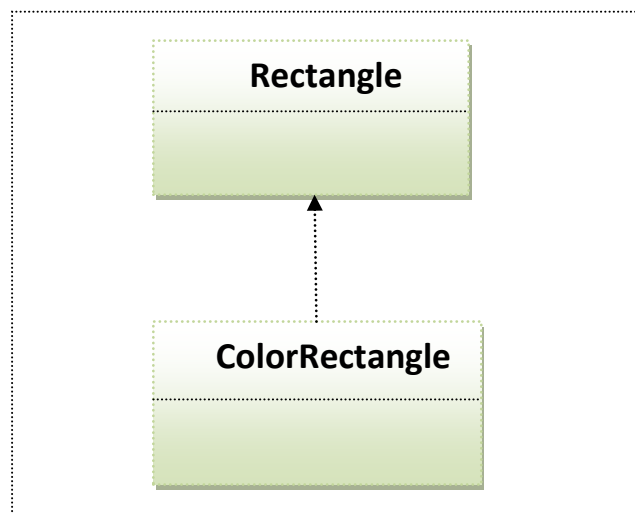


Рисунок 2.4 - Базовый и производный классы

Обратите внимание, что базовый класс располагается в верхней части диаграммы, а производный — в нижней части. Стрелка, обозначающая наследование, идет в направлении от производного класса к базовому классу.

Диаграммы зависимостей классов могут быть достаточно сложными. На рисунке 2.5 мы показали пример более сложной схемы, построенной для случая, когда производные классы базового класса **Rectangle** становятся базовыми для других классов. В этом случае наша схема превращается в *дерево иерархии классов*.

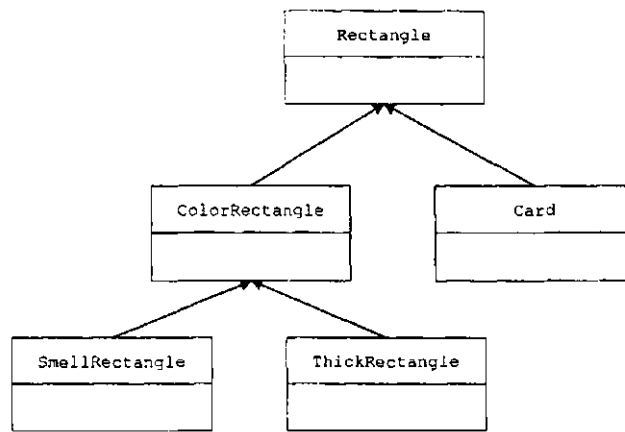


Рисунок 2.5 - Дерево иерархии классов

Как видно на рис. 3.3, на базе класса **Rectangle** созданы два класса — **ColorRectangle** и **Card**. О первом из них мы уже рассказывали — это класс для представления прямоугольников. Второй класс с именем **Card** может быть использован, например, для создания игровых карт. Помимо координат и размеров игральные карты имеют и другие атрибуты, такие, например, как масть.

Производный класс **ColorRectangle** служит базовым для классов **SmellRectangle** и **ThickRectangle**.

Класс **SmellRectangle** предназначен для представления прямоугольников с запахом, а класс **ThickRectangle** — прямоугольников, имеющих не только высоту и ширину, но и толщину.

Кстати, если вы думаете, что для программистов работа с запахами — дело весьма отдаленной перспективы, то знайте, что в прессе уже появляются сообщения о разработке устройств, генерирующих запахи. Подключив такое устройство к компьютеру, имеющему соединение с Интернетом, вы сможете не только посмотреть на Web-сайты, но и понюхать их.

Заметим, что не допускается рекурсивное или циклическое наследование классов, когда, например, второй класс наследуется от первого, а первый — от второго. Поэтому такая конструкция будет ошибочной:

```

class MyBaseClass : MyDerivedClass // Ошибка! {
}
class MyDerivedClass : MyBaseClass {
}
  
```

Важной особенностью библиотеки классов C# является то, что все классы этой библиотеки происходят от одного общего класса с названием **Object**. Как мы увидим дальше, это имеет далеко идущие последствия. В частности, такое наследование позволяет выполнять некоторые действия с объектами любых классов, созданных на базе **Object**. Программист может, например, помещать эти объекты в *контейнеры* (массивы, списки, словари), получать текстовое описание любого объекта и т. д.

Пример программы

В примере приведен исходный текст программы, в которой используется производный класс **ColorRectangle**. Базовым для него служит уже знакомый вам класс **Rectangle**.

Пример:

```
using System;
namespace Rectangle
{
    class Rectangle
    {
        int xPos; int yPos;
        int width;
        int height;
        public Rectangle()
        { xPos = yPos = width = height = 0; }

        public void SetPosition(int x, int y)
        { xPos = x; yPos = y; }

        public void SetSize(int w, int h)
        { width = w; height = h; }
    }
    class ColorRectangle : Rectangle
    {
        byte colorR; byte colorG; byte colorB;
        public void SetColor(byte r, byte g, byte b)
        { colorR = r; colorG = g; colorB = b; }
    }
    class RectangleApp
    {
        static void Main(string[] args)
        {
            ColorRectangle cr;
            cr = new ColorRectangle();
            cr.SetPosition(0, 10);
            cr.SetSize(20, 30);
            cr.SetColor(0, 0, 0xFF);
            Console.ReadLine();
        }
    }
}
```

Получив управление, метод **Main** создает объект производного класса

ColorRectangle cr;

```
cr = new ColorRectangle();
```

Далее этот метод вызывает два метода базового класса для изменения расположения и размеров прямоугольника:

```
cr.SetPosition(0, 10); cr.SetSize(20, 30);
```

Обратите внимание: в классе **ColorRectangle** нет определения методов **SetPosition** и **SetSize**, так как это методы базового класса. Тем не менее мы вызываем их для объекта **cr** производного класса. Модификатор доступа **public**, примененный при объявлении методов **SetPosition** и **SetSize** в базовом классе, допускает такой вызов.

Аналогичным образом мы вызываем метод **SetColor** производного класса:

```
cr.SetColor (0, 0, 0xFF) ;
```

Заметим, что, создав объект базового класса **Rectangle**, вы не сможете вызвать для него метод **SetColor**:

```
Rectangle rect;  
rect = new RectangleO;  
rect.SetColor(0, 0, 0xFF); // Ошибка!
```

Причина очевидна — в классе **Rectangle** нет объявления метода **SetColor**.

2.1.4 Полиморфизм

Наряду с инкапсуляцией и наследованием полиморфизм представляет собой одну из важнейших концепций ООП. Применение этой концепции позволяет значительно облегчить разработку сложных программ.

Термин **полиморфизм** имеет греческое происхождение и означает «наличие многих форм».

В программировании с полиморфизмом тесно связаны такие понятия, как абстрактные классы, виртуальные методы, перегрузка методов и операторов. Частично перегрузка методов уже была рассмотрена в предыдущей теме. Здесь же вы узнаете дополнительные подробности.

Применение полиморфизма

Мы будем изучать применение полиморфизма в программах **C#** на простом примере, имеющем некоторое отношение к геометрии. Пусть в нашем распоряжении имеются объекты различных типов — точка, прямая, прямоугольник, круг и т. д. Нашей задачей будет написать классы **C#**, с помощью которых программа может рисовать эти фигуры на плоскости.

Применение классов

Вначале мы попытаемся решить эту задачу известным нам на данный момент способом без использования полиморфизма.

Пример:

```
using System;
namespace Shape
{
    class Point
    {
        int x;
        int y;
        public Point(int x, int y)
        {
            this.x = x;
            this.y = y;
        }
        public void Draw(int x, int y)
        {
            Console.WriteLine("Рисование точки в ({0}, {1})", x, y);
            this.x = x;
            this.y = y;
        }
    }
    class Rectangle
    {
        int x;
        int y;
        int w;
        int h;
        public Rectangle(int x, int y, int w, int h)
        {
            this.x = x;
            this.y = y;
            this.w = w;
            this.h = h;
        }
        public void Draw(int x, int y)
        {
            Console.WriteLine("Рисование прямоугольника в ({0}, {1})",
            x, y);
            this.x = x;
            this.y = y;
        }
    }
    class ShapeApp
    {
        static void Main(string[] args)
```

```

    {
        Point pt = new Point(10, 25);
        pt.Draw(4, 7);
        Rectangle rect = new Rectangle(1, 4, 10, 20);
        rect.Draw(10, 12);
        Console.ReadLine();
    }
}

```

Мы будем работать с точками и прямоугольниками. Точки инкапсулируются в классе `Point`, а прямоугольники — в классе `Rectangle`.

Обратите внимание, как мы инициализируем поля класса `Point` в конструкторе:

```
this.x = x; this.y = y;
```

Здесь мы намеренно выбрали для параметров конструктора имена, совпадающие с именами соответствующих полей класса. При выполнении присваивания нам нужно помочь компилятору сделать различие между параметрами метода и полями класса, называемыми одинаково.

Эта задача решается с применением ключевого слова `this`. В данном контексте это ключевое слово означает текущий объект класса `Point`.

Если бы параметры конструктора и поля класса назывались по-разному, в использовании ключевого слова `this` не было бы никакой необходимости:

```

public Point (int xNew, int yNew)
{
    x = xNew;
    y = yNew;
}

```

Для того чтобы программа могла нарисовать точку и прямоугольник, мы определили в классах `Point` и `Rectangle` методы с названием `Draw`. Эти методы имеют одинаковые параметры, однако несколько различаются по своему действию.

Исходный текст метода `Draw`, предназначенного для рисования точки, представлен ниже:

```

public void Draw(int x, int y)
{
    Console.WriteLine("Рисование точки в ({0}, {1})", x, y);
    this.x = x;
    this.y = y;
}

```

```
}
```

Собственно рисование мы заменяем выводом на консоль строки сообщения о том, что точка нарисована в таком-то месте координатной плоскости. После этого метод изменяет текущие координаты точки. Обратите внимание, что здесь мы использовали ключевое слово `this`, чтобы компилятор смог сделать различие между названиями параметров метода и названиями полей класса.

Метод `Draw`, с помощью которого можно нарисовать прямоугольник, выглядит следующим образом:

```
public void Draw(int x, int y)
{
    Console.WriteLine("Рисование прямоугольника в ({0}, {1})", x, y);
    this.x = x;
    this.y = y;
}
```

От предыдущего метода он отличается текстом сообщения, отображаемого на консоли.

Теперь, когда у нас есть классы с методами `Draw`, можно приступать к рисованию фигур. Наша программа рисует фигуры, создавая по очереди объекты каждого класса и вызывая для этих объектов метод `Draw`:

```
Point pt = new Point (10, 25);
pt.Draw(4, 7);
Rectangle rect = new Rectangle(1, 4, 10, 20);
rect.Draw(10, 12);
```

Важно, что в первом случае вызывается метод `Draw` из класса `Point`, а во втором — метод `Draw` из класса `Rectangle`.

Попытка обобщения с помощью наследования

В предыдущем примере мы имели дело всего с двумя фигурами — точкой и прямоугольником. Всего же, как известно из школьного курса геометрии, существует великое множество разных фигур.

Зададимся вопросом, можно ли обобщить поведение различных фигур в одном базовом классе, отразив особенности каждой фигуры в соответствующем производном классе?

Создадим базовый класс `Shape`, призванный инкапсулировать в себе общие элементы поведения всех фигур. На базе класса `Shape` создадим классы `Point` и `Rectangle`, инкапсулирующих в себе особенности поведения точек и прямоугольников.

Исходный текст программы, реализующей данную структуру классов, представлен в листинге.

Пример:

```
using System;
namespace Shapel
{
    class Shape
    {
        protected int xPos; protected int yPos;
        public Shape(int x, int y)
        {
            xPos = x; yPos = y;
        }
    }
    class Point : Shape
    {
        public Point(int x, int y)
            : base(x, y)
        {
            Console.WriteLine("Рисование точки в ({0}, {1})", x, y);
            xPos = x; yPos = y;
        }
    }

    class Rectangle : Shape
    {
        int width; int height;
        public Rectangle(int x, int y, int w, int h)
            : base(x, y)
        {
            width = w; height = h;
        }
        public void Draw(int x, int y)
        {
            Console.WriteLine("Рисование прям-ка в ({0},{1})", x, y);
            xPos = x; yPos = y;
        }
    }
    class ShapelApp
    {
        static void Main(string[] args)
        {
            Point pt = new Point(10, 25); pt.Draw(4, 7);
        }
    }
}
```

```

        Rectangle rect = new Rectangle(1, 4, 10, 20);
        rect.Draw(10, 12);
        Console.ReadLine();
    }
}

```

Каждая фигура имеет свои координаты на плоскости. Положение точки однозначно определяется координатами по осям X и Y, а положение прямоугольника мы определяем по координатам его левого нижнего угла.

Пример:Создание 2 потомков (Potomok1 и Potomok2) класса BASE, которые реализуют одну и ту же задачу разными способами (метод MaxAB).

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

class BASE //базовый класс
{
    public double a;
    protected double b;
} // конец базового класса

class Potomok1:BASE // класс-потомок 1
{
    public void MaxAB(double a, double b) { if (a>b)Console.Write("Max="+a);
    else Console.Write("Max="+b);}
} // класс производного класса

class Potomok2:BASE // класс-потомок 1
{
    public void MaxAB(double a, double b)
    { Console.Write("Максимальное = "+Math.Max(a,b));}
}

class Program
{
    static void Main(string[] args)
    {
        double c = 1; double d=2;
        Potomok1 ob1 = new Potomok1(); // объект произ класса
        ob1.MaxAB(c, d);
        Potomok2 ob2 = new Potomok2(); // объект произ класса
        c = 3; d = 1;
        ob2.MaxAB(c, d);
        Console.ReadKey();
    }
}

```

```

    }
    }
}

```

2.2 Файловый ввод-вывод

В приложениях .NET Framework часто возникают две, в общем случае, схожие задачи: сохранить (прочитать) содержимое данных (файла) и сохранить (прочитать) текущее состояние объекта в файле или в таблице базы данных.

Классы обработки файлов сосредоточены в пространстве имен System.IO. Процесс сохранения текущего состояния объекта в памяти или на носителе информации называется *сериализацией* объекта, а обратный процесс — *десериализацией*. Эти процессы управляются классами, находящимися в пространстве имен System.Runtime.Serialization.

Классы для работы с файловой системой

Работа с файловой системой подразумевает обработку как папок (каталогов), так и зарегистрированных в них файлов. Соответствующие классы определены в пространстве имен System.IO.

В таблице 2.2 перечислены классы, использующиеся для работы с файловой системой.

Таблица 2.2 - Классы для работы с файловой системой

Класс	Назначение
Directory	Содержит статические методы для создания и использования папок. Эти методы можно вызывать без создания соответствующих объектов
DirectoryInfo	Содержит свойства и методы для создания и использования папок. Доступ к членам класса возможен только после создания его экземпляра
File	Содержит статические методы для работы с файлами. Эти методы можно вызывать без создания объекта
FileInfo	Содержит методы для работы с кассами, которые становятся доступными после создания объекта этого класса
Path	Вспомогательный класс для работы с маршрутами доступа (путями)

Статические классы Directory и File не требуют создания соответствующих объектов. Однако при обращении к их методам запускается *система проверки безопасности доступа к коду* (Code Access System, CAS), что замедляет работу этих методов. В связи с этим при обработке нескольких папок (файлов) удобнее использовать классы DirectoryInfo и FileInfo.

В 32-разрядных версиях Windows, как известно, для передачи данных между различными устройствами (оперативной памятью, дисковой памятью, сетью) используется концепция *потоков данных* (stream). Классы для работы с потоками данных также определены в пространстве имен System.IO и представлены в таблице 2.3

Таблица 2.3 - Классы для работы с потоками данных

Класс	Назначение
MemoryStream	Хранилище данных в оперативной памяти
NetworkStream	Позволяет передавать поток по сетевому соединению
FileStream	Представляет собой базовый класс потока для записи и чтения файлов
BinaryReader	Считывает данные из двоичных файлов
BinaryWriter	Записывает данные в двоичный файл
StreamReader	Считывает данные из текстовых файлов
StreamWriter	Записывает данные в текстовый файл

Создание и уничтожение папок

Создание и уничтожение папки проще всего осуществлять методами CreateDirectory() и Delete() класса Directory. Например:

using System.IO;

```

class Program
{
    static void Main(string[] args)
    {
        string DirPath = @"c:\C#_proba"; // Имя папки
        if (Directory.Exists(DirPath))    // Папка существует?
            Directory.Delete(DirPath);    // -Да. Уничтожаем ее
        else                               // -Нет. Создаем ее
            Directory.CreateDirectory(DirPath);
    }
}

```

Строка DirPath может содержать вложенные папки. Если они не существуют, то при выполнении метода CreateDirectory() они будут созданы.

Метод Delete() удаляет пустую папку. Если нужно уничтожить также пустые вложенные папки, ему передается параметр **true**.

Использовать методы класса DirectoryInfo для тех же целей менее удобно:

```

using System.IO;
class Program
{

```

```

static void Main()
{
    DirectoryInfo Dir = new DirectoryInfo(@"c:\C#_proba");
    if (!Dir.Exists())
        Dir.Create();
    else
        Dir.Delete(true);
}
}

```

При обращении к конструктору `DirectoryInfo()` ему передается маршрут доступа к создаваемой папке, но сама папка не создается. Лишь после проверки факта отсутствия папки она создается методом `Create()`.

Проверка существования папки необходима, так как попытка создать заново существующую папку или уничтожить несуществующую вызовет исключение.

При обращении к методу `Delete` ему передается логический параметр, указывающий на необходимость уничтожения вложенных папок.

Перемещение и копирование папок

Для перемещения папки используется метод `Move()` класса `Directory` или метод `MoveTo()` класса `DirectoryInfo`. В качестве параметров первому передаются имена обеих папок, а второму — только имя папки назначения:

```

Directory.Move("FromMove", "ToMove");
...
DirectoryInfo Dir = new DirectoryInfo("FromMove");
Dir.Create();
Dir.MoveTo("ToMove");

```

Фактически, методы `Move()` и `MoveTo()` не перемещают папку, а лишь переименовывают ее, причем обе папки должны находиться в одном и том же разделе одного и того же диска. После выполнения метода начальная папка исчезает, если ее имя отличается от имени папки назначения. Методы `Move()` и `MoveTo()` обычно используют для добавления вложенных папок в уже существующие. В этом случае имя папки назначения повторяет имя исходной папки и расширяет ее именами вложенных папок:

```

Directory.Move("FromMove", @"FromMove\SubFolder1\SubFolder2");

```

На практике значительно чаще возникает проблема физического изменения положения папки, например, перенос ее на другой дисковый носитель. Для этого необходимо скопировать содержимое исходной папки в папку назначения. Задача решается рекурсивным вызовом некоторой процедуры, которой передаются имена обеих папок. В процедуре просматривается содержимое исходной папки и ее файлы копируются в

папку назначения. Если среди файлов встречается вложенная папка, процедура вызывает сама себя (это и называется рекурсивным вызовом), но уже с другими именами папок.

При копировании нужно предусмотреть возможность отсутствия папки назначения, существование в ней копируемого файла и пр. Поскольку все это выходит за рамки обсуждаемой темы, в следующем примере возможность такого рода сопутствующих проблем игнорируется. Показанная в листинге программа копирует файлы из одной, поставляемой с Visual Studio 2008, папки в другую. В выбранной для копирования папке отсутствуют вложенные папки, что исключает необходимость рекурсии. Копируемые файлы заменяют одноименные файлы, существующие в папке назначения.

Пример:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace CopyDir
{
    class Program
    {
        static void Main(string[] args)
        {
            string FromDir =           // Откуда копируем
                @"d:\Program Files\Microsoft Visual Studio 8\ReportViewer";
            string ToDir = @"d:\ReportViewer\"; // Куда копируем
            if (!Directory.Exists(ToDir))      // Папка ToDir существует?
                Directory.CreateDirectory(ToDir); // -Нет. Создаем ее
            string[] Files; // Приемник имен файлов – динамический массив
            // Получаем все элементы копируемой папки:
            Files = Directory.GetFileSystemEntries(FromDir);
            // Цикл копирования:
            for (int k = 0; k < Files.Length; k++)
            {
                // Получаем имя очередного файла:
                string FromFile = Path.GetFileName(Files[k]);
                // Получаем его атрибуты:
                FileAttributes FileAttr = File.GetAttributes(Files[k]);
                if ((FileAttr & FileAttributes.Directory) ==
                    FileAttributes.Directory) // Это папка?
                    continue;                // -Да. Пропускаем ее
                Console.WriteLine(FromFile);
            }
        }
    }
}
```

```

        string ToFile = ToDir + FromFile;
        File.Copy(Files[k], ToFile, true); // Копируем файл
    }
    Console.ReadLine();
}
}
}

```

Во всех современных операционных системах (и в Windows в том числе) в файловой системе используются так называемые *таблицы размещения файлов* (File Allocation Tables, FAT). В этих таблицах указываются файлы корневого каталога и вложенные папки верхнего уровня. В FAT каждой вложенной папки, в свою очередь, указываются файлы и вложенные папки. В каждой строке FAT помимо имени файла (папки) указываются также дополнительные сведения: начальный кластер расположения файла (папки), дата создания и, в том числе, так называемые файловые атрибуты, которые задаются перечислением System.IO.FileAttributes. Значения этого перечисления указаны в таблице 2.4.

Таблица 2.4 - Значения перечисления FileAttributes

Значение	Описание
Archived	Обычный файл, доступный для копирования и удаления
Compressed	Сжатый файл
Device	Зарезервировано для будущего использования
Directory	Вложенная папка
Encrypted	Файл зашифрован (если речь идет о файле) или все включенные в папку файлы зашифрованы (если речь идет о папке)
Hidden	Скрытый файл
Normal	Обычный файл, для которого не могут быть установлены никакие другие атрибуты
NotContentIndexed	Файл не может быть индексирован службой индексации контекста операционной системы
Offline	Файл не подключен и его данные в настоящее время недоступны
ReadOnly	Файл, предназначенный только для чтения
ReparsePoint	Файл содержит реперную точку, то есть блок данных, ассоциированных с другим файлом или папкой
SparseFile	«Рыхлый» файл, большая часть которого содержит нули
System	Системный файл, который может использовать только операционная система
Temporary	Временный файл

Элементы перечислений в C# могут связываться с любыми целочисленными константами. Перечисление FileAttributes связывается с последовательностью констант 1, 2, 4, 8 и т. д., то есть файловым атрибутом может быть комбинация перечисленных значений. Используемая в примере проверка блокирует обработку любой папки:

```
if ((FileAttr & FileAttributes.Directory) ==  
    FileAttributes.Directory) continue;
```

В выбранной для копирования папке нет вложенных папок, поэтому указанная проверка не нужна. Но на практике для копирования не только файлов, но и всех вложенных папок вместо оператора **continue** следует вставить код, реализующий рекурсивный вызов процедуры обхода дерева каталогов (папок).

Исследование информации о строке в таблице FAT

Информация о единичной строке FAT хранится в свойствах класса FileSystemInfo. В таблице 2.5 перечислены эти свойства.

Таблица 2.5 - Свойства класса FileSystemInfo

Свойство	Описание
Attributes	Атрибуты
CreationTime	Время создания файла (папки)
CreationTimeUTC	Время создания в формате UTC (Universal Time Coordinated — всемирное скоординированное время). Время в формате UTC соответствует времени по гринвичскому меридиану и координируется с показаниями атомных часов
Exists	Признак существования
Extension	Расширение файла
FullName	Полное имя (с маршрутом доступа)
LastAccessTime	Время последнего обращения к файлу (папке)
LastAccessTimeUTC	Время последнего обращения в формате UTC
LastWriteTime	Время последнего изменения
LastWriteTimeUTC	Время последнего изменения в формате UTC
Name	Имя файла (папки)

Классы DirectoryInfo и FileInfo являются наследниками базового класса FileSystemInfo и получают все его свойства.

Пример иллюстрирует использование класса DirectoryInfo.

Пример:
using System;

```

using System.Collections.Generic;
using System.Text;
using System.IO;

namespace DirectoryInfo
{
    class Program
    {
        static void Main(string[] args)
        {
            DirectoryInfo DirInfo = new DirectoryInfo(@"c:\Program Files");
            Console.WriteLine("{0,30}{1}", "Полное имя:",
DirInfo.FullName);
            Console.WriteLine("{0,30}{1}", "Имя:", DirInfo.Name);
            Console.WriteLine("{0,30}{1}", "Время создания:",
                DirInfo.CreationTime);
            Console.WriteLine("{0,30}{1}", "Время создания UTC:",
                DirInfo.CreationTimeUtc);
            Console.WriteLine("{0,30}{1}", "Признак существования:",
                DirInfo.Exists);
            Console.WriteLine("{0,30}{1}", "Расширение:",
                DirInfo.Extension);
            Console.WriteLine("{0,30}{1}", "Время последнего доступа:",
                DirInfo.LastAccessTime);
            Console.WriteLine("{0,30}{1}", "Время последнего обновления:",
                DirInfo.LastWriteTime);
            Console.WriteLine("{0,30}{1}", "Атрибуты:", DirInfo.Attributes);
            Console.ReadLine();
        }
    }
}

```

Результат работы программы показан на рисунке 2.6.

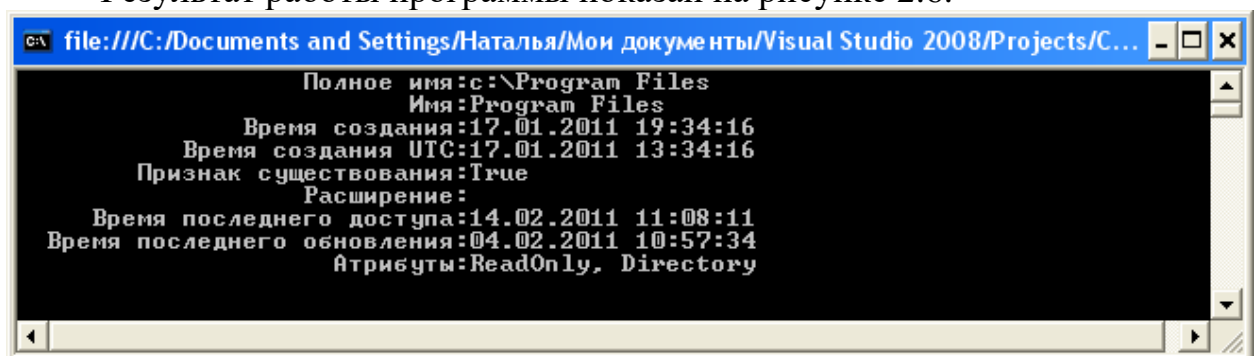


Рисунок 2.6 - Результат работы программы

Запись и чтение файлов

Техника работы с файлами зависит от типа файла (текстовый или двоичный).

Обработка текстовых файлов

При работе с текстовыми файлами сначала создается поток класса `FileStream`. Данные записываются в файл с помощью вспомогательного объекта класса `StreamWriter`, а читаются объектом класса `StreamReader`. В примере показан пример работы с текстовым файлом.

Пример:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace StringFileDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            // Запись в текстовый файл:
            FileStream Stream = new FileStream("strings.dat",
                                             FileMode.Create, FileAccess.Write);
            StreamWriter Writer = new StreamWriter(Stream);
            Writer.WriteLine("Мело, мело по всей Земле.");
            Writer.WriteLine("Во все пределы.");
            Writer.WriteLine("Свеча горела на столе.");
            Writer.WriteLine("Свеча горела...");
            Writer.Close();
            Stream.Close();
            // Чтение из файла:
            Stream = new FileStream("strings.dat",
                                   FileMode.Open, FileAccess.Read);
            StreamReader Reader = new StreamReader(Stream);
            string S;
            do
            {
                S = Reader.ReadLine();
                if (S != null)
                    Console.WriteLine(S);
            }
            while (S != null);
            Reader.Close();
        }
    }
}
```

```

        Stream.Close();
        Console.ReadLine();
    }
}

```

При создании объекта `FileStream` ему передаются имя файла и два параметра: `FileMode`, определяющий способ создания потока, и `FileAccess`, регулирующий доступ потока к данным. В таблицах 2.6 и 2.7 указываются допустимые значения этих параметров.

Таблица 2.6 - Значения параметра `FileMode`

Значение	Описание
Append	Добавляет записи в существующий файл или создает новый. Требуется, чтобы параметр <code>FileMode</code> имел значение <code>Write</code>
Create	Создает новый файл или переписывает существующий. Требуется, чтобы параметр <code>FileMode</code> имел значение <code>Write</code>
CreateNew	Создает новый файл, а если он уже существует, возникает исключение. Требуется, чтобы параметр <code>FileMode</code> имел значение <code>Write</code>
Open	Открывает существующий файл. Если файла нет, возникает исключение
OpenOrCreate	Открывает существующий или создает новый файл, если он еще не создан
Truncate	Открывает существующий файл и делает его размер равным нулю

Таблица 2.7 - Значения параметра `FileAccess`

Значение	Описание
Read	Поток может читать данные
ReadWrite	Поток может читать и записывать данные
Write	Поток может записывать данные

Для потока существует понятие текущей записи — в эту запись помещаются данные и из нее они считываются. Положением текущей записи можно управлять с помощью метода `Seek()`, имеющего такую сигнатуру:

public virtual long Seek(int Offset, SeekOrigin Origin);

Здесь `Offset` — смещение относительно позиции, указанной параметром `Origin`. Перечисление `SeekOrigin` может иметь значение, указанное в таблице 2.8.

Таблица 2.8 - Значения перечисления `SeekOrigin`

Значение	Описание
Вариант	Соответствует началу потока

Current	Соответствует текущей записи потока
End	Соответствует концу потока

Таким образом, указанный далее вызов сделает текущей запись с индексом 10 от начала потока (индексация начинается с 0):

```
Stream.Seek(10, SeekOrigin.Вариант );
```

Физическая запись данных в файл реализуется в момент закрытия потока методом `Close()` или выталкивания записей из промежуточного буфера методом `Flush()`. Во втором случае поток не закрывается и готов к продолжению операций.

При чтении строк из текстового файла нужно контролировать конец файла. Для этого переменная типа **string** в C# может принимать значение **null**, если из файла ничего не прочитано, то есть если файл исчерпан (см. показанный ранее пример). Другим способом контроля является обращение к методу `StreamReader.Peek()`, который возвращает положительное число, если файл не исчерпан, или `-1` в противном случае. Таким образом, цикл чтения записей из текстового файла предельно упрощается:

```
while (Reader.Peek() > 0)
    Console.WriteLine(Reader.ReadLine());
```

Обработка двоичных файлов

Обработка двоичных файлов во многом подобна обработке текстовых: сначала создается поток, затем — объекты `BinaryWriter` или `BinaryReader` в зависимости от направления передачи данных (в файл или из файла).

С помощью метода `Write()` объекта `BinaryWrite` данные передаются в файл. Метод `Write()` имеет множество перегруженных вариантов, позволяющих записывать в файл любые данные (в том числе строки **string**, так что деление файлов на строковые и двоичные носит чисто условный характер).

Объект `BinaryReader` имеет метод `Read()`, предназначенный для чтения из потока массива символов или байтов. Кроме того, он имеет многочисленные методы `ReadXXXX` (`ReadBoolean()`, `ReadByte()`, `ReadDouble()`, `ReadString()` и т. д.) для чтения значений примитивных типов. Его метод `PeekChar()` позволяет контролировать конец файла: подобно рассмотренному ранее методу `StreamReader.Peek()` он возвращает `-1`, если файл исчерпан.

Пример: Работа со строками. Ввести в файл 4 строки и вывести из файла записанные данные.

```
using System;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            FileStream file = new FileStream("lat.txt", FileMode.Create);
            StreamWriter fileW = new StreamWriter(file);
            Console.WriteLine("Введите 4 строки для записи в файл");
            for (int i = 1; i <= 4; i++) // запись в файл 4 строк lat.txt
            {
                fileW.Write(Console.ReadLine()); }
            fileW.Flush(); //очистка буфера памяти
            fileW.Close(); //файл file закрыт
            file = new FileStream("lat.txt", FileMode.Open);
            StreamReader fileR = new StreamReader(file);

            //файловая переменная для считывания
            string k; int m = 1;
            Console.WriteLine("Вывод данных из файла:");
            do
            {
                k = fileR.ReadLine(); //считывание из файла строк
                Console.WriteLine(k);
                ++m;
            } //ВЫВОД !!
            while (m < 4); //считывание до конца файла
            fileR.Close(); //файл file закрыт
            Console.ReadKey();
        }
    }
}

```

На рисунке 2.7 представлено окно выполнения программы.

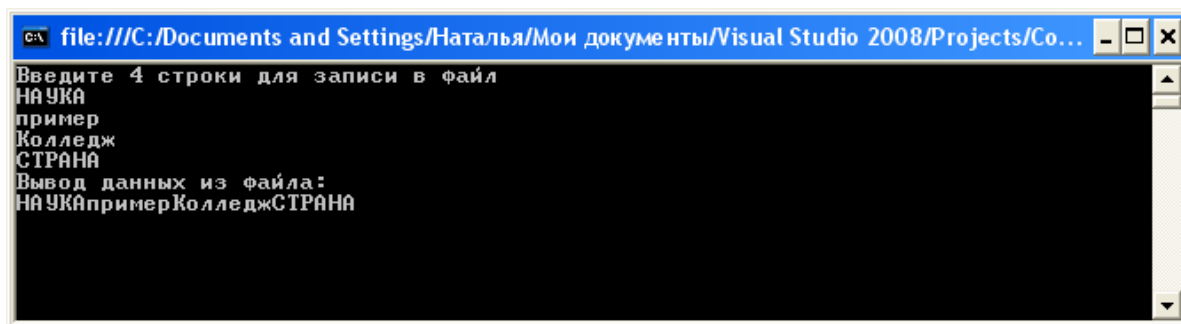


Рисунок 2.7 – Пример выполнения программы

Задание 1: Осуществить работу с файлами.

Вариант 1.

Дано целое число N и текстовый файл. Создать строковый файл, содержащий все слова длины N из исходного файла (знаки препинания в начале и в конце слов, не учитывать). Если исходный файл не содержит слов длины N , оставить результирующий файл пустым.

Вариант 2.

Дан символ C (прописная русская буква) и текстовый файл. Создать строковый файл, содержащий все слова из исходного файла, начинающиеся1|оканчивающиеся2 этой буквой (как прописной, так и строчной). Знаки препинания, расположенные в начале и в конце слов, не учитывать. Если исходный файл не содержит подходящих слов, оставить результирующий файл пустым.

Вариант 3.

Дано число N и текстовый файл. Удалить из файла абзац с номером N (абзацы отделяются друг от друга одной или несколькими пустыми строками и нумеруются от 1). Пустые строки, предшествующие и следующие за удаляемым абзацем, не удалять. Если абзац с данным номером отсутствует, то оставить файл без изменений.

Вариант 4.

Дано число N и текстовый файл. Удалить из файла абзац с номером N (абзацы выделяются с помощью красной строки (5 пробелов) и нумеруются от 1). Пустые строки между абзацами не учитывать и не удалять. Если абзац с данным номером отсутствует, то оставить файл без изменений.

Вариант 5.

Дан текстовый файл, каждая строка которого изображает целое число, дополненное слева и справа несколькими пробелами. Вывести сумму этих чисел и их количество.

Вариант 6.

Дан текстовый файл, каждая строка которого изображает целое или вещественное число, дополненное слева и справа несколькими пробелами (вещественные числа имеют ненулевую дробную часть). Вывести сумму x_1 вещественных x_2 чисел и их количество.

Вариант 7.

Дан текстовый файл, каждая строка которого содержит изображения нескольких вещественных чисел, разделенных пробелами. Создать файл вещественных чисел, содержащий эти числа в том же порядке.

Вариант 8.

Даны два текстовых файла с именами Name1 и Name2. Добавить в начало x_1 конец x_2 каждой строки файла Name1 соответствующую строку файла Name2. Если файл Name2 короче файла Name1, то оставшиеся строки файла Name1 не изменять.

Вариант 9.

Дан текстовый файл NameT и файл целых чисел NameN. Добавить в начало x_1 конец x_2 каждой строки файла NameT изображение соответствующего числа из файла NameN. Если файл NameN короче файла NameT, то оставшиеся строки файла NameT не изменять.

Вариант 10.

Дан текстовый файл с именем NameT. В каждой его строке первые 60 позиций отводятся под текст, а оставшаяся часть — под вещественное число. Создать два файла: строковый файл с именем NameS, содержащий текстовую часть исходного файла, и файл вещественных чисел с именем NameR, содержащий числа из исходного файла.

Вариант 11.

Даны два файла целых чисел одного размера с именами Name1 и Name2. Создать текстовый файл с именем NameT, содержащий изображения этих чисел, расположенные в два столбца шириной по 30 символов: первый содержит числа из файла Name1, второй — из файла Name2. В начале и конце каждой строки текстового файла ввести разделитель " | " (код 124). Числа выравниваются по левому x_1 правому x_2 краю столбца.

Вариант 12.

Даны вещественные числа A, B и целое число N. Создать текстовый файл, содержащий таблицу значений функции $f(x) = [\sin(x)]^1 [\cos(x)]^2 [\exp(x)]^3$ на промежутке [A, B] с шагом $(B - A)/N$. Таблица состоит из двух столбцов: с аргументами x (10 позиций, из них 3 под дробную часть) и со значениями f(x) (15 позиций, из них 8 под дробную

часть). Столбцы выравниваются по правому краю и разделяются 10 пробелами.

Вариант 13.

Дан текстовый файл с именем NameT, содержащий таблицу из трех столбцов вещественных чисел. Ширина столбцов таблицы и способ их выравнивания являются произвольными. Специальных символов-разделителей таблица не содержит. Создать файлы вещественных чисел с именами Name1, Name2 и Name3 каждый из которых содержит числа из соответствующей таблицы.

Вариант 14.

Дан текстовый файл, представляющий собой таблицу, состоящую из трех столбцов с целыми числами. В начале и в конце каждой строки таблицы, а также между ее столбцами располагается символ-разделитель. Ширина столбцов таблицы и способ их выравнивания являются произвольными. Создать файл целых чисел, содержащий сумму чисел из каждой строки исходной таблицы.

Вариант 15.

Дан текстовый файл. Создать символьный файл, содержащий все символы, встретившиеся в тексте, включая пробел и знаки препинания (без повторений). Символы располагать в порядке [возрастания их кодов]1|[убывания их кодов]2|[их первого появления в тексте]3.

Вариант 16.

Дан текстовый файл с именем NameT. Подсчитать число повторений в нем строчных русских букв ("а"–"я") и создать строковый файл с именем NameS, элементы которого имеют вид: "<буква>–<число повторений данной буквы>". Буквы, отсутствующие в тексте, в файл не включать. Строки упорядочить по [возрастанию кодов букв]1|[убыванию числа повторений букв, а при равном числе повторений — по возрастанию кодов букв]2.

Вариант 17.

Дано целое число N и текстовый файл с именем Name1, содержащий один абзац текста, выровненный по левому краю. Отформатировать текст так, чтобы его ширина не превосходила N позиций, и выровнять текст по левому1|правому2 краю. Пробелы в конце строк удалить. Сохранить отформатированный текст в новом текстовом файле с именем Name2.

Вариант 18.

Дано целое число N и текстовый файл Name1, содержащий текст, выровненный по левому краю. Абзацы текста отделяются друг от друга одной пустой строкой. Отформатировать текст так, чтобы его ширина не

превосходила N позиций, и выравнивать текст по левому¹правому² краю, сохранив деление на абзацы. Пробелы в конце строк удалить. Сохранить отформатированный текст в новом текстовом файле Name2.

Вариант 19.

Дана строка K, состоящая из 10 цифр, и файл с русским текстом. Зашифровать файл, выполнив циклическую замену каждой русской буквы, стоящей на i-й позиции строки, на букву того же регистра, расположенную в алфавите на K[i]-м месте после шифруемой буквы (символы строки K также перебираются циклически: для i = 11 снова используется смещение K[1] и т.д.). Букву "ё" в алфавите не учитывать, знаки препинания и пробелы не изменять.

Вариант 20.

Дана строка S1 и файл с русским текстом, зашифрованным по правилу, описанному в задании Text39. Строка S1 представляет собой первую расшифрованную строку текста. Расшифровать остальные строки и заменить в файле зашифрованный текст на расшифрованный. Если информации для расшифровки недостаточно, то исходный файл не изменять.

Пример 2: Работа с символами. Ввести в файл символы с A до z , вывести из файла записанные данные.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            string fileStr = "lat.txt"; //имя файла для записи
            FileStream file; //файловая переменная для записи
            file = new FileStream(fileStr, FileMode.OpenOrCreate);
            StreamWriter fileW = new StreamWriter(file);

            for (char cc = 'A'; cc <= 'z'; cc++) // запись в файл lat.txt
            { fileW.Write(cc); Console.Write(cc); }
            fileW.Flush(); //очистка буфера памяти
            fileW.Close(); //файл file закрыт
        }
    }
}
```

```

file = new FileStream(fileStr, FileMode.Open);
StreamReader fileR = new StreamReader(file);

//файловая переменная для считывания
int k;
do
{
    k = fileR.Read(); //считывание из файла
    Console.Write((char)k + " " + k + " ");
} //ВЫВОД !!
while (k != -1); //считывание до конца файла
fileR.Close(); //файл file закрыт
Console.ReadKey();
}
}
}

```

На рисунке 2.7 представлено окно выполнения программы.

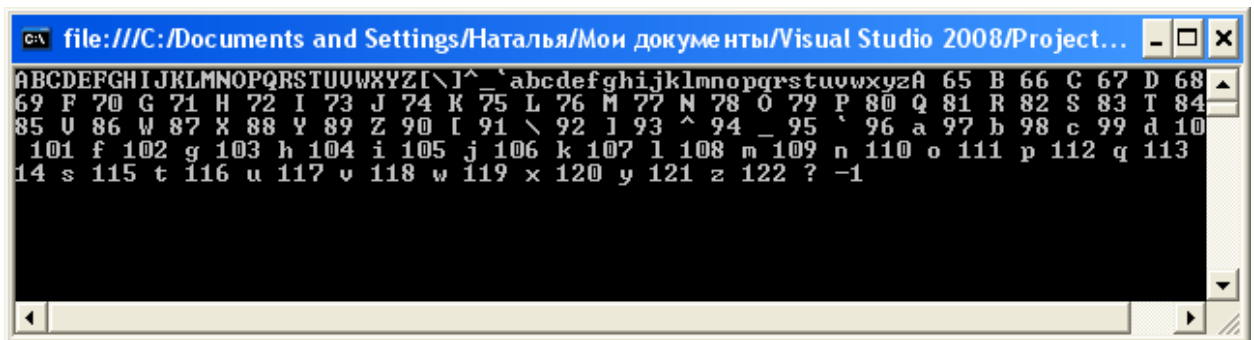


Рисунок 2.8 – Пример выполнения программы

Задание 2: Осуществить выполнение программы с файлами.

Вариант 1.

Даны два файла вещественных чисел с именами Name1 и Name2, элементы которых упорядочены по возрастанию1|убыванию2. Объединить эти файлы в новый файл с именем Name3, сохранив упорядоченность элементов.

Вариант 2.

Даны два целых числа i и j и файл вещественных чисел, содержащий элементы квадратной матрицы (по строкам). Вывести элемент матрицы, расположенный в i -й строке и j -м столбце (строки и столбцы нумеруются от 1). Если требуемый элемент отсутствует, то вывести 0.

Вариант 3.

Даны два целых числа i и j и файл вещественных чисел, содержащий элементы прямоугольной матрицы (по строкам), причем начальный элемент файла содержит количество столбцов матрицы. Вывести элемент матрицы, расположенный в i -й строке и j -м столбце (строки и столбцы нумеруются от 1). Если требуемый элемент отсутствует, то вывести 0.

Вариант 4.

Дан файл вещественных чисел, содержащий элементы квадратной матрицы (по строкам). Создать файл, содержащий элементы матрицы, транспонированной к исходной.

Вариант 5.

Дан файл вещественных чисел, содержащий элементы прямоугольной матрицы (по строкам), причем начальный элемент файла содержит количество столбцов матрицы. Создать новый файл той же структуры, содержащий матрицу, транспонированную к исходной.

Вариант 6.

Даны два файла вещественных чисел с именами NameA и NameB, содержащие элементы квадратных матриц A и B (по строкам). Создать новый файл с именем NameC, содержащий элементы произведения $A \cdot B$. Если матрицы A и B нельзя перемножать, то оставить файл NameC пустым.

Вариант 7.

Даны два файла вещественных чисел с именами NameA и NameB, содержащие элементы прямоугольных матриц A и B (по строкам), причем начальный элемент каждого файла содержит количество столбцов соответствующей матрицы. Создать файл той же структуры с именем NameC, содержащий произведение $A \cdot B$. Если матрицы A и B нельзя перемножать, то оставить файл NameC пустым.

Вариант 8.

Дан файл вещественных чисел, содержащий элементы [верхней треугольной]1|[нижней треугольной]2|трехдиагональной3 матрицы (по строкам). Создать новый файл, содержащий элементы ненулевой части данной матрицы (по строкам).

Вариант 9.

Даны два целых числа i и j и файл вещественных чисел, $qndepf\ yhi$ ненулевую часть [верхней треугольной]1 | [нижней треугольной]2| трехдиагональной3 матрицы (по строкам). Вывести порядок матрицы и ее элемент, расположенный в i -й строке и j -м столбце (строки и столбцы

нумеруются от 1). Если требуемый элемент находится в нулевой части матрицы, то вывести 0; если элемент отсутствует, то вывести -1 .

Вариант 10.

Дан файл вещественных чисел, содержащий ненулевую часть [верхней треугольной]¹[[нижней треугольной]²|трехдиагональной]³ матрицы (по строкам). Создать новый файл, содержащий все элементы данной матрицы (по строкам).

Вариант 11.

Даны два файла вещественных чисел с именами NameA и NameB, содержащие ненулевые части [верхних треугольных]¹[[нижних треугольных]² матриц A и B (по строкам). Создать новый файл с именем NameC, содержащий ненулевую часть произведения A·B исходных матриц (по строкам). Если матрицы A и B нельзя перемножать, то оставить файл NameC пустым.

Вариант 12.

Дано целое число N ($N < 5$) и N файлов целых чисел разного размера с именами Name1,..., NameN. Объединить их содержимое в новом файле целых чисел с именем Name0, используя следующий формат: в начальном элементе файла Name0 хранится число N , в следующих N элементах хранятся размеры исходных файлов, а затем последовательно размещаются данные из каждого исходного файла.

Вариант 13.

Дан файл целых чисел, содержащий данные из нескольких (не более четырех) файлов в формате, описанном в задании File41. Восстановить файлы, использованные при создании исходного файла, присвоив им имена вида ".tst", где — порядковый номер файла ($n = 1, 2, \dots$).

Вариант 14.

Дан символьный файл, содержащий по крайней мере один символ пробела. Удалить все его элементы, расположенные после первого¹|последнего² символа пробела, включая и сам этот пробел.

Вариант 15.

Дан символьный файл, содержащий по крайней мере один символ пробела. Удалить все его элементы, расположенные перед первым¹|последним² символом пробела, включая и сам этот пробел.

Вариант 16.

Дан символьный файл. Упорядочить его элементы по возрастанию¹|убыванию² их кодов.

Вариант 17.

Дано число k и строковый файл с именем `Name1`, содержащий непустые строки. Создать два новых файла: строковый с именем `Name2`, содержащий первые 1 |последние 2 k символов каждой строки исходного файла (если строка короче k символов, то она сохраняется целиком), и символьный с именем `Name3`, содержащий k -й символ каждой строки (если строка короче k , то в файл `Name3` записывается пробел).

Вариант 18.

Дан строковый файл, содержащий непустые строки. Создать новый файл, содержащий все строки исходного файла наименьшей 1 |наибольшей 2 длины (в том же порядке).

Вариант 19.

Дан строковый файл с именем `NameS`, содержащий даты в формате "день/месяц/год", причем под день и месяц отводится по две позиции, а под год — четыре. Создать файлы целых чисел с именами `Name1` и `Name2`, содержащие соответственно значения [дней и месяцев] 1 |[дней и лет] 2 |[месяцев и лет] 3 для дат из исходного строкового файла (в том же порядке).

Вариант 20.

Дан строковый файл, содержащий даты в формате "день/месяц/год", причем под день и месяц отводится по две позиции, а под год — четыре. Вывести строку, содержащую самую раннюю 1 |позднюю 2 весеннюю 3 |летнюю 4 |осеннюю 5 |зимнюю 6 дату. Если даты с требуемым временем года в файле отсутствуют, то вывести дату "01/01/1900".

Вариант 21.

Дан строковый файл, содержащий даты в формате "день/месяц/год", причем под день и месяц отводится по две позиции, а под год — четыре. Создать новый строковый файл, в котором даты из исходного файла располагались бы в порядке возрастания 1 |убывания 2 .

2.3 Обработка исключительных ситуаций

При разработке сложных программных систем практически невозможно избежать ошибок, которые проявляются не всегда (иначе они были бы обнаружены и устранены), но в большинстве случаев приводят к краху системы или к получению неверного результата. Причиной таких ошибок являются так называемые *исключительные ситуации*, или *исключения*. Они могут быть связаны с попыткой деления на ноль, открытием

несуществующего файла, выходом индекса массива из допустимых границ и т. п.

Средства обработки исключений

Как и в большинстве других современных языков программирования, в С# есть средства обработки исключений, то есть средства программной реакции на ошибочную ситуацию. Цель такой обработки — как минимум, известить пользователя о недостоверности получаемого результата, а как максимум — попытаться устранить последствия ошибки и получить достоверный результат.

Для реализации обработки исключительной ситуации в месте возможного обнаружения исключения создается так называемый защищенный блок. При возникновении исключения в защищенном блоке управление передается некоторому обработчику исключения, который может проинформировать пользователя о некорректной работе программы и (или) попытаться устранить последствия ошибки.

Синтаксис защищенного блока таков:

```
try
{ // Защищенный блок кода ...}
catch (T1 e1)
{ // Обработчик исключения. Срабатывает, если тип исключения T1
...}
...
catch(Tk ek)
{ // Обработчик исключения. Срабатывает, если тип исключения Tk
...}
finally
{ // Блок финализации. Срабатывает в любом случае
...}
```

Здесь **try**, **catch**, **finally** — зарезервированные слова.

В CTS есть специальный класс `Exception`, который вместе со своими многочисленными потомками участвует в обработке исключения: если возникло исключение, автоматически прерывается дальнейшее выполнение защищенного кода, создается объект класса `Exception` или одного из его потомков и в обработчиках **catch**() ищется первый по ходу программы блок, предназначенный для обработки исключения данного класса `Ti`. Блоку обработчика передается объект `ei` класса `Ti`, уточняющий обстоятельства возникновения ошибки. После срабатывания обработчика управление передается в блок финализации, в котором осуществляются некоторые общие действия, связанные с работой защищенного блока. В блок **finally** управление передается и в случае, если в защищенном блоке не возникла исключительная ситуация, то есть этот блок срабатывает в любом случае. В

нем, например, могут закрываться открытые файлы или освобождаются ресурсы, выделенные защищенному блоку.

Допускается произвольное количество (в том числе — ни одного) обработчиков **catch()**, разрешается также не использовать блок **finally**. Защищенные блоки могут быть вложенными.

При построении защищенного блока необходимо блокировать выдачу неверного результата, который может ввести пользователя в заблуждение и привести к непредсказуемым последствиям — в этом состоит главное назначение механизма обработки исключительных ситуаций.

Итак, при возникновении исключительной ситуации создается объект класса **Exception** или одного из его потомков. Класс **Exception** уведомляет пользователя о возникшей проблеме.

Наиболее важные свойства этого класса показаны в табл. 15.1.

Таблица 15.1. Свойства класса **Exception**

Свойство	Назначение
HyperLink	Возвращает URL файла справки с описанием ошибки
Message	Возвращает текстовое описание ошибки
Source	Возвращает имя объекта или приложения, сгенерировавшего исключение
StackTrace	Возвращает состояние стека на момент возникновения исключения
InnerException	Используется для сохранения сведений о текущем исключении между сериями исключений

Потомки **Exception** специализируются на обработке конкретных исключительных ситуаций. Например, в пространстве имен **System** определены такие важные классы, как **ArgumentOutOfRangeException**, **IndexOutOfRangeException**, **StackOverflowException** и многие другие.

Исключение возбуждает в подавляющем большинстве случаев ядро операционной системы, однако в **C#** это может сделать и сама программа.

Генерация исключения

Для генерации исключения используется зарезервированное слово **throw** (*throw* по-английски означает бросить, поэтому иногда вместо *генерации* исключения говорят о *вбрасывании* исключения).

Продемонстрируем генерацию и технику обработки исключения на примере класса **Book**, в котором хранится информация о книге. В примере конструктор класса проверяет год выпуска книги, и если год больше текущего, генерирует исключение.

Пример: Найти $k=18/kk$.

```
using System;  
using System.Collections.Generic;  
using System.Linq;
```

```

using System.Text;

namespace ConsoleApplication4
{
    class Program
    {
        static void Main(string[] args)
        {
            string str = "88g01";
            int k; double a = 18.0;
            int kk = 0; int i;
try
{   i=Convert.ToInt32(str); //несоответствие str и Int32
    k=18/kk;                //деление на ноль
} //конец блока try
catch (FormatException)
    {Console.WriteLine("Строка не отображает число,
допустимое для заданного типа");
    Console.ReadKey();return;}
catch
    {Console.WriteLine("Неопределенная ошибка");
    Console.ReadKey(); return; }
    //конец блока catch
Console.WriteLine(kk+" "+k+" "+i);

```

Пример: использование finally.

```

class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("Enter a number: ");

            var num = int.parse(Console.ReadLine());

            Console.WriteLine($"Squre of {num} is {num * num}");
        }
        catch(Exception ex)
        {
            Console.Write("Error info:" + ex.Message);
        }
        finally
        {
            Console.Write("Re-try with a different number.");
        }
    }
}

```

```
    }  
}
```

Пример: использование различных типов catch.

```
class Program  
{  
    static void Main(string[] args)  
    {  
        Console.Write("Please enter a number to divide 100: ");  
  
        try  
        {  
            int num = int.Parse(Console.ReadLine());  
  
            int result = 100 / num;  
  
            Console.WriteLine("100 / {0} = {1}", num, result);  
        }  
        catch(DivideByZeroException ex)  
        {  
            Console.Write("Cannot divide by zero. Please try again.");  
        }  
        catch(InvalidOperationException ex)  
        {  
            Console.Write("Invalid operation. Please try again.");  
        }  
        catch(FormatException ex)  
        {  
            Console.Write("Not a valid format. Please try again.");  
        }  
        catch(Exception ex)  
        {  
            Console.Write("Error occurred! Please try again.");  
        }  
    }  
}
```

В примере используется объект исключения класса Exception. Как уже говорилось, этот класс предназначен для любых исключений. В таблице 2.9 описаны некоторые классы исключений, определенные в пространстве имен System.

Таблица 2.9 - Некоторые классы исключений

Класс	Описание ошибки
AccessViolationException	Попытка доступа к защищенной памяти
ApplicationException	Обнаружение нефатальной ошибки
ArgumentException	Передача методу в одном из параметров вызова недопустимого значения

ArgumentNullException	Передача недопустимого значения null в качестве параметра вызова метода
ArgumentOutOfRangeException	Параметр вызова вышел из допустимых границ
ArithmeticException	Арифметическая ошибка или ошибка приведения типа
ArrayTypeMismatchException	Попытка поместить в элемент массива значение недопустимого типа
BadImageFormatException	Попытка воспроизвести изображение из файла с недопустимым форматом
DataMisalignedException	Попытка разместить блок данных в памяти недостаточного размера
DivideByZeroException	Попытка деления на ноль
FieldAccessException	Обращение к закрытому или защищенному полю объекта
FormatException	Обнаружение несоответствия параметров и спецификации форматирования
IndexOutOfRangeException	Выход индекса массива из допустимых границ
InsuffisientMemoryException	Исчерпание доступной памяти
InvalidCastException	Недопустимое приведение типов
InvalidProgramException	Обнаружение ошибки в тексте программы на промежуточном языке CIL (обычно из-за ошибки компилятора)
MemberAccessException	Ошибка доступа к члену класса
MissingFieldException	Попытка доступа к несуществующему полю
MissingMemberException	Попытка доступа к несуществующему члену класса
MissingMethodException	Попытка доступа к несуществующему методу класса
MulticastNotSupportedException	Попытка объединения двух несовместимых делегатов
NotFinityNumberException	Значение числа с плавающей точкой равно бесконечности или не является числом
NoySupportedException	Исполняемый метод не поддерживается или делается попытка чтения, записи или смещения в потоке, который не поддерживает эту функциональность
NullReferenceException	Попытка обращения к ссылочному типу по неправильной ссылке
OperationCancelledException	Ошибочное завершение текущего

	программного потока
OutOfMemoryException	Нехватка памяти
PlatformNotSupportedException	Попытка выполнить CIL-программу на машине, которая не поддерживает платформу .NET
RankException	Передача методу массива недопустимой размерности
StackOverflowException	Переполнение стека
TimeoutException	Исчерпано время, выделенное процессу

В других пространствах имен определены собственные классы исключений. Например, в пространстве System.IO определены такие классы, как DdriveNotFoundException, FileNotFoundException, InvalidDataException.

Если ни один из стандартных классов исключений не удовлетворяет нужным целям, программист может создать собственный класс, объявив для него родителем класс System.Exception.

Захват исключения

Как уже говорилось, при вбрасывании исключения нормальное исполнение защищенного блока прерывается и в списке обработчиков **catch** ищется первый обработчик, способный обработать исключение данного типа. Если в защищенном блоке могут генерироваться несколько исключений, обработчики **catch** нужно располагать в определенной последовательности: самыми первыми должны располагаться наиболее специализированные обработчики, а обработчики типа Exception — последними. Если, например, в защищенном блоке возможно деление на ноль или переполнение стека, обрабатывать эти ошибки нужно так:

```
try
{...}
catch(DivideByZeroException e){...} // Обработка деления на 0
catch(StackOverflowException e){...} // Обработка переполнения стека
catch(ArithmeticException e){...} // Обработка любой арифметической
// ошибки
catch(Exception e){...} // Обработка любой ошибки
finally {...}
```

Если изменить порядок следования на обратный, ни один из специализированных обработчиков никогда не получит управления.

Пример: Дано целое число kk . Найти значение выражения: $k=18/kk$.

```
using System;
using System.Collections.Generic;
```



```

using System.Linq;
using System.Text;

namespace ConsoleApplication4
{
    class Program
    {
        static void Main(string[] args)
        {
            string str = "88g01"; int k; double a = 18.0; int kk = 0; int i;
            try
            {
                i = Convert.ToInt32(str);    //несоответствие str и Int32
                k = 18 / kk;                  //деление на ноль
            } //конец блока try
            catch (FormatException)
            {
                Console.WriteLine("Строка не отображает число, допустимое для
                                   заданного типа");
                Console.ReadKey(); return;
            }
            catch
            {
                Console.WriteLine("Неопределенная ошибка");
                Console.ReadKey(); return;
            }
            //конец блока catch
            Console.WriteLine(kk + " " + k + " " + i);
        }
    }
}

```

Результат выполнения программы показан на рисунке 2.9.

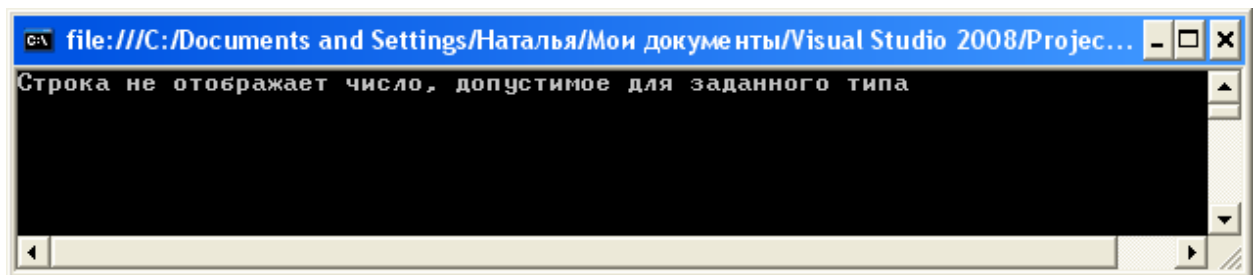


Рисунок 2.9 - Окно работающей программы

Задание: Вычислить значение Y, используя операторы для обработки исключительных ситуаций. Проверка на неверный ввод данных, подкоренное выражение не отрицательное и при делении на «ноль».

Вариант №1

$$Y = \frac{\sqrt{5+2X}}{|7-7X|} + (\sqrt[3]{6-X}) - \sin(4x)$$

Вариант №2

$$Y = \left| \frac{\sqrt{2X}}{-X} + \sqrt[5]{6+X} - \cos(4x) \right|$$

Вариант №3

$$Y = \frac{3+4X}{7^x - 7X} + (\sqrt[3]{5+X}) - (4x)$$

Вариант №4

$$Y = \frac{\sqrt{10X+2X^3}}{7-7X} + \sqrt[4]{|6+X|} - X$$

Вариант №5

$$Y = \left| \frac{\sqrt{5+2X}}{2-5X} + (\sqrt[3]{2+X}) - \sin(4x) \right|$$

Вариант №6

$$Y = \frac{\sqrt{5-X}}{-7X} + (\sqrt[3]{6+X}) - |\sin(x)|$$

Вариант №7

$$Y = \sin\left(\frac{\sqrt{5+2X}}{|7-7X|}\right) + (\sqrt[3]{6+X}) - (4x)$$

Вариант №8

$$Y = \frac{\sqrt{5+2X}}{|1-5X|} + (\sqrt[3]{6+X}) - \sin(4x)$$

Вариант №9

$$Y = \frac{\sqrt{|5+2X|}}{-3X} + (\sqrt[3]{6+X}) - \sin(4x)$$

Вариант №10

$$Y = \frac{\sqrt{2X-4}}{X} + \left(\sqrt[3]{\frac{X}{4} + X}\right) - \sin(4x)$$

Вариант №11

$$Y = \frac{\sqrt{5+2X}}{|7-7X|} + (\sqrt[3]{6+X}) - \cos(4x)$$

Вариант №12

$$Y = \cos(X) \left| \frac{\sqrt{5+2X}}{X} + (\sqrt[3]{6+X}) \right|$$

Вариант №13

$$Y = \frac{\sqrt{5+2X}}{|3-7X|} + (\sqrt[3]{6+X}) - \sin(4x)$$

Вариант №14

$$Y = \frac{\sqrt{2X}}{|-8X|} + (\sqrt[3]{6X^3+X}) - \sin(4-x)$$

Вариант №15

$$Y = \operatorname{tg} \left| \frac{\sqrt{5+2X}}{-3-X} \right| + (\sqrt[3]{X^4+X}) - \sin(x)$$

ПРИЛОЖЕНИЕ А

Задачник

1. Найти значение выражения $1*1+2*2+...+n*n$.
2. Грани куба можно раскрасить: а) все в белый цвет; б) все в чёрный; в) часть в белый цвет-часть в чёрный; Напечатать возможные варианты и их кол-во.
3. Составить из двух таблиц 3-ю упорядоченную по возрастанию.
4. Найти максимальное число из трёх.
5. Найти максимальное число из четырёх.
6. Кол-во букв "а" в тексте.
7. Среднее арифметическое таблицы.
8. Степень числа.
9. Факториал числа.
10. Подсчет кол-ва часов, минут и секунд в данном числе суток.
11. Составить программу проверки есть ли в тексте буква "s".
12. Сколько различных ожерелий можно составить из 2-ух белых, 2-ух синих и 2-ух красных бусин. Напечатать возможные варианты и их кол-во.
13. Вывести на печать трехзначные числа, которые делятся на свои цифры и перевертыш этого числа тоже делится на свои цифры.
14. Определить лежит ли точка а на прямой $y=kx+l$.
15. Расположить слова в порядке убывания их длины в предложении.
16. Найти кол-во отрицательных элементов таблицы.
17. Найти максимальный элемент таблицы $a[1..10]$.
18. Получить элементы таблицы, которые находятся между \max и \min .
19. Яв-ся ли треугольник равнобедренным.
20. Лежит ли точка на прямой.
21. Проверить существует ли строгое чередование.
22. Пересекаются ли отрезки.
23. Является ли n-угольник выпуклым.
24. Определить расстояния от точки до прямой.
25. Найти площадь треугольника (используя формулу Герона).
26. Даны координаты диагонали прямоугольника. Найти его площадь.
27. Найти номер максимального элемента таблицы $a[1..10]$.
28. Составить программу упорядочивания элементов таблицы.
29. Составить программу вычисления $(\min(a,c)-\min(a,b))/(5+\min(b,c))$
30. Является ли число b делителем числа a.
31. Составить программу определяющую является ли число простым.
32. Составить программу нахождения НОД и НОК двух чисел a и b.
33. Составить программу решения квадратного уравнения.
34. Найти сумму элементов прямоугольной таблицы размером $[n,m]$
35. Найти максимальный элемент прямоугольной таблицы размером $[n,m]$.
36. Напечатать словарь состоящий из четырёх букв неповторяющихся в слове.

37. Найти максимальный элемент таблицы и их кол-во.
38. Дано предложение, определить количество слов в нём.
39. Дан текст, определить количество слов "кот".
40. Определить является ли данное слово перевертышем.
41. Найти количество различных чисел в одномерной таблице.
42. Каждую букву слова А поместить в таблицу.
43. Найти наименьшее однозначное число x удовлетворяющее условию $x * x * x - x * x = n$.
44. Составить алгоритм нахождения суммы цифр числа.
45. Найти двузначное число сумма кубов цифр которого равна n .
46. Получить из слова a , вычеркивание некоторого количества букв, слово b .
47. Заданы 2 точки. Определить какой из отрезков АО или ВО образует больший угол с осью ОХ.
48. Записать положительные элементы таблицы А в таблицу В, а отрицательные элементы таблицы А в табл С.
49. Является ли перевёртышем число.
50. Построить таблицу С в которой сначала размещаются все элементы А, затем все элементы таблицы В.
51. Решить систему уравнений $ax + by + c = 0$ и $alx + bly + cl = 0$.
52. Определить площадь и периметр треугольника.
53. Дана таблица содержащая группы одинаковых подряд идущих чисел. Вывести на экран "число - кол-во чисел в группе, число - кол-во чисел в группе, ... "
54. Определить площадь четырёхугольника.
55. Разбить выпуклый n -угольник на треугольники диагоналями так, чтобы...
56. Определить стоимость телеграммы (цена слова k).
57. Дана таблица $a[1..n]$. Ввести таблицу $b[1..n]$ отбросив из a каждый второй элемент.
58. Дана таблица $a[1..n]$ из целых чисел. Поставить сначала четные, а потом нечетные элементы.
59. Найти наибольшее кол-во одинаковых элементов.
60. Дана точка. Лежит ли она в кольце.
61. Примеры типов величин.
62. Табличные величины. Одномерный массив.
63. Табличные величины. Двумерный массив.
64. На оси Ох заданы N точек с координатами x_1, x_2, \dots, x_n . Найти такую точку Z , сумма расстояния от которой до данных точек минимальная.
65. Имеется n банок с целочисленными объёмами $v_1, v_2, v_3, \dots, v_n$ литров, пустой сосуд и кран с водой. Можно ли с помощью этих банок налить в сосуд ровно v литров воды. Решение: Обозначим $s = \text{nod}(v_1, v_2, \dots, v_n)$. Если v делится нацело на s , то в сосуд с помощью банок можно налить v литров воды, иначе - нет.

66. Дана последовательность натуральных чисел. Найти наименьшее натуральное число, которое отсутствует в последовательности.

67. Дан выпуклый n -угольник и точка (x_1, y_1) . Определить: а) является ли точка вершиной; б) принадлежит ли точка n -угольнику.

68. Используя вспомогательную функцию нахождения $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$ процесс суммирования остановить если очередной член станет меньше 0.001.

69. Используя вспомогательную функцию нахождения $\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$ процесс суммирования остановить если очередной член станет меньше 0.001.

70. Дано предложение. Сколько слов яв-ся перевёртышами и будет ли это число совершенным.

71. Дано предложение заканчивающееся '!', '!', '?'. Разделитель слов - пробел. Определить будет ли число простых множителей числа S - кол-ва букв "т", больше заданного числа L .

72. Дано предложение заканчивающееся '!', '!', '?'. Разделитель слов - пробел. В скольких словах предложения имеется словосочетание "ка".

73. Дана целочисленная таблица $a[1..m]$. Среди её элементов есть хотя бы один отрицательный. Больше ли сумма сумм простых множителей элементов идущих после последнего отрицательного элемента заданного числа L .

74. Дана целочисленная таблица $a[1..m]$. Среди элементов таблицы есть хотя бы один отрицательный. Найти сумму S элементов расположенных после отрицательного элемента, затем найти сумму простых множит. числа S .

75. Слова в предложении разделены пробелом. Предложение заканчивается "." "!" "?" Определить слово с максимальным числом букв "а" и количество таких букв "а".

76. Даны вершины треугольника. Определить можно ли разместить этот треугольник в круге радиуса r .

77. Дано натуральное число. Представьте его в виде суммы степеней двойки. Кол-во слагаемых k . Будет ли удвоенная сумма простых множителей числа k больше самого k

$$201 = 128 + 64 + 8 + 1 = 2^7 + 2^6 + 2^3 + 2^0.$$

т.е $k=4$. Простой множитель k : 2; $2 \cdot 2 < 4 < k$

78. Дано предложение. Сколько слов яв-ся перевёртышами и сколько букв "а". Найти их разность.

79. Дана вещественная таблица $a[1..50]$ Найти среднее арифметическое положительных элементов таблицы и минимум абсолютного значения элементов. Найти их произведение.

80. Дана целочисленная таблица $a[1..20]$ из положительных элементов. Найти среднее арифметическое элементов таблицы и выяснить является ли данное натуральное число совершенным (натур число называется

совершенным если оно равно сумме своих делителей, исключая само число, например $6=1+2+3$)

81. Дано предложение заканчивающееся точкой. Из слов предложения вычеркивается буква а. Определить сколько слов в новом предложении яв-ся перевертышами.

82. Дано слово. Найти сколько раз буква "а" встречается в этом слове. Будет ли это число простым.

83. Дано предложение. Найти в каком из слов, больше четырёх символов, буква "а" встречается реже.

84. Дано предложение заканчивающееся .,!,?. Разделитель слов - пробел. Определить, сколько слов в предложении является перевёртышами и будет ли это число простым.

85. Дан текст. Установить пробелы вместо символов, номера позиций которых при делении на 4 дают в остатке 3.

86. Дан текст. Удалить в нём все слова "функция".

87. Дано предложение. Расположить слова в нём в порядке возрастания числа букв в словах.

88. Заменить данную букву в слове многоточием.

89. Даны слово и буква. Сколько раз эта буква встречается в данном слове.

90. Зашифровать слово, поставив букве её номер в алфавите ("ё" не учитывать)

91. Дано предложение. Определить все слова которые начинаются с заданной буквы. Слова в предложении разделены пробелами.

92. По номеру месяца определить его название и время года к которому он относится.

93. Дан текст. Определить все слова оканчивающиеся на "ая".

94. Дан текст. Сколько в нём слов "что".

95. Дано предложение. Определить кол-во слов в нём.

96. Заполнить элементами таблицу, располагая их по спирали.

97. Определить сколькими различными способами можно подняться на десятую ступеньку, если за шаг можно подняться следующую или через одну.

98. Фишка может двигаться по полю длиной n только вперёд. Длина хода фишки не более k. Найти число различных путей, по которым фишка может пройти поле от позиции 1 до позиции n. ПРИМЕР: $n=4, k=2$

Ответ: 1,1,1

1,2

2,1

99. В выражении $((((1?2)?3)?4)?5)?6$ вместо каждого знака "?" вставить знак одной из четырех операций ("+", "-", "*", ".") так, чтобы результат вычислений равнялся X (при делении дробная часть отбрасывается). Найти все варианты.

100. Найти кол-во n -значных чисел в десятичной системе счисления, у каждого из которых сумма цифр равна k . При этом в качестве n -значного числа мы допускаем и числа, начинающиеся с одного или нескольких нулей. Например, число 000102 рассматривается как шестизначное, сумма цифр которого равна 3.

101. Составить алгоритм определения кол-ва $2N$ -значных "счастливых" билетов, у которых сумма первых N цифр равна сумме последних N цифр; N - произвольное натуральное число.

102. Ввести строку длиной до 30 символов, заменить в ней двойных символов на одиночные, пробелов - на знак подчёркивания, сочетания '*' на многоточие '...'.

103. Ввести массив из 10 положительных чисел. Определить три стоящих подряд числа, сумма которых максимальна. Вывести эту сумму, а числа заменить нулями.

104. Дано целое число $N < 20$. Составьте программу, которая определяет кол-во различных делителей числа $N!$.

105. Посчитать слова (слова разделены одним или несколькими пробелами) в текстовом файле и добавить информацию об этом (например: 'В этом файле .. слов') в конец данного файла.

106. Ввести матрицу целых чисел. Найти и вывести пару элементов матрицы, модуль разности которых минимален.

107. Дана строка текста, состоящая из слов разделенных одним из знаков [#,\$,*, -]. Если кол-во слов в предложении четно, поменяйте местами два центральных слова, а если нечетно удалите одно центральное слово.

108. Имеется ожерелье, которое состоит из k ($k \leq 20$) бусинок(з), желтого(ж) и красного(к) цветов. Найти максимальное кол-во бусинок одного цвета, идущих подряд.

109. На натуральном отрезке $[a, b]$ найдите и выведите число N с наибольшей суммой своих делителей. Само число и единицу в качестве делителей не учитывать.

110. Данные контрольной работы учащихся по информатике представлены следующим образом:

"отлично"	- кол-во учащихся a
"хорошо"	- кол-во учащихся b
"удовлетворительно"	- кол-во учащихся c
"неудовлетворительно"	- кол-во учащихся d .

Постройте или столбчатую гистограмму с легендой, которая отражает результаты контрольной работы.

111. Результаты таблицы выигрышей денежной лотереи представлены последовательностью натуральных чисел, записанных в текстовом файле в несколько строк через пробел. Три первые цифры каждого числа - номер билета, а последние три цифры величина выигрыша. Определите и выведите номера билетов с наибольшим выигрышем. Например,

Входные данные:

10245857 1254387 132563
6377739 4237857

Выходные данные:

102 -857

423 -857.

112. Экономия в строительстве дорог при строительстве ж/д. станции.

113. Строительство ж/д. станции по принципу справедливости.

114. Фишка может двигаться по полю длиной n только вперёд. Длина хода фишки не более k . Найти число различных путей, по которым фишка может пройти поле от позиции 1 до позиции n

ПРИМЕР: $n=4, k=2$

Ответ: 1,1,1

1,2

2,1

115. Задаётся словарь. Найти в нём все анаграммы(слова составленные из одних и тех же букв).

116. Найти числа x, y, z , удовлетворяющие условию $ax+by+cz=n$ (пусть $n=270$ $a=15$, $b=20, c=30$ то $15x+20y+30z=270$). Решение: если $x=0$ и $y=0$, то $30z=270$ т.е. $z \leq 9$ аналогично находим ,что $y \leq 14, x \leq 18$.

117. Треугольник ABC задан координатами и точка $D(x_4, y_4)$. Лежит ли точка D внутри ABC. МЕТОД-точка внутри если сумма площадей 3-х треугольников равна площади треугольника ABC.

118. В таблице a заменить отрицательные элементы на 0.

119. Дана таблица из n строк и n столбцов. Найти суммы элементов записанных по диагоналям.

120. Дана таблица $a(n:m)$ Умножить каждый элм первой строки на $a[1,1]$ (в том числе и элемент $a[1,1]$) а каждый элемент второй строки на $a[2,2]$ и т.д.

121. Дана линейная таблица a . Найти максимальный элемент таблицы и найти его среди элементов таблицы b .

122. Даны n -троек a, b, c . Можно ли построить треугольник с данными сторонами.

123. Напечатать в возрастающем порядке все трёхзначные числа, в десятичной записи которых нет одинаковых цифр.

124. Являются ли числа a, b, c (≤ 100) пифагоровыми тройками.

125. Составить программу определения суммы цифр числа a .

126. Дан выпуклый n -угольник и точка (x_1, y_1) . Определить: а) является ли точка вершиной; б) принадлежит ли точка n -угольнику.

127. Даны координаты 2-х точек. Найти точку на оси X чтобы сумма расстояний до данных было минимальной.

128. Дано натуральное число n . Верно ли, что сумма цифр этого числа яв-ся нечётной.

129. Натуральное число из n цифр является числом Армстронга, т.е. сумма его цифр, возведенная в n степень, равна самому числу ($153=1*1*1+5*5*5+3*3*3$). Получить все числа Армстронга для $n=4$ и $n=3$.

130. Посчитать сумму цифр всех целых чисел 1 до n .

131. Дано число n . Верно ли, что это число содержит ровно 3 одинаковых цифры.

132. Имеется n бактерий красного цвета. Через 1 такт времени красная бактерия меняется на зелёную, затем через 1 такт времени делится на красную и зелёную. Сколько будет всех бактерий через k тактов времени?

133. Дано число n . Выбросить из него все единицы и пятёрки, оставив порядок цифр

ПРИМЕР: 527012 преобразуется в 2702

134. Дано натуральное число n . Выбросить из записи числа все чётные цифры.

135. Найти все числа палиндромы в диапазоне от n до m которые при возведении в квадрат так же дают палиндром.

136. Перевести число из десятичной в двоичную систему счисления.

137. Перевести число из двоичной в десятичную систему счисления.

138. Дана таблица $a[m,n]$ содержащая числа 0,1,5 или 11. Посчитать кол-во четвёрок $a[i,j]$, $a[i+1,j]$, $a[i,j+1]$, $a[i+1,j+1]$ в каждой из которых все элементы разные.

139. Сократима ли дробь a/b . Дробь a/b несократима, если НОД=1.

140. Вывести в порядке возрастания все несократимые дроби, заключённые между 0 и 1.

141. Дано предложение составить программу располагающую слова в порядке убывания длины слов.

142. Дано натуральное число A . Составить программу определения такого наибольшего N , что $N! < A$ ($A > 1$)

143. Составить программу для определения пройдёт ли кирпич с рёбрами a, b, c в прямоугольное отверстие со сторонами x, y .

144. Зашифровать слово, поставив букве её номер в алфавите.

145. Расшифровать слово, поставив соответствующей цифре букву.

146. Можно ли данное натуральное число представить в виде суммы двух квадратов чисел.

147. Расположить по краям таблицы нули.

148. (1)Получить n четырёхзначных чисел, в записи которых нет двух одинаковых цифр.

149. (2)Получить n 4-знач чисел, в записи которых нет двух одинаковых цифр.

150. Тройку чисел (a, b, c) назовём Героновой тройкой, если эти числа натуральные и площадь треугольника тоже натуральное число. Вывести n Героновых троек.

151. ПРИМЕР :

Шаг0: Пустая последовательность

Шаг1: a

Шаг2: baa

Шаг3: cbaabaa

Составить программу определения заданному числу n символ на n -ом месте.

152. По заданным координатам клетки выдать координаты клеток имеющих с ней общую сторону.

153. Ввести натуральные числа n и m , и напечатать период десятичной дроби m/n , если дробь конечна, то период=0.

154. Составить программу дешифровки сообщения, закодированному по принципу. Например:

Шифр 432513 шифруем следующим образом:

НАСТОЯЩИЙ

432513432

СГУЧПВЭЛЛ

155. Дан текст. Можно ли из данных букв составить два слова.

156. Найти минимальное число, которое представляется суммой четырёх квадратов натуральных чисел не единственным образом.

157. Даны две последовательности x и y . Найти последовательность z , которую можно

получить вычёркиванием элементов как из x , так и из y .

158. Ввод '352', вывод - 'три пять два'.

159. Дан одномерный массив. Упорядочить массив удалив нули со сдвигом влево ненулевых элементов.

160. Дан текст. Отбросить повторяющиеся слова. Вывести повторяющиеся слова и их кол-во.

161. Вычислить в какой координатной четверти расположен треугольник образованный осями координат и прямой $y=kx+b$.

162. Вводится текст из файла INPUT.txt. Записать в файл с именем OUTPUT.txt слова в записи которых нет одинаковых букв

163. Вводится слово из файла INPUT.txt. Удалить из слова символы так, чтобы получить палиндром. Ответ записать в файл OUTPUT.txt.

164. Имеется n -вагонов стоящих в произвольном порядке и m -путей. Необходимо отсортировать вагоны по порядку т.е. 123456789... n .

165. В послед $a_1, a_2, a_3, \dots, a_n$ каждый член, начиная с четвёртого, равен последней цифре суммы трёх предыдущих. Найти n -ый элемент последовательности.

ПРИЛОЖЕНИЕ Б

Тесты

1 Оператор ввода:

- A) WriteLn
- B) Read
- C) ReadLn
- D) Write
- E) Array

2 Оператор вывода:

- A) WriteLn
- B) Read
- C) ReadLn
- D) Write
- E) Array

3 Оператор конца программы:

- A) End
- B){
- C) Вариант
- D) END
- E) }

4 // используется для :

- A) написание комментариев
- B) вычисления выражений
- C) ввода переменных
- D) вывода результатов
- E) ввода на печать

5 Оператор != используется для:

- A) равно
- B) не равно
- C) not
- D) присваивания
- E) ввода на печать

6 Оператор = используется для:

- A) написание комментариев
- B) вычисления выражений
- C) ввода переменных
- D) вывода результатов
- E) оператор присваивания

7 Для написания новой программы выполняется команда:

- A) FILE –RUN PROGRAM
- B) FILE –LIST PROGRAM
- C) FILE – NEW- PROJECT
- D) FILE –STOP PROGRAM
- E) FILE-QUIT

8 Оператор выбора :

- A) Switch
- B) switch
- C) Case
- D) SWITCH
- E) CASE

9 Оператор безусловного перехода:

- A) IF – THEN - ELSE
- B) FOR -TO-DO
- C) GOTO
- D) Writeln
- E) Readln

10 Оператор цикла с параметрами:

- A) do ... while
- B) for
- C) while
- D) repeat ... until
- E) while do

11 Оператор объявления одномерного массива:

- A) `int[] a = new int[n,m];`
- B) `int[,] a = new int[n];`
- C) `const int n = 6;`
`int[] a = new int[n];`
- D) `int[] a = new int[6];`
- E) `int[,] a = new int[n];`

12 Оператор `for (int i = 0; i < n; ++i)-`

- A) безусловный переход
- B) условный переход
- C) цикл с параметрами
- D) цикл с предусловием
- E) цикл с постусловием

13 Каким знаком обозначаются символьные переменные

- A) ' ' (апостроф)
- B) \$
- C) “” (кавычки)
- D) %
- E) ^

14 Оператор do – while выполняет функцию:

- A) безусловного перехода
- B) условного перехода
- C) цикл с параметрами
- D) цикл с предусловием
- E) цикл с постусловием

15 Оператор if –else выполняет функцию:

- A) безусловного перехода
- B) условного перехода
- C) цикл с параметрами
- D) цикл с предусловием
- E) цикл с постусловием

16 Оператор GOTO выполняет функцию:

- A) безусловного перехода
- B) условного перехода
- C) возврат с подпрограммы
- D) оператор цикла
- E) оператор ввода

17 Можно ли писать оператор } в конце подпрограммы

- A) обязательно
- B) нельзя
- C) без разницы
- D) обязательно
- E) в конце оператора

18 После выполнения оператора найти значение переменной S.

```
S=0; i=0;  
while (i<5) ++i;  
S+=1/i;  
A) 0,25  
B) 21/12  
C) 0,2  
D) 2,1
```

Е) 217/60

19 К какому циклическому оператору относится while :

- А) цикл - «до»
- В) ветвление
- С) повторение
- Д) цикл —«пока»
- Е) условие

20 К какому циклическому оператору относится for:

- А) ветвление
- В) цикл — «до»
- С) цикл — “с параметрами ”
- Д) цикл — «пока»
- Е) условий

21 К какому циклическому оператору относится do while:

- А) Цикл — “до”
- В) цикл — “FOR”
- С) цикл — “пока”
- Д) ветвление
- Е) условие

22 if else операторы:

- А) ветвление
- В) цикл — “до”
- С) цикл — “FOR”
- Д) цикл - “пока”
- Е) ввода

23 \sqrt{x} на языке C# пишется:

- А) Math.Sqrt(2,x);
- В) Math.Sqrt(x);
- С) Math.Sqrt(x,2);
- Д) Math.Sqr(x);
- Е) Math.Pow(x,2);

24 Как описывается двумерный массив:

- А) int [5,5] a=new int[5,5];
- В) int [,] a=new int[5,5];
- С) int [5,5] a=new int[,];
- Д) int [] a=new int[5,5];
- Е) new int [5,5] a= int[5,5];

25 Как описывается одномерный массив из целых чисел:

- A) `int [] a=new int[5];`
- B) `int [,] a=new int[5];`
- C) `int [5] a=new int[5];`
- D) `int [5] a=new int[];`
- E) `new int [] a=int[5];`

26 `||`, `&&` в операторе `if` означает:

- A) `||` – если выполняется одно из условий, то выполняется оператор после условия; `&&` – если выполняются оба условия, то выполняется оператор после условия
- B) `&&` – если выполняется одно из условий, то выполняется оператор после условия; `||` – если выполняются оба условия, то выполняется оператор после условия
- C) `||` – если выполняется оба условия, то выполняется оператор после условия; `&&` – если оба условия не выполняются, то выполняется оператор после условия
- D) `||` – пишется одно условие, `&&` – обязательно пишется две условия
- E) `&&`- пишется одно условие, `||`- обязательно пишется два условия

27 В конце подпрограммы обязательно пишется оператор:

- A) `end`
- B) `}`
- C) `END`
- D) `End`
- E) `;`

28 С использованием условного оператора, найти значение функции.

$$y = \begin{cases} \cos^2 x & \text{если } 0 < x < 2, \\ 1 - \sin x^2 & \end{cases}$$

- A) `if (x>0 && x<2) y=Math.Sqrt(cos(x));`
`else y=1-Math.Sin(sqr(x))`
- B) `if (x>0)&& (x<2) y=Math.Pow(Math.Cos(x),2)`
`else y=1-Math.Sin(Math.Pow(x,2));`
- C) `if (x>0)&&(x<2) then y:=sqr(cos(x))`
`else y:=1-sin(sqr(x))`
- D) `if (x>0 && x<2) y=Math.Sqrt(cos(x));`
`else y=1-Sin(sqr(x))`
- E) `if (x>0)&& (x<2) y:=Math.Pow(Math.Cos(x),2)`
`else y:=1-Math.Sin(Math.Pow(x,2));`

29 Найдите правильную запись выражения $\frac{|x+1| - \sqrt{x} - 1}{2x}$ на языке C#:

- A) `abs(x+1)-sqr(x)-1/2*x`
- B) `(Abs(x+1)-Math.Sqrt(x) -1)/2*x`
- C) `(abs(x+1)-sqr(x-1))/2*x`
- D) `(Math.Abs(x+1)- Sqrt(x) -1)/2-x`
- E) `(Math.Abs(x+1)-Math.Sqrt(x) -1)/2*x`

30 Написать на языке C# оператора присваивания $y = x^5 + \cos(x^2 + 1)$

- A) `y=Math.Pow(x,5)+Math.Cos(x*x+1)`
- B) `y:= Math.Pow(x,5)+Math.Cos(x*x+1)`
- C) `y:=sqr(sqr(x)*x+cos*sqr(x)+1`
- D) `y:= Pow(x,5)+ Cos(x*x+1)`
- E) `y=Math.Pow(5,x)+Math.Cos(sqr(x)+1)`

31 Укажите действительное число X:

- A) `X: float;`
- B) `X: double;`
- C) `X: int;`
- D) `double X;`
- E) `int X;`

32 Укажите правильное описание матрицы $\hat{A}(4,4)$ целых чисел:

- A) `const int n = 4;`
`int[,] a = new int[n];`
- B) `int[,] a = new int[4];`
- C) `const int n = 4;`
`int[,] a = new int[n, n];`
- D) `int[] a = new int[n, n];`
- E) `const int n = 4;`
`int[] a = new int[n, n];`

33 Даны переменные $A=14$, $B=12,3$ Укажите их правильное описание:

- A) `char A=14; double B=12.3;`
- B) `int A=14; int B=12.3;`
- C) `int a=14; double b=12.3;`
- D) `int A=14; double B=12.3;`
- E) `int A=14; double B=12,3;`

34 Как описывается действительный тип:

- A) `integer`
- B) `double`
- C) `int`

- D) char
- E) string

35 Как описывается целый тип:

- A) integer
- B) double
- C) int
- D) char
- E) string

36 Как описывается одномерный массив целых чисел:

- A) `int[] a = new int[4,4];`
- B) `int[,] a = new int[n];`
- C) `const int n = 6;`
`int[] a = new int[n];`
- D) `int[] a = new int[6];`
- E) `int[,] a = new int[n];`

37 Как описывается символьный тип:

- A) char
- B) byte
- C) real
- D) string
- E) bool

38 Как описывается одномерный массив из действительных чисел:

- A) `int[] a = new int[4,4];`
- B) `int[,] a = new int[n];`
- C) `const int n = 6;`
`double[] a = new double[n];`
- D) `int[] a = new int[6];`
- E) `int[,] a = new int[n];`

39 $c=a+b$ Если a, b – целый тип , тогда с какой тип:

- A) int
- B) float
- C) bool
- D) double
- E) real

40 $b=3/a$. Если a -целый тип ($a \neq 3, a \neq 1$). Тогда b какой тип?

- A) int
- B) byte
- C) bool

- D) double
- E) real

41 После выполнения оператора найти значение переменной D.

```
D=0; I=1;  
while (I<25)  
I+=2;  
D=D+1/(2*I+1);
```

- A) 0.02
- B) 1/47
- C) 1/51
- D) 1/49
- E) 0.03

42 Какому типу цикла относится следующая запись

```
for (<переменная>=< начальное значение; <условие>; <итерация> )  
<оператор>
```

- A) цикл с предусловием
- B) цикл с постусловием
- C) цикл до
- D) цикл с параметрами
- E) цикл пока

43 Укажите какому оператору относится следующая запись?

```
<переменная>= <выражение>;
```

- A) оператор перехода
- B) присваивания значения
- C) условный оператор
- D) оператор выбора
- E) оператор цикла

44 Укажите запись к какому оператору относится следующий формат?

```
if <логическое выражение> <оператор> [ELSE <оператор>;]
```

- A) оператор перехода
- B) оператор присваивания
- C) условный оператор
- D) оператор выбора
- E) оператор цикла

45 Укажите запись к какому оператору относится следующая запись

```
while <логическое выражение> { <оператор> }
```

- A) оператор перехода
- B) оператор присваивания
- C) оператор цикла с предусловием

- D) оператор выбора
- E) оператор цикла с параметром

46 Оператор switch

- A) оператор перехода
- B) оператор присваивания
- C) условный оператор
- D) оператор выбора
- E) оператор цикла

47 Укажите результат

- 1) Truncate (5.61) 2) Truncate (-5.61)
- A) 1) 5 и 2) - 6
 - B) 1) 6 и 2) - 5
 - C) 1) 6 и 2) -6
 - D) 1) 5 и 2) – 5
 - E) 1) 4 и 2) 6

48 Укажите результат

1. Math.Round (17.96) 2. Math.Round (-17.16)
- A) 17 и -17
 - B) 17 и -18
 - C) 18 и -18
 - D) 18 и -17
 - E) 4 и 6

49 Найти правильное из следующих записей

1. А значение (0,3) промежутке не подлежит:
(A<= 0) || (A>= 3)
1. А значение (-2,0) промежутке подлежит:
(A>=-2) && (A>=0)
- A) 1
 - B) 1 и 2
 - C) 2
 - D) 1 и -2
 - E) 1 3

50 Выберите правильный вариант из следующих операторов

- A) switch(x)
{ case 1: Console.Write("Рабочий день");
 case 2: Console.Write("Выходной день");
default: Console.Write("Не верный ввод данных"); }
- B) switch
{ case 1: Console.Write("Рабочий день"); break;

```

    case 2: Console.Write("Выходной день"); break; }
C) switch(x)
{ case1: Console.Write("Рабочий день"); break;
  case2: Console.Write("Выходной день"); break;
  default: Console.Write("Не верный ввод данных"); break; }
D) case(x)
{ switch 1: Console.Write("Рабочий день"); break;
  switch 2: Console.Write("Выходной день"); break;
  default: Console.Write("Не верный ввод данных"); break; }
E) switch(x)
{ case 1: Console.Write("Рабочий день"); break;
  case 2: Console.Write("Выходной день"); break; }

```

51 После выполнения оператора цикла найти значение переменной S

```

int s,i ; char sim;
1) s =0; for ( i=5; i<7; ++i) ++s;
2) s =0; for (i=10; i>6; --i) ++s;
3) s = 0; for (sim = 'A'; sim < 'D'; ++sim) ++s;
A) 1) 6      2) 5   3) 4
B) 1) 3      2) 4   3) 4
C) 1) 2      2) 4   3) 3
D) 1) 2      2) 5   3) 3
E) 1) 4      2) 2   3) 4

```

52 Укажите оператор ввода переменной “b” : .

```

A) b=Console.Read ();
B) Console.WriteLine (b);
C) Console.WriteLine (b);
D) b=Console.ReadLine ();
E) Console.WriteLine (b);

```

53 Найти неправильные условные операторы

```

1. if (a < b) a = a*a; else b=b * b;
2. if (x and y) s=s+1; else s = s - 1 ;
3. if (k != m) k = m;
4. if 5 then s = s+5;
1. if ( a == b ) && p then p= p+10.5
A) 3,4,5
B) 1,2 и 3
C) 4 и 5
D) 2, 4 и 5
E) 4,2 и 3

```

54 Какой из операторов удовлетворяет следующую запись

```
int x,y;  
A) x=3.0 ; y=4;  
B) x='3'; y='4';  
C) x= not y;  
D) x= y+ Math.Pow(3,2);  
E) x=y % 3;
```

55 Укажите правильную запись, после выполнения оператора цикла
int k=0;

```
while (k<=8)  
{ k+=2;  
Console.Write (k);}  
A) 10 8 6 4 2  
B) 2 4 6 8  
C) 2 4 6 8 10  
D) 10 8 6 4 2  
E) 10 5 4 8 2
```

56 Укажите правильный результат:

```
double x=0.1,y=-0.2,mm;  
if (x > y) mm=y; else mm = x;  
Console.Write ( mm);  
A) 0.2  
B) 0.1,-0.2  
C) -0.1  
D) 0.1  
E) -0.2
```

Правильный ответ= E

57 Укажите правильный результат

```
int n=1000;  
Console.Write ( n / 10 );  
A) 100  
B) 10.00  
C) 10  
D) 0  
E) 1000
```

58 Укажите правильный результат

```
char x='c';  
Console.Write ('x');  
A) c
```

- B) C
- C) 'x'
- D) x
- E) X

59 Укажите правильный результат

```
string s= "bcbcccckk"; int mI,k=0,k1=0;
for( mI=1;mI<8; ++mI)
{
if (s[mI]=='c') ++k;
if (s[mI] =='b') ++k1; }
Console.Write (k+ " "+k1);
```

- A) 1 и 3
- B) 2 и 2
- C) 4 и 1
- D) 10 и 8
- E) 2 и 4

60 Укажите правильный результат

```
double x=2.0,y=4.0,z=-0.1;
if ((x >= y) && (y >= z)) x = 2 * x; else y = Math.Abs(y);
Console.Write ("x="+x+"y="+y+"z="+z);
```

- A) x=2, y=4, z=-0.1
- B) x=2, y=8, z=-0.1
- C) x=4, y=16, z=0.2
- D) x=2, y=16, z=-0.4
- E) x=5, y=6,z=7

61 Укажите правильный результат

```
int n= 1000; Console.WriteLine ( n % 23 );
```

- A) 2
- B) 11
- C) 23
- D) 100
- E) 10.0

62 Найти неправильно записанное вещественное число

- 1) 1,81 2) 2.600 3) 0.56 4) 7,77
- A) 1 и 4
 - B) 1 и 2
 - C) 2 и 4
 - D) 2 и 4
 - E) 2 и 1

63 Покажите оператор цикла

- A) goto
- B) case
- C) if
- D) while .. do
- E) repeat until

64 В каком месте объявляются постоянные

- A) до объявления всех переменных программы
- B) после объявления переменных программы
- C) в строке объявления переменных процедуры функции
- D) в той части программы где постоянная необходима
- E) после тела программы

65 Как определяются формальные параметры процедуры

- A) введенные в тело процедуры идентификаторы
- B) процедура в части объявления операторов
- C) переменные объявленные в части программы
- D) постоянные в части программы
- E) в конце программы

66 Укажите правильную запись оператора присваивания

Если A,I,J,K : целый; X, Y, Z : вещественный; B : логический;

- 1. $X = Y + \text{Math.Sin}(z);$
 - 2. $A = (X < Y) \text{ or } (B \text{ and } (I < > K));$
 - 3. $X = I + J - B;$
 - 4. $I = I + K / J;$
- A) 1 и 3
 - B) 2 и 4
 - C) 1 и 2
 - D) 3 и 4
 - E) 4 и 1

67 Единица измерения информации

- A) бод
- B) 1 бар
- C) 1 байт
- D) 1 литр
- E) 1 бит

68 Укажите результат выполнения программы

```
{ int x=0,y=5;  
while (x< 6)  
{ y+= x; x+= 2; }
```

- A) 5
- B) 6
- C) 11
- D) 9
- E) 10

69 Вычислить и найти значение выражения
`a = Math.Truncate(5.67) + Math.Round(6.9);`

- A) 0
- B) 11
- C) 12
- D) -1
- E) -2

70 Описание строковых переменных:

- A) string
- B) int
- C) char
- D) double
- E) integer

71 Чему равен результат `Math.Truncate (5,61)`

- A) 5
- B) 61
- C) 6
- D) -5
- E) 10

72 Чему равен результат? `Math.Truncate (-5,61)`

- A) 5
- B) 61
- C) 6
- D) -5
- E) 11

73 Чему равен результат `Math.Round (17,96)`

- A) 17
- B) 96
- C) 18
- D) -18
- E) 12

74 `int s=0, i=1;`

`do { s += 1 / i; ++i; } while (i <= 1);`

После выполнения оператора найти значение переменной `s`

- A) 1
- B) 0
- C) -1
- D) 2
- E) 3

75

После ввода 1, 2, 3 каким будет результат этой программы

```
int a, b;  
a=Convert.ToInt32(Console.ReadLine());  
b=Convert.ToInt32(Console.ReadLine());  
a=Convert.ToInt32(Console.ReadLine());  
Console.Write(a,b,a);
```

- A) 1 2 3
- B) 2 3 2
- C) 3 2 1
- D) 3 2 3
- E) 3 3 3

76 Какие результаты выводит данная программа?

```
int x=2; Console.WriteLine("x+1")
```

- A) x+1
- B) 2
- C) 3
- D) 0
- E) 1

77

```
const d=5;  
d=Math.Sqrt(d);  
Console.WriteLine('d**2='+d)
```

В данной программе в каком фрагменте есть ошибки?

- A) 3, 4
- B) 2, 4
- C) 1, 3
- D) 1, 2, 3
- E) 2,3

78 После выполнения оператора найти значение переменной S.

```
double s=0.0; int n=1,i;  
for ( i=2; i<n; ++i) s+=1/i;
```

- A) 0
- B) 0.5
- C) 1.0

- D) 0.1
- E) 2.8

79 Вычислить и найти значение выражение

$$0/11 + 36/13 - 4\%11$$

- A) 4
- B) 2
- C) -2
- D) -1
- E) -3

80 Что получите в результате выполнения следующей программы, если с экрана введут - 16,17 и 26

```
int P,Q;
P=Convert.ToInt32(Console.ReadLine());
Q=Convert.ToInt32(Console.ReadLine());
P=Convert.ToInt32(Console.ReadLine());
Console. Write ("P+Q="+( P+Q)+"P =" + P);
```

- A) 43 16
- B) P+Q=43 P=26
- C) 33 16
- D) P+Q=33 P=17
- E) P+Q=43 P=16

81 С использованием условного оператора найти значение

$$\text{функции. } Y = \begin{cases} 5x^2 + 6, & x < 0, \\ x^2 - 35, & 0 < x < 6, \\ 7.1x - 7, & x \geq 6, \end{cases}$$

- A) if (X==0) Y=5*Math.Pow(X,2)+6; else
if ((X>0) && (X<6)) Y=Math.Pow(X,2)-35; else Y=7.1*x-7;
- B) if (X<0) Y=5* Pow(X,2)+6; else
if ((X>0) && (X<6)) Y= Pow(X,2)-35; else Y=7.1*x-7;
- C) if (X<0) Y=5*Math.Pow(X,2)+6; else
if ((X>0) && (X<6)) Y=Math.Pow(X,2)-35; else Y=7.1*x-7;
- D) if (X<0) Y=5 Math.Pow(X,2)+6; else
if ((X>0) && (X<6)) Y= Pow(X,2)-35; else Y=7.1 x-7;
- E) if (X<0) then Y=5*Math.Pow(X,2)+6; else
if ((X>0) && (X<6)) Y=Math.Pow(X,2)-35; else Y=7.1*x-7;

82 b=a. Если a-целый тип, тогда b какой тип:

- A) bool
- B) int

- C) string
- D) char
- E) integer

83 Укажите правильный вариант условного оператора:

- A) if условие
Операторы
else
{Операторы};
- B) if условие then {
Операторы
Else
операторы
};
- C) if условие {
Операторы
Else { Операторы};
- D) if (условие)
Оператор;
else { Операторы};
- E) if условие then {
Операторы
Else { }

84 Как описывается постоянное число:

- A) const int n=5;
- B) const n=5;
- C) const n=5 int;
- D) const n=5;
- E) const n=7;

85 Какое условие верно для элементов главной диагонали матрицы

$A[n,n]=\{a_{ij}\}$:

- A) $i+j = n+1$
- B) $i < j$
- C) $i = j$
- D) $i > j$
- E) $j > i$

86 Какое условие верно для элементов побочной диагонали матрицы

$A[n,n]=\{a_{ij}\}$:

- A) $i+j == n+1$
- B) $i < j$
- C) $i == j$

- D) $i > j$
- E) $j > i$

87 Какое условие верно для элементов матрицы расположенных выше главной диагонали $A[n,n] = \{a_{ij}\}$:

- A) $i+j == n+1$
- B) $i < j$
- C) $i == j$
- D) $i > j$
- E) $j > i$

88 Какое условие верно для элементов матрицы расположенных ниже главной диагонали $A[n,n] = \{a_{ij}\}$:

- A) $i+j == n+1$
- B) $i < j$
- C) $i == j$
- D) $i > j$
- E) $j < i$

89 Укажите фрагмент программы для $F = N!$.

- A)
 $F=1;$
 $\text{for (int } i=1; i<N; ++i) F*=i;$
 \dots
- B)
 $F=0;$
 $\text{for (int } i=1; i<N; ++i) F+=i;$
 \dots
- C)
 $F=1;$
 $\text{for (int } i=1; i<N; ++i) F*=N;$
 \dots
- D)
 $F=1;$
 $\text{for (int } i=1; i<N; ++i) F=F*i;$
- E) $F=F*(N+1)$

90 Укажите фрагмент программы для $S = \sum_{i=1}^N i$:

- A)

S=1;
for (int i=1; i<n+1;++i) S=S*i;

...

B)

S=0;
for (int i=1; i<n+1;++i) S+=i;

...

C)

S=1;
for (int i=1; i<n+1;++i) S=S*N;

...

D)

S=1;
for (int i=1; i<n+1;++i) S*=i;

E) F=F*(N+1)

91 Укажите фрагмент программы для вывода чисел от 1 до 10:

A)

```
for (int i=1; i<10; ++i){  
    K=i;  
    Console.WriteLine( K );  
}
```

...

B)

```
for (int i=0; i<10; ++i){  
    K=i;  
    Console.WriteLine( K );  
}
```

...

C)

```
for (int i=1; i<11; ++i){  
    K=i;  
    Console.WriteLine( K );  
}
```

...

D) Console.WriteLine(2*i);

E) :

```
for (int i=1; i>11; ++i){  
    K=i;  
    Console.WriteLine( K );  
}
```

92 Укажите фрагмент программы для вывода четных чисел от 2 до 20:

A)

```
for (int i=0; i<20; ++i){  
    K=2*i;  
    Console.WriteLine( K );  
}
```

...

B)

```
for (int i=1; i<11; ++i){  
    K=2*i;  
    Console.WriteLine( K );  
}
```

...

C)

```
for (int i=1; i<11; ++i){  
    K=i;  
    Console.WriteLine( K );  
}
```

...

D) for (int i=1; i<11; i+=2){
 K=i;
 Console.WriteLine(K);
}

93 Укажите фрагмент программы для вывода нечетных чисел от 1 до 21:

A)

```
for (int i=0; i<21; ++i){  
    K=2*i-1;  
    Console.WriteLine( K );  
}
```

...

B)

```
for (int i=1; i<11; ++i){  
    K=2*i-1;  
    Console.WriteLine( K );  
}
```

...

C)

```
for (int i=1; i<11; ++i){  
    K=i;  
    Console.WriteLine( K );  
}
```

...

D)

```

K=2*i-1;
Console.WriteLine( K );
E) :
for (int i=1; i<11; i+=2){
    K=i-1;
    Console.WriteLine( K );
}

```

94 Укажите фрагмент программы для вывода квадратов чисел от 1 до 10

A)

```

for (int i=1; i<11;++i) {
    K=Math.Pow(i,2);
    Console.WriteLine(K);
}

```

...

B)

```

for (int i=0; i<10;++i) {
    K=Math.Pow(i,2);
    Console.WriteLine(K);
}

```

...

C)

```

for (int i=1; i<11;++i) {
    K=SQR(i);
    Console.WriteLine(K);
}

```

...

D)

```

for (int i=1; i>11;++i) {
    K=Math.Pow(2,i);
    Console.WriteLine(K);
}

```

...

95 Укажите правильную запись на C# $a + x^2 - \frac{a + bx}{3}$.

- A) $a+2*x-a+b*x/3$
- B) $a+x*x-(a+b*x)/3$
- C) $(a+x*2)-a+bx/3$
- D) $a+SQR(x)-a+b*x/3$
- E) $a+2*x-a+b*x/3$

96 Если X=7, Y=15, Z=11, R=450,08 Укажите результат работы оператора Console.Write (X+""+Y);

- A) 7 11
- B) X =7
- C) 7_,_15
- D) 7 15
- E) 7^11

97 Если X=7, Y=15, Z=11, R=450,08 Укажите результат работы оператора Console.Write (X+"_,_" +Y);

- A) 7 11
- B) X =7
- C) 7_,_15
- D) 715
- E) 7,15

98 Если X=7, Y=15, Z=11, R=450,08 Укажите результат работы оператора Console.Write ("X=" +X);

- A) 7 11
- B) X =7
- C) 7_,_15
- D) 715
- E) 455

99 Если X=7, Y=15, Z=11, R=450,08 Укажите результат работы оператора Console.Write ("X+Y="+(X+Y));

- A) 11- результат
- B) X+Y =22
- C) end
- D) 450,08
- E) 555

100 Если X=7, Y=15, Z=11, R=450,08 Укажите результат работы оператора Console.Write (Z+"- результат");

- A) 11- результат
- B) X+Y =22
- C) end
- D) 450,08
- E) 666

101 Если X=7, Y=15, Z=11, R=450,08 Укажите результат работы оператора Console.Write ("end ");

- A) 11- результат
- B) X+Y =22

- C) end
- D) 450,08
- E) 45

102 Если $X=7$, $Y=15$, $Z=11$, $R=450,08$ Укажите результат работы оператора Console.Write (R);

- A) 11- результат
- B) $X+Y = 22$
- C) end
- D) 450,08
- E) 33

103 Укажите результат $20 \% 7 + 5 / 4 * 2$

- A) 0
- B) 11
- C) 2
- D) 8
- E) 33

104 Укажите результат $48 \% (2+3) / 4$

- A) 0
- B) 11
- C) 2
- D) 8
- E) 10

105 Укажите результат $((24 \% 7 + 5) / 4) * 2$:

- A) 4
- B) 11
- C) 2
- D) 8
- E) 10

106 Укажите результат $16 \% (7+9) / 3 * 2$

- A) 0
- B) 11
- C) 2
- D) 8
- E) 10

107 Укажите результат $X : (x \text{ целое число})$

$$x / 5 = x \% 5$$

- A) 30
- B) 45

- C) 43
- D) 12
- E) 10

108 Укажите результат X: (x целое число)

$$20 / x = 20 \% x$$

- A) 0
- B) 10
- C) 19
- D) 12
- E) 11

109 Укажите результат X: (x целое число)

$$x / 5 = 8$$

- A) 30
- B) 75
- C) 53
- D) 42
- E) 12

110 Укажите результат X:

$$50 \% x = 7$$

- A) 3
- B) 7
- C) 43
- D) 1
- E) 13

111 Каковы значения X и Y после выполнения:

```
int x=15 / (8 % 3);
```

```
int y=17 % x*5-19 % 6*2;
```

- A) x=7; y=13
- B) x=7; y=4
- C) x=12; y=7
- D) x=1; y=7
- E) x=5

112 Каковы значения X и Y после выполнения:

```
int x=2*5 / 3 % 2;
```

```
int y=2*5 /(3 % 2);
```

```
x*=y; y=y*y ;
```

- A) x=10; y=10
- B) x=7; y=49
- C) x=12; y=144

- D) $x=10$; $y=100$
E) $x=11$

113

После выполнения фрагмента программы , чему будет равно число X

```
int x=5;  
if (x>=5) x*=2; else if (x<=10) x*=-1;  
x*=5;
```

- A) $x=5$
B) $x=10$
C) $x=-5$
D) $x=50$
E) $x=8$

114 После выполнения фрагмента программы , X будет равно числу

```
int x=5;  
if (x>=5) x*=2; else if (x<=10) x*=-1;
```

- A) $x=5$
B) $x=10$
C) $x=-5$
D) $x=-10$
E) $x=22$

115 После выполнения фрагмента программы , чему будет равно число A и B

```
int a=1, b=2;  
if (a<b) a+=1; else if (a=b)  
a+=2;  
else if (a>b)  
b+=2;
```

- A) $a=2$; $b=5$
B) $a=10$; $b=2$
C) $a=-2$; $b=4$
D) $a=2$; $b=2$
E) $a=6$

116 После выполнения фрагмента программы , будет равно числу Aи B?

```
int a=1, b=2;  
if (a>b) a+=1; else a=b;
```

- A) $a=2$; $b=2$
B) $a=2$; $b=1$
C) $a=2$; $b=4$
D) $a=1$; $b=2$
E) $a=7$

117 После выполнения фрагмента программы , А и В будет равно числу

```
int a=12, b=2;
```

```
if (a<b) a+=10; else b*=2;
```

A) a=12; b=2

B) a=22; b=12

C) a=12; b=4

D) a=24; b=2

E) a=8

118 После выполнения фрагмента программы , А и В равна

```
int a=2, b=4;
```

```
if (a<b) a+=2; else if (a=b)
```

```
    a+=2;
```

```
    else if (a>b)
```

```
        b+=2
```

A) a=2; b=5

B) a=10; b=2

C) a=-2; b=4

D) a=4; b=4

E) a=7

119 Вычислите значение S.

```
int S=0, I=1;
```

```
while (I<10){
```

```
    S+=(2*I+1);
```

```
    I+=2};
```

A) 52

B) 55

C) 75

D) 65

E) 33

120 Сколько раз выполнится тело цикла

```
int N=0;
```

```
for (int I=1; I<3*N+4; ++I)
```

```
    N+=2;
```

A) 2

B) 5

C) 4

D) 3

E) 11

121 Укажите правильный вариант записи.

A) int[10] b=new int[];

- B) `int[] b=new int[10];`
- C) `int[] b=new int[0..10];`
- D) `integer[] b=new integer[10];`
- E) `new int[] b=int[10];`

122 Дана матрица A[10,10]. Построить массив B[10] по следующему правилу: элементы B[0]- сумма элементов 0-ой строки, B[1]-сумма элементов 1-ой строки.

- A)

```
for (int I=0; I<10; ++I) {  
    for (int J=0; J<10; ++J) {  
        B[I]+=A[I,J];  
    }  
    Console.WriteLine(B[I]);  
}
```
- B)

```
for (int i=0; i<10; ++i) {  
    for (int j=0; j<10; ++j) {  
        B[i]+=A[i,j];  
    }  
    Console.WriteLine(B[j]);  
}
```
- C)

```
for (int I=0; I<10; ++I) {  
    for (int J=0; J<10; ++J) {  
        B[I]=A[I,J];  
    }  
    Console.WriteLine(B[I]);  
}
```
- D)

```
for (int I=0; I<10; ++I) {  
    for (int J=0; J<10; ++J) {  
        B[i]+=A[i,j];  
    }  
    Console.WriteLine(B[i]);  
}
```
- E)

```
for (int J=1; J<10; ++J) {  
    for (int I=1; I<10; ++I) {  
        B[I]+=A[I,J];  
    }  
    Console.WriteLine(B[I]);  
}
```

123 Дана матрица A [12,12]. Найти максимальный элемент матрицы.

- A)

```
int MAX=A[1,1];  
for (int I=0; I<13; ++I)  
for (int J=0; J<13; ++J)  
if (A[I,J]<MAX) MAX=A[I,J];
```

```

B) int MAX=A[1,1];
for (int I=1; I<12; ++I)
for (int J=1; J<12; ++J)
if (A[I,J]>MAX) MAX=A[I,J];
C) int MAX=A[1,1];
for (int I=0; I<13; ++I)
for (int J=0; J<13; ++J)
if (A[I,J]>MAX) MAX=A[I,J];
D) int MAX=A[1,1];
for (int I=1; I<12; ++I)
for (int J=1; J<12; ++J)
if (A[I,J]<MAX) MAX=A[I,J];
E) int MAX=A[1,1];
for (int I=1; I<13; ++I)
if (A[I,J]>MAX) MAX=A[I,J];

```

124 Дана матрица A[12,12]. Присвоить нечетным столбцам отрицательные значения.

```

A) for (int i=1; i<13; ++i )
for (int j=1; j<13; ++j )
    A[i,2*j]*=-1;
B) for (int i=1; i<12; ++i )
for (int j=1; j<12; ++j )
    A[i,2*j]*=-1;
C) for (int i=1; i<12; ++i )
for (int j=1; j<6; ++j )
    A[i,2*j-1]*=-1;
D) for (int i=0; i<13; ++i )
for (int j=0; j<7; ++j )
    A[i,2*j-1]*=-1;
E) for (int i=1; i<13; ++i )
for (int i=1; i<7; ++i )
    A[i,2*j+1]*=-1;

```

125 Что выведет данный фрагмент

```

public int Q( int X; int Y)
{
    return X+Y;

```

```

}

```

Основная программа

```

int C = 2, D = 0; int P=Q(C,D) ; Console.Write(P);

```

A) 2
B) 0

- C) 5
- D) 10
- E) 11

126 Если L-целый, S-постоянный, P-действительный тип, тогда как описываются эти переменные

- A) int p: double l; const int s=2;
- B) int const s=2; int L; double P;
- C) const s=2; int L; double P;
- D) const int s=2; int L; double P;
- E) s=2 const; int L; double P;

127 Укажите правильный оператор цикла:

- A) while условие do
{Операторы};
- B) while (условие)
{Операторы }
- C) while условие do
Операторы
end;
- D) while условие do
Операторы
end;
- E) while условие do
Операторы

128 Укажите правильный оператор цикла:

- A) do
{Операторы}
while (условие);
- B) while
Операторы
do условие;
- C) repeat
{Операторы
until условие;
}
- D) do
Операторы
while условие;
- E) while условие do
{операторы}

129 Укажите оператор ввода переменной “a”:

- A) Read (a)
- B) Write (a)
- C) Input a
- D) Print a
- E) Label a

130 Укажите оператор вывода переменной “a”:

- A) Read (a)
- B) Write (a)
- C) Input a
- D) Print a
- E) Label a

131 Укажите правильное описание функции $tg x$

- A) Tan(X)
- B) TGx
- C) Atan(x)
- D) Math.Sin(x)/Math.Cos(x)
- E) Math.Artan(x)

132 Модуль числа X:

- A) Sqr(x)
- B) Sqrt(x)
- C) Mod(x)
- D) Abs(x)
- E) Sin(x)/Cos(x)

133 Квадрат числа X:

- A) Math.Pow (x,2)
- B) Sqr(x)
- C) Mod(x)
- D) Abs(x)
- E) Sin(x)/cos(x)

134 Корень квадратный числа X:

- A) Sqr(x)
- B) Sqrt(x)
- C) Mod(x)
- D) Abs(x)
- E) Sin(x)/cos(x)

135 Остаток от деления числа x на b:

- A) x%b
- B) x div b

- C) x/b
- D) `abs(x,b)`
- E) `x mod b`

136 Укажите правильное описание возведение x в степень a :

- A) `sqrt(x,a)`
- B) `Pow(x,a)`
- C) `asqrt(a,x)`
- D) `exp(x,a)`
- E) `Pow(a,x)`

137 Какой знак ставится в конце каждого оператора:

- A) `:`
- B) `;`
- C) `/`
- D) `\`
- E) `*`

138 Возведение числа в квадрат:

- A) `**`
- B) `^`
- C) `pow(x,2)`
- D) `sqrt`
- E) `*`

139 Целочисленное деление числа A на B :

- A) `A / B`
- B) `A % B`
- C) `A chr B`
- D) `A and B`
- E) `A orx B`

140 Остаток от деления числа A на B :

- A) `A / B`
- B) `A % B`
- C) `A chr B`
- D) `A || B`
- E) `A && B`

141 Если $\dot{A}=5$, `Console.Write ("B – C =" + A)` ; Тогда что выйдет на экран:

- A) $A=5$
- B) $B=A+C$
- C) $B - C=5$

- D) $B - C = A$
- E) A and B

142 $j=10$, $A[10]=45$, `Console.Write ("A("+j+")="+A[10]);` Что выйдет на экран монитора:

- A) $A[j]=45$
- B) 45
- C) 10
- D) $A(10)=45$
- E) 15

143 Что выйдет на экран в результате выполнения программы:

$A=5$; $B=19$;

$C=A+B$;

`Console.Write (C)`

- A) 5
- B) 19
- C) 24
- D) 0
- E) 15

144 Что выйдет на экран в результате выполнения программы:

$A=15$; $B=30$; $C=0$;

`Console.Write (C)`

$C=A+B$;

- A) 15
- B) 45
- C) 0
- D) 30
- E) 20

145 Что выйдет на экран в результате выполнения программы:

$A=5$; $B=19$; $C=3$;

$C*=A$;

`Write (C)`

- A) 19
- B) 5
- C) 24
- D) 15
- E) 20

146 Что выйдет на экран в результате выполнения программы:

$S=0$;

`For (int j=1; i<5; ++i) S+=j;`

Console.Write (S);

- A) 0
- B) 5
- C) 15
- D) 6
- E) 20

147 Выберите вариант при котором X будет четным:

- A) $X=2*a-1$
- B) $X=2*a$
- C) $X=2*a+1$
- D) $X=3*a$
- E) $X=(3*a)$

148 Выберите вариант при котором X будет нечетным($a=1:N$):

- A) $X=2*a-1$
- B) $X=2*a$
- C) $X=2*a+1$
- D) $X=3*a$
- E) $X=3*a$

149 Выберите вариант при котором остаток от деления будет равен 0,если $X=51$:

- A) $X \% 3=0$
- B) $\text{abs}(x/3)=0$
- C) $\text{sqrt}(x/3)=0$
- D) $X/3=0$
- E) $X=3*a$

150 Выберите вариант при котором X будет положительным:

- A) $X=\text{sqrt}(x)-1$
- B) $X=\text{len}(x)$
- C) $X=\text{abs}(x)$
- D) $X=\text{pow}(x,3)$
- E) $X=3*a$

СПИСОК ЛИТЕРАТУРЫ

- 1) Д.Либерти. Создание .NET приложений Программирование на С#. – М.:
- 2) Ишкова Э.А. С#. Начала программирования. – М.: ООО «Бином-Пресс», 2007. – 336с.
- 3) Карли Уотсон, Кристиан Нейгел, Якоб Хаммер Педерсон, Джон Д.Рид, Морган Скиннер, Эрик Уайт. Microsoft Visual C# 2008. – М.: «Диалектика», 2009. – 1216с.
- 4) Культин Н.Б. Microsoft Visual C# в задачах и примерах. Спбю: БХВ- Петербург, 2009. – 240с.
- 5) Культин Н.Б. С# в задачах и примерах. – Спб.: БХВ-Петербург, 2007.- 240с.
- 6) Павловская Т.А. С#. Программирование на языке высокого уровня. Учебник для вузов. – Спб.: Питер, 2007. – 432с.
- 7) Фаронов В. В. Программирование на языке С#– М.: Изд-во «Эксмо», 2008. – 425с.
- 8) Фаронов В. В. Создание приложений с помощью С#. Руководство программиста – М.: Изд-во «Эксмо», 2008. – 576с.
- 9) Фролов А. В., Фролов Г. В. Язык С#. Самоучитель.- М.:Диалог-МИФИ, 2003. – 560с.
- 10) Электронная база данных. <http://www.libkruz.com/1-41/c.html>.

СОДЕРЖАНИЕ

	стр
ВВЕДЕНИЕ	3
Глава 1. Основы программирования на языке C#	7
1.1 Элементы языка C#.....	7
1.2 Переменные.....	8
1.2.1 Типы данных.....	8
1.2.2 Арифметические операции.....	20
1.2.3 Логические операции.....	22
1.2.4 Математические операции.....	22
1.3 Управляющие структуры языка C#.....	27
1.3.1 Структура выбора if/else.....	28
1.3.2 Тернарная структура выбора.....	35
1.3.3 Оператор выбора.....	36
1.3.4 Операторы повторений.....	41
1.4 Массивы.....	50
1.4.1 Одномерные массивы.....	50
1.4.2 Многомерные массивы.....	54
1.5 Функции.....	60
1.6 Строки.....	68
Глава 2. Объектно-ориентированное программирование	110
2.1 Введение в объектно-ориентированное программирование.....	110
2.1.1 Классы и объекты.....	110
2.1.2 Конструкторы и деструкторы.....	116
2.1.3 Наследование.....	124
2.1.4 Полиморфизм.....	130
2.2 Файловый ввод-вывод.....	136
2.3 Обработка исключительных ситуаций.....	154
Приложение А. Задачник	163
Приложение Б. Тесты	171