# Глава 2. Въведение в програмирането

## В тази тема...

В настоящата тема ще разгледаме основните термини от програмирането и ще напишем **първата си програма на C#**. Ще се запознаем с това какво е програмиране и каква е връзката му с компютрите и програмните езици.

Накратко ще разгледаме основните етапи при писането на софтуер.

Ще въведем **езика C#** и ще се запознаем с .NET платформата и техноло­гиите на Microsoft за разработка на софтуер. Ще разгледаме какви помощни средства са ни необходими, за да можем **да програмираме на C#**. Ще използваме езика C#, за да напишем първата си програма, ще я компилираме и изпълним както от командния ред, така и от средата за разработка **Microsoft Visual Studio**. Ще се запознаем още и с MSDN Library – документацията на .NET Framework, която ни помага при по-нататъшно изследване на възможностите на езика и платформата.

## Какво означава "да програмираме"?

В днешно време компютрите навлизат все по-широко в ежедневието ни и все повече имаме нужда от тях, за да се справяме със сложните задачи на работното място, да се ориентираме, докато пътуваме, да се забав­ляваме или да общуваме. Неизброимо е приложението им в бизнеса, в развлека­телната индустрия, в далекосъобщенията и в областта на финансите. Няма да преувеличим, ако кажем, че **компютрите изграждат нервната система на съвременното общество** и е трудно да си представим съществу­ването му без тях.

Въпреки масовото им използване, малко хора имат представа как всъщ­ност работят компютрите. На практика не компютрите, а програмите, които се изпълняват върху тях (софтуерът), имат значение. Този софтуер придава стойността за потребителите и чрез него се реализират различ­ните типове услуги, променящи живота ни.

### Как компютрите обработват информация?

За да разберем какво значи да програмираме, нека грубо да сравним компютъра и операционната система, работеща на него, с едно голямо предприятие заедно с неговите цехове, складове и транспортни меха­низми. Това сравнение е грубо, но дава възможност да си представим степента на сложност на един съвременен компютър. В компютъра работят много **процеси**, които съответстват на цеховете и поточните линии в предприятието. Твърдият диск заедно с файловете на него и оперативната (RAM) памет съответстват на складовете, а различните протоколи са транспортните системи, внасящи и изнасящи информация.

Различните видове продукция в едно **предприятие** се произвеждат в различните цехове. Цеховете използват суровини, които взимат от скла­довете, и складират готовата продукция обратно в тях. Суровините се транспортират в складовете от доставчиците, а готовата продукция се транспортира от складовете към пласмента. За целта се използват различни видове транспорт. Материалите постъпват в предприятието, минават през различни стадии на обработка и напускат предприятието, преобразувани под формата на продукти. Всяко предприятие преобразува суровините в готов за употреба продукт.

Компютърът е **машина за обработка на информация** и при него както суровината, така и продукцията е информация. Входната информация най-често се взима от някой от складовете (файлове или RAM памет), където е била транспортирана, преминава през обработка от един или повече процеси и излиза модифицирана като нов продукт. Пример за това са уеб базираните приложенията. При тях за транспорт както на сурови­ните, така и на продукцията, се използва протоколът HTTP, а обработката на информация обикновено е свързана с извличане на съдържание от база данни и подготовката му за визуализация във вид на HTML.

### Управление на компютъра

Целият процес на изработка на продуктите в едно предприятие има много степени на **управ­ление**. Отделните машини и поточни линии се управля­ват от оператори, цеховете се управляват от управители, а предприятието като цяло се управлява от директори. Всеки от тях упражнява контрол на различно ниво. Най-ниското ниво е това на машинните оператори – те управляват машините, образно казано, с помощта на копчета и ръчки. Следващото ниво е на управителите на цехове. На най-високо ниво са директорите, те управляват различните аспекти на производствените процеси в предприя­тието. Всеки от тях управ­лява, като издава заповеди.

По аналогия при компютрите и софтуера има много нива на управление. На най-ниско **машинно ниво** се управлява самият процесор и регистрите му (чрез машинни програми на ниско ниво) – можем да сравним това с управлението на машините в цеховете. На по-високо **системно ниво** се управляват различните отговорности на операционната система (напри­мер Windows 7) като файлова система, периферни устройства, потре­бители, комуникационни протоколи – можем да сравним това с управле­нието на цеховете и отделите в предприятието. На най-високо ниво в софтуера са приложе­нията (**приложните програми**). При тях се управлява цял ансамбъл от процеси, за изпълнението на които са необходими огро­мен брой операции на процесора. Това е нивото на директорите, които управляват цялото предприятие с цел мак­симално ефективно използване на ресурсите за получаване на качествени резултати.

### Същност на програмирането

Същността на програмирането е да се управлява работата на компютъра на всичките му нива. Управлението става с помощта на "**заповеди**" и "**команди**" от програмиста към компютъра, известни още като програмни инструкции. Да програмираме, означава **да организираме управлението на компютъра с помощта на поредици от инструкции**. Тези заповеди (инструкции) се издават в писмен вид и биват безпрекословно изпълня­вани от компютъра (съответно от операционната система, от процесора и от периферните устройства).

Програмистите са хората, които създават инструкциите, по които работят компютрите. Тези **инструкции** се наричат **програми**. Те са много на брой и за изработката им се използват различни видове програмни езици. Всеки език е ориентиран към някое ниво на **управление на компю­търа**. Има езици, ориентирани към машинното ниво – например асемблер, други са ориентирани към системното ниво (за взаимодействие с операционната система), например C. Същест­вуват и езици от високо ниво, ориентирани към писането на приложни програми. Такива са езиците C#, Java, C++, PHP, JavaScript, Visual Basic, Python, Ruby, Perl и други.

В настоящата книга ще разгледаме **програмния език C#**, който е съвременен език за програмиране от високо ниво. При използването му позицията на програмиста в компютърното предприятие се явява тази на директора. Инструкциите, подадени като програми на C# могат да имат достъп и да управляват почти всички ресурси на компютъра директно или посредством операционната система. Преди да разгледаме как може да се използва C# за писане на прости компютърни програми, нека разгледаме малко по-широко какво означава да разработваме софтуер, тъй като програмирането е най-важната дейност в този процес, но съвсем не е единствената.

## Етапи при разработката на софтуер

Писането на софтуер може да бъде сложна задача, която отнема много време на цял екип от **софтуерни инженери** и други специалисти. Затова с времето са се обособили различни методики и практики, които улесняват живота на програмис­тите. Общото между всички тях е, че разработката на всеки софтуерен продукт преминава през няколко **етапа**, а именно:

* Събиране на изискванията за продукта и изготвяне на задание;
* Планиране и изготвяне на архитектура и дизайн;
* Реализация (включва писането на програмен код);
* Изпитания на продукта (тестове);
* Внедряване и експлоатация;
* Поддръжка.

Фазите реализация, изпитания, внедряване и поддръжка се осъществяват в голямата си част с помощта на **програмиране**.

### Събиране на изискванията и изготвяне на задание

В началото съществува само идеята за определен продукт. Тя включва набор от изисквания, дефиниращи действия от страна на потребителя и компютъра, които в общия случай улесняват извършването на досега съществуващи дейности. Като пример може да дадем изчисляването на заплатите, пресмятане на балистични криви, търсене на най-пряк път в Google Maps. Много често софтуерът реализира несъществуваща досега функционалност като например авто­матизиране на някаква дейност.

Изискванията за продукта обикновено се дефинират под формата на доку­менти, написани на естествен език – български, английски или друг. На този етап не се програмира. Из­искванията се дефинират от експерти, запознати с проблемати­ката на конкретната област, които умеят да ги описват в разбираем за програмис­тите вид. В общия случай тези експерти не са специалисти по програ­ми­ране и се наричат **бизнес анализа­тори**.

### Планиране и изготвяне на архитектура и дизайн

След като изискванията бъдат събрани, идва ред на етапа на планиране. През този етап се съставя технически план за изпълнението на проекта, който описва платформите, техноло­гиите и първоначалната архитектура (дизайн) на програмата. Тази стъпка включва значителна творческа работа и обикно­вено се реализира от софтуерни инженери с много голям опит, наричани понякога **софтуерни архитекти**. Съобразно изисквани­ята се избират:

* Вида на приложението – например конзолно приложение, настолно приложение (GUI, Graphical User Interface application), клиент-сървър приложение, уеб при­ложение, Rich Internet Application (RIA) или peer-to-peer прило­жение;
* Архитектурата на програмата – например еднослойна, двуслойна, трислойна, многослойна или SOA архитектура;
* Програмният език, най-подходящ за реализирането – например C#, Java или C++, или комбинация от езици;
* Технологиите, които ще се ползват: платформа (например Microsoft .NET, Java EE, LAMP или друга), сървър за бази данни (например Oracle, SQL Server, MySQL или друг), технологии за потребителски интер­фейс (например Flash, JavaServer Faces, Eclipse RCP, ASP.NET, Windows Forms, WPF или други), технологии за достъп до данни (например Hibernate, JPA или LINQ-to-SQL), технологии за изготвяне на отчети (например SQL Server Reporting Services, Jasper Reports или други) и много други техно­логии и комбинации от технологии, които ще бъдат използвани за реализирането на раз­лични части от софтуерната система.
* Броят и уменията на хората, които ще съставят екипа за разработка (големите и сериозни проекти се пишат от големи и сериозни екипи от разработчици);
* План на разработката – етапи, на които се разделя функционал­ността, ресурси и срокове за изпълнението на всеки етап.
* Други (големина на екипа, местоположение на екипа, начин на комуникация и т.н.).

Въпреки че съществуват много правила, спомагащи за правилния анализ и планиране, на този етап се изискват значителна интуиция и усет. Тази стъпка предопределя цялостното по-нататъшно развитие на процеса на разработка. На този етап не се извършва програмиране, а само подго­товка за него.

### Реализация

Етапът, най-тясно свързан с програмирането, е етапът на реализацията (имплементацията). На този етап съобразно заданието, дизайна и архи­тектурата на програмата (приложението) се пристъпва към реализирането (написва­нето) й. Етапът "реализация" се изпълнява от **програмисти**, които пишат програм­ния код (сорс кода). При малки проекти останалите етапи могат да бъдат много кратки и дори да липсват, но етапът на реализация винаги се извършва, защото иначе не се изработва софтуер. Настоящата книга е посветена главно на описание на средствата и похва­тите, използвани на този етап – изграждане на програмистско мислене и използване на средствата на езика C# и платформата .NET Framework за реализация на софтуерни приложения.

### Изпитания на продукта (тестове)

Важен етап от разработката на софтуер е етапът на изпитания на продукта. Той цели да удостовери, че реализацията следва и покрива изискванията на заданието. Този процес може да се реализира ръчно, но предпочитаният вариант е написването на автоматизирани тестове, които да реализират проверките. Тестовете са малки програми, които автомати­зират, до колкото е възможно, изпитанията. Съществуват парчета функ­ционалност, за които е много трудно да се напишат тестове и поради това процесът на изпитание на продукта включва както автоматизирани, така и ръчни процедури за проверка на функционалността и качеството.

Процесът на тестване (изпитание) се реализира от екип **инженери по осигуряването на качеството** – quality assurance (QA) инженери. Те работят в тясно взаимодействие с програмистите за откриване и кориги­ране на дефектите (бъговете) в софтуера. На този етап почти не се пише нов програ­мен код, а само се отстраняват дефекти в съществуващия код.

В процеса на изпитанията най-често се откриват множество пропуски и грешки и програмата се връща обратно в етап на реализация. До голяма степен етапите на реализация и изпитания вървят ръка за ръка и е възможно да има множество преминавания между двете фази преди продуктът да е покрил изискванията на заданието и да е готов за етапа на внедряване и експлоатация.

### Внедряване и експлоатация

Внедряването или инсталирането (deployment) е процесът на въвеждане на даден софтуерен продукт в експлоатация. Ако продуктът е сложен и обслужва много хора, този процес може да се окаже най-бавният и най-скъпият. За по-малки програми това е относително бърз и безболезнен процес. Най-често се разработва специална програма – инсталатор, която спомага за по-бързата и лесна инсталация на продукта. Понякога, ако продуктът се внедрява в големи корпорации с десетки хиляди копия, се разработва допълнителен поддържащ софтуер специално заради внедря­ването. След като внедряването приключи, продуктът е готов за експлоа­тация и следва обучение на служителите как да го ползват.

Като пример можем да дадем внедряването на Microsoft Windows в българ­ската държавна администрация. То включва инсталиране и конфигури­ране на софтуера и обучение на служителите.

Внедряването се извършва обикновено от екипа, който е разработил продукта или от специално обучени **специалисти по внедряването**. Те могат да бъдат системни администратори, администратори на бази данни (DBA), системни инженери, специализирани консултанти и други. В този етап почти не се пише нов код, но съществу­ващият код може да се доработва и конфигурира докато покрие специ­фичните изисквания за успешно внедряване.

### Поддръжка

В процеса на експлоатация неминуемо се появяват проблеми – заради грешки в самия софтуер или заради неправилното му използване и конфигурация или най-често заради промени в нуждите на потребителите. Тези проблеми довеждат до невъзможност за решаване на бизнес зада­чите чрез употреба на продукта и налагат допълнителна намеса от страна на разработчиците и експертите по поддръжката. Процесът по поддръжка обикновено продължава през целия период на експлоатация, независимо колко добър е софтуерният продукт.

Поддръжката се извършва от екипа по разработката на софтуера и от специално обучени **експерти по поддръжката**. В зависимост от проме­ните, които се правят, в този процес могат да участват бизнес анализа­тори, архитекти, програмисти, QA инженери, администратори и други.

Ако например имаме софтуер за изчисление на работни заплати, той ще има нужда от актуализация при всяка промяна на данъчното законода­телство, което касае обслужвания счетоводен процес. Намеса на екипа по поддръжката ще е необходима и например ако бъде сменен хардуерът, използван от крайните клиенти, защото програмата ще трябва да бъде инсталирана и конфигурирана наново.

### Документация

Етапът на документацията всъщност не е отделен етап, а съпътства всички останали етапи. Документацията е много важна част от разработката на софтуер и цели предаване на знания между различните участници в разработката и поддръжката на продукта. Информацията се предава както между отделните етапи, така и в рамките на един етап. Документацията обикновено се прави от самите разработчици (архитекти, програмисти, QA инженери и други) и представлява съвкупност от документи.

### Разработката на софтуер не е само програмиране

Както сами се убедихте, разработването на софтуер не е само програми­ране и включва много други процеси като анализ на изискванията, проектиране, планиране, тестване и поддръжка, в които участват не само програмисти, но и много други специалисти, наричани **софтуерни инже­нери**. Програмирането е само една малка, макар и много съществена, част от разработката на софтуера.

В настоящата книга ще се фокусираме само и единствено върху **програми­рането**, което е единственото действие от изброените по-горе, без което не можем да разработваме софтуер.

## Нашата първа C# програма

Преди да преминем към подробно описание на езика C# и на .NET платформата, нека да се запознаем с прост пример на това какво представлява една програма, написана на C#:

|  |
| --- |
| class HelloCSharp  {  static void Main(string[] args)  {  System.Console.WriteLine("Hello, C#!");  }  } |

Единственото нещо, което прави тази програма, е да изпише съобщението "Hello, C#!" на стандартния изход. Засега е още рано да я изпълняваме и затова само ще разгледаме структурата й. Малко по-нататък ще дадем пълно описание на това как да се компилира и изпълни дадена програма както от командния ред, така и от среда за разработка.

### Как работи нашата първа C# програма?

Нашата първа програма е съставена от три логически части:

* Дефиниция на клас HelloCSharp;
* Дефиниция на метод Main();
* Съдържание на метода Main().

#### Дефиниция на клас

На първия ред от нашата програма дефинираме клас с името HelloCSharp. Най-простата дефиниция на клас се състои от ключовата дума class, следвана от името на класа. В нашия случай името на класа е HelloCSharp. Съдържанието на класа е разположено в блок от програмни редове, ограден във фигурални скоби: {}.

#### Дефиниция на метод Main()

На третия ред дефинираме метод с името Main(), който представлява входна или стартова точка за програмата. Всяка програма на C# се стартира от метод Main()със следната заглавна част (сигна­тура):

|  |
| --- |
| static void Main(string[] args) |

Методът трябва да е деклариран точно по начина, указан по-горе, трябва да е static и void, трябва да има име Main и като списък от параметри трябва да има един единствен параметър от тип масив от string. В нашия пример параметърът се казва args, но това не е задължително. Тъй като този параметър обикновено не се използва, той може да се пропусне. В такъв случай входната точка на програмата може да се опрости и да добие следния вид:

|  |
| --- |
| static void Main() |

Ако някое от гореспоменатите изисквания не е спазено, програмата ще се компилира, но няма да може да се стартира, защото не е дефинирана коректно нейната входна точка.

#### Съдържание на Main() метода

Съдържанието на всеки метод се намира след сигнатурата на метода, заградено от отваряща и затваряща къдрави скоби. На следващия ред от примерната програма използваме системния обект System.Console и неговия метод WriteLine(), за да изпишем някакво съобщение в стан­дартния изход (на конзолата) в случая текста "Hello, C#!".

В Main() метода можем да напишем произволна последователност от изрази и те ще бъдат изпълнени в реда, в който сме ги задали.

Подробна информация за изразите може да се намери в главата "[Оператори и изрази](#_Глава_3._Оператори)", работата с конзолата е описана в главата "[Вход и изход от конзолата](#_Глава_4._Вход)", а класовете и методите са описани подробно в главата "[Дефиниране на класове](#_Глава_14._Дефиниране_1)".

### C# различава главни от малки букви!

В горния пример използвахме някои ключови думи, като class, static и void и имената на някои от системните класове и обекти, като System.Console.

|  |  |
| --- | --- |
|  | Внимавайте, докато пишете! Изписването на един и същ текст с главни, малки букви или смесено в C# означава различни неща. Да напишем Class е различно от **class** и да напишем System.Console е различно от SYSTEM.CONSOLE. |

Това правило важи за всички конструкции в кода – **ключови думи, имена на променливи, имена на класове** и т.н.

### Програмният код трябва да е правилно форматиран

Форматирането представлява добавяне на символи, несъществени за ком­пилатора, като **интервали, табулации и нови редове**, които структури­рат логически програмата и улесняват четенето й. Нека отново разгледаме кода на нашата първа програма (с краткия вариант за Main() метод):

|  |
| --- |
| class HelloCSharp  {  static void Main()  {  System.Console.WriteLine("Hello, C#!");  }  } |

Програмата съдържа седем реда и някои от редовете са повече или по-малко отместени навътре с помощта на табулации. Всичко това можеше да се напише и без отместване, например така:

|  |
| --- |
| class HelloCSharp  {  static void Main()  {  System.Console.WriteLine("Hello, C#!");  }  } |

или на един ред:

|  |
| --- |
| class HelloCSharp{static void Main(){System.Console.WriteLine( "Hello, C#!");}} |

или дори така:

|  |
| --- |
| class  HelloCSharp  {  static void Main()  { System .  Console.WriteLine("Hello, C#!") ;} } |

Горните примери ще се компилират и **изпълнят по абсолютно същия начин** като форматирания, но са **далеч по-нечетливи**, трудни за разбиране и осмисляне и съответно неудобни за промяна.

|  |  |
| --- | --- |
|  | Не допускайте програмите ви да съдържат неформатиран код! Това силно намалява четимостта и довежда до трудно модифициране на кода. |

#### Основни правила на форматирането

За да е форматиран кодът, трябва да следваме няколко важни правила за отместване:

* Методите се отместват по-навътре от дефиницията на класа;
* Съдържанието на методите се отмества по-навътре от дефиницията на метода;
* Отварящата фигурна скоба { трябва да е сама на ред и да е разполо­жена точно под метода или класа, към който се отнася;
* Затварящата фигурна скоба } трябва да е сама на ред и да е поставена вертикално точно под съответната й отваряща скоба (със същото отместване като нея);
* Имената на класовете трябва да започват с главна буква;
* Имената на променливите трябва да започват с малка буква;
* Имената на методите трябва да започват с главна буква;

### Имената на файловете съответстват на класовете

Всяка C# програма се състои от един или няколко класа. Прието е всеки клас да се дефинира в отделен файл с име, съвпадащо с името на класа и разширение .cs. При неизпълнение на тези изисква­ния **програмата пак ще работи,** **но** **ориентацията в кода ще е затруднена**. В нашия пример, тъй като класът се казва HelloCSharp, трябва да запишем неговия изходен (сорс) код във файл с име HelloCSharp.cs.

## Езикът C# и платформата .NET

Първата версия на C# е разработена от Microsoft в периода 1999-2002 г. и е пусната официално в употреба през 2002 година, като част от .NET платформата, която **има за цел да улесни съществено разработката на софтуер за Windows среда** чрез качествено нов подход към програми­рането, базиран на концепциите за "виртуална машина" и "управляван код". По това време езикът и платформата Java, изградени върху същите концепции, се радват на огромен успех във всички сфери на разработката на софтуер и разработката на C# и .NET е естественият отговор на Microsoft срещу успехите на Java технологията.

### Езикът C#

C# e съвременен обектно-ориентиран език за програмиране от високо ниво с общо предназначение. Синтаксисът му е подобен на C и C++, но не поддържа много от техните възможности с цел опростяване на езика, улесняване на програмирането и повишаване на сигурността.

Програмите на C# представляват **един или няколко файла с разширение** .cs, в които се съдържат дефиниции на класове и други типове. Тези фай­лове се **компилират** от компилатора на C# (csc) до изпълним код и в резултат се получават **асемблита** – файлове със същото име, но с раз­лично разширение (.exe или .dll). Например, ако компилираме файла HelloCSharp.cs, ще получим като резултат файл с име HelloCSharp.exe (както и други помощни файлове, които не са от значение за момента).

Компилираният код може да се изпълни както всяка друга програма от нашия компютър (с двойно щракване върху нея). Ако се опитаме да изпълним компилирания C# код (например програмата HelloCSharp.exe) на компютър, на който няма .NET Framework, ще получим съобщение за грешка.

#### Ключови думи

Езикът C# използва следните **ключови думи** за построяване на своите програмни конструкции:

|  |  |  |  |
| --- | --- | --- | --- |
| [**abstract**](http://msdn.microsoft.com/en-us/library/sf985hc5(VS.71).aspx) | [**event**](http://msdn.microsoft.com/en-us/library/8627sbea(VS.71).aspx) | [**new**](http://msdn.microsoft.com/en-us/library/51y09td4(VS.71).aspx) | [**struct**](http://msdn.microsoft.com/en-us/library/ah19swz4(VS.71).aspx) |
| [**as**](http://msdn.microsoft.com/en-us/library/cscsdfbt(VS.71).aspx) | [**explicit**](http://msdn.microsoft.com/en-us/library/xhbhezf4(VS.71).aspx) | [**null**](http://msdn.microsoft.com/en-us/library/edakx9da(VS.71).aspx) | [**switch**](http://msdn.microsoft.com/en-us/library/06tc147t(VS.71).aspx) |
| [**base**](http://msdn.microsoft.com/en-us/library/hfw7t1ce(VS.71).aspx) | [**extern**](http://msdn.microsoft.com/en-us/library/e59b22c5(VS.71).aspx) | [**object**](http://msdn.microsoft.com/en-us/library/9kkx3h3c(VS.71).aspx) | [**this**](http://msdn.microsoft.com/en-us/library/dk1507sz(VS.71).aspx) |
| [**bool**](http://msdn.microsoft.com/en-us/library/c8f5xwh7(VS.71).aspx) | [**false**](http://msdn.microsoft.com/en-us/library/67bxt5ee(VS.71).aspx) | [**operator**](http://msdn.microsoft.com/en-us/library/s53ehcz3(VS.71).aspx) | [**throw**](http://msdn.microsoft.com/en-us/library/1ah5wsex(VS.71).aspx) |
| [**break**](http://msdn.microsoft.com/en-us/library/adbctzc4(VS.71).aspx) | [**finally**](http://msdn.microsoft.com/en-us/library/zwc8s4fz(VS.71).aspx) | [**out**](http://msdn.microsoft.com/en-us/library/t3c3bfhx(VS.71).aspx) | [**true**](http://msdn.microsoft.com/en-us/library/eahhcxk2(VS.71).aspx) |
| [**byte**](http://msdn.microsoft.com/en-us/library/5bdb6693(VS.71).aspx) | [**fixed**](http://msdn.microsoft.com/en-us/library/f58wzh21(VS.71).aspx) | [**override**](http://msdn.microsoft.com/en-us/library/ebca9ah3(VS.71).aspx) | [**try**](http://msdn.microsoft.com/en-us/library/0yd65esw(VS.71).aspx) |
| [**case**](http://msdn.microsoft.com/en-us/library/06tc147t(VS.71).aspx) | [**float**](http://msdn.microsoft.com/en-us/library/b1e65aza(VS.71).aspx) | [**params**](http://msdn.microsoft.com/en-us/library/w5zay9db(VS.71).aspx) | [**typeof**](http://msdn.microsoft.com/en-us/library/58918ffs(VS.71).aspx) |
| [**catch**](http://msdn.microsoft.com/en-us/library/0yd65esw(VS.71).aspx) | [**for**](http://msdn.microsoft.com/en-us/library/ch45axte(VS.71).aspx) | [**private**](http://msdn.microsoft.com/en-us/library/st6sy9xe(VS.71).aspx) | [**uint**](http://msdn.microsoft.com/en-us/library/x0sksh43(VS.71).aspx) |
| [**char**](http://msdn.microsoft.com/en-us/library/x9h8tsay(VS.71).aspx) | [**foreach**](http://msdn.microsoft.com/en-us/library/ttw7t8t6(VS.71).aspx) | [**protected**](http://msdn.microsoft.com/en-us/library/bcd5672a(VS.71).aspx) | [**ulong**](http://msdn.microsoft.com/en-us/library/t98873t4(VS.71).aspx) |
| [**checked**](http://msdn.microsoft.com/en-us/library/74b4xzyw(VS.71).aspx) | [**goto**](http://msdn.microsoft.com/en-us/library/13940fs2(VS.71).aspx) | [**public**](http://msdn.microsoft.com/en-us/library/yzh058ae(VS.71).aspx) | [**unchecked**](http://msdn.microsoft.com/en-us/library/a569z7k8(VS.71).aspx) |
| [**class**](http://msdn.microsoft.com/en-us/library/0b0thckt(VS.71).aspx) | [**if**](http://msdn.microsoft.com/en-us/library/5011f09h(VS.71).aspx) | [**readonly**](http://msdn.microsoft.com/en-us/library/acdd6hb7(VS.71).aspx) | [**unsafe**](http://msdn.microsoft.com/en-us/library/chfa2zb8(VS.71).aspx) |
| [**const**](http://msdn.microsoft.com/en-us/library/e6w8fe1b(VS.71).aspx) | [**implicit**](http://msdn.microsoft.com/en-us/library/z5z9kes2(VS.71).aspx) | [**ref**](http://msdn.microsoft.com/en-us/library/14akc2c7(VS.71).aspx) | [**ushort**](http://msdn.microsoft.com/en-us/library/cbf1574z(VS.71).aspx) |
| [**continue**](http://msdn.microsoft.com/en-us/library/923ahwt1(VS.71).aspx) | [**in**](http://msdn.microsoft.com/en-us/library/ttw7t8t6(VS.71).aspx) | [**return**](http://msdn.microsoft.com/en-us/library/1h3swy84(VS.71).aspx) | [**using**](http://msdn.microsoft.com/en-us/library/zhdeatwt(VS.71).aspx) |
| [**decimal**](http://msdn.microsoft.com/en-us/library/364x0z75(VS.71).aspx) | [**int**](http://msdn.microsoft.com/en-us/library/5kzh1b5w(VS.71).aspx) | [**sbyte**](http://msdn.microsoft.com/en-us/library/d86he86x(VS.71).aspx) | [**virtual**](http://msdn.microsoft.com/en-us/library/9fkccyh4(VS.71).aspx) |
| [**default**](http://msdn.microsoft.com/en-us/library/06tc147t(VS.71).aspx) | [**interface**](http://msdn.microsoft.com/en-us/library/87d83y5b(VS.71).aspx) | [**sealed**](http://msdn.microsoft.com/en-us/library/88c54tsw(VS.71).aspx) | [**volatile**](http://msdn.microsoft.com/en-us/library/x13ttww7(VS.71).aspx) |
| [**delegate**](http://msdn.microsoft.com/en-us/library/900fyy8e(VS.71).aspx) | [**internal**](http://msdn.microsoft.com/en-us/library/7c5ka91b(VS.71).aspx) | [**short**](http://msdn.microsoft.com/en-us/library/ybs77ex4(VS.71).aspx) | [**void**](http://msdn.microsoft.com/en-us/library/yah0tteb(VS.71).aspx) |
| [**do**](http://msdn.microsoft.com/en-us/library/370s1zax(VS.71).aspx) | [**is**](http://msdn.microsoft.com/en-us/library/scekt9xw(VS.71).aspx) | [**sizeof**](http://msdn.microsoft.com/en-us/library/eahchzkf(VS.71).aspx) | [**while**](http://msdn.microsoft.com/en-us/library/2aeyhxcd(VS.71).aspx) |
| [**double**](http://msdn.microsoft.com/en-us/library/678hzkk9(VS.71).aspx) | [**lock**](http://msdn.microsoft.com/en-us/library/c5kehkcz(VS.71).aspx) | [**stackalloc**](http://msdn.microsoft.com/en-us/library/cx9s2sy4(VS.71).aspx) |  |
| [**else**](http://msdn.microsoft.com/en-us/library/5011f09h(VS.71).aspx) | [**long**](http://msdn.microsoft.com/en-us/library/ctetwysk(VS.71).aspx) | [**static**](http://msdn.microsoft.com/en-us/library/98f28cdx(VS.71).aspx) |  |
| [**enum**](http://msdn.microsoft.com/en-us/library/sbbt4032(VS.71).aspx) | [**namespace**](http://msdn.microsoft.com/en-us/library/z2kcy19k(VS.71).aspx) | [**string**](http://msdn.microsoft.com/en-us/library/362314fe(VS.71).aspx) |  |

Не всички ключови думи съществуват от създаването на езика. Някои от тях са добавени в по-късните версии. Основни конструкции в C# (които се дефинират и използват с помощта на клю­човите думи) са класовете, методите, операторите, изразите, услов­ните конструкции, циклите, типовете данни и изключенията.

Всички тези конструкции, както и употребата на повечето ключови думи от горната таблица, предстои да бъдат разгле­дани подробно в следващите глави на настоящата книга.

#### Автоматично управление на паметта

Едно от най-големите предимства на .NET Framework е вграденото автома­тично управление на паметта. То **предпазва** програмистите от сложната задача **сами да заделят памет за обектите** и да търсят подхо­дящия момент за нейното освобождаване. Това сериозно **повишава произ­водителността** на програмистите и увеличава качеството на програмите, писани на C#.

За управлението на паметта в .NET Framework се грижи специален компонент от CLR, наречен "**събирач на боклука**" или "**система за почистване на паметта**" (**garbage collector**). Основните задачи на събирача на боклука са да следи кога заделената памет за променливи и обекти вече не се използва, да я освобождава и да я прави достъпна за последващи заделяния на нови обекти.

|  |  |
| --- | --- |
|  | Важно е да се знае, че не е сигурно в точно кой момент паметта се изчиства от неизползваните обекти (например от локалните променливи). В спецификациите на езика C# е описано, че това става след като дадената променлива излезе от обхват, но не е посочено дали веднага или след изминаване на някакво време или при нужда от памет. |

### Независимост от средата и от езика за програмиране

Едно от предимствата на .NET е, че програмистите, пишещи на различни .NET езици за програмиране, могат да обменят кода си безпроблемно. Например C# програмист може да използва кода на програмист, написан на VB.NET, Managed C++ или F#. Това е възможно, тъй като програмите на различните .NET езици ползват обща система от типове данни и обща инфраструктура за изпълнение, както и единен формат на компилирания код (асемблита).

Като голямо предимство на .NET технологията се счита възможността веднъж написан и компи­лиран код да се изпълнява на различни операци­онни системи и хар­дуерни устройства. Можем да компилираме C# програ­ма в Windows среда и да я изпълняваме както върху Windows така и върху Windows Mobile или Linux. Официално Microsoft поддържат .NET Framework само за Windows, Windows Mobile и Windows Phone платформи, но трети доставчици предлагат .NET импле­ментации за други операционни системи. Например проектът с отворен код Mono ([www.mono-project.com](http://www.mono-project.com)) импле­ментира основната част от .NET Framework заедно с всички прилежащи библиотеки за Linux.

### Common Intermediate Language (CIL)

Идеята за независимост от средата е заложена още при самото създаване на .NET платформата и се реализира с малка хитрина**. Изходният код не се компилира до инструкции**, предназначени за даден конкретен микропро­цесор, и не използва специфични възможности на дадена опера­ционна система, а се компилира до междинен език – така нареченият Common Intermediate Language (CIL). Този език CIL **не се изпълнява директно от микро­процесора**, а се изпълнява от виртуална среда за изпълнения на CIL кода, наречена **Common Language Runtime (CLR)**.

### Common Language Runtime (CLR) – сърцето на .NET

В самия център на .NET платформата работи нейното сърце – Common Language Runtime (CLR) – средата за контролирано изпълнение на управляван код (CIL код). Тя **осигурява изпълнение** на .NET програми върху **различни хардуерни платформи и операционни системи**.

CLR е абстрактна изчислителна машина (виртуална машина). По аналогия на реалните електронноизчислителни машини тя поддържа набор от инструкции, регистри, достъп до паметта и входно-изходни операции. CLR осигурява контролирано изпълнение на .NET програмите, използвайки в пълнота възможностите на процесора и операционната система. CLR осъществява контролиран достъп до паметта и другите ресурси на маши­ната, като съобра­зява правата за достъп, зададени при изпълнението на програмата.

### .NET платформата

.NET платформата, освен езика C#, съдържа в себе си CLR и множество помощни инструменти и библиотеки с готова функционалност. Същест­вуват няколко нейни разно­видности съобразно целевата потре­бителска група:

* **.NET Framework** е най-използвания вариант на .NET среда, понеже тя е с общо, масово предназначение. Използва се при разработката на конзолни прило­жения, Windows програми с графичен интерфейс, уеб приложения и много други.
* **.NET Compact Framework** (CF) е "олекотена" версия на стандарт­ния .NET Framework и се използва за разработка на приложения за мобилни телефони и други PDA устройства (използващи Windows Mobile Edition).
* **Silverlight** също е "олекотена" версия на .NET Framework, предназ­на­чена да се изпълнява в уеб браузърите за реализация на мултиме­дийни и RIA приложения (Rich Internet Applications).

#### .NET Framework

Стандартната версия на .NET платформата е предназначена за разра­ботката и използването на конзолни приложения, настолни приложения, уеб приложения, уеб услуги, RIA приложения и още много други. Почти всички .NET програмисти използват стандартната версия.

### .NET технологиите

Въпреки своята големина и изчерпателност .NET платформата не дава инструменти за решаването на всички задачи от разработката на софтуер. Съществуват множество независими производи­тели на софтуер, които разширяват и допълват стандартната функционалност, която предлага .NET Framework. Например фирми като българската софтуерна корпо­рация Telerik разработват **допълнителни набори от компоненти** за създа­ване на графичен потребителски интерфейс, средства за управление на уеб съдържание, библиотеки и инструменти за изготвяне на отчети и други инструменти за улесняване на разработката на приложения.

Разши­ренията, предлагани за .NET Framework, са програмни компоненти, достъпни за преизползване при писането на .NET програми. Преизползването на програмен код същест­вено **улеснява и опростява разработката на софтуер**, тъй като решава често срещани проблеми и предоставя наготово сложни алгоритми, импле­ментации на технологични стандарти и др. Съвременният програмист ежедневно използва готови библиотеки и така си спестява огромна част от усилията.

Да вземем за пример писането на програма, която визуализира данни под формата на графики и диаграми. Можем да вземем библиотека написана за .NET, която рисува самите графики. Всичко, от което се нуждаем, е да **подадем правилните входни данни** и библиотеката ще **изрисува графиките** вместо нас. Много е удобно и ефективно. Освен това води до **понижаване на разходите за разработка**, понеже програмистите няма да отделят време за разработване на допълнителната функционалност (в нашия случай самото чертаене на графиките, което е свързано със сложни математи­чески изчисления и управление на видеокартата). Самото приложение също ще бъде с **по-високо качество**, понеже разширението, което се използва в него, е разработвано и поддържано от специалисти, които имат доста повече опит в тази специфична област.

Повечето разширения се използват като инструменти, защото са сравни­телно прости. Съществуват и разширения, които представляват съвкуп­ност от средства, библиотеки и инструменти за разработка, които имат сложна структура и вътрешни зависимости и е по-коректно да се нарекат **софтуерни технологии**. Съще­ствуват множе­ство .NET технологии с различни области на приложение. Типични примери са уеб технологиите (ASP.NET), позволя­ващи бързо и лесно да се пишат динамични уеб приложения и XAML технологиите, които позволяват да се пишат мултиме­дийни приложения с богат потребителски интерфейс.

.NET Framework стандартно включва в себе си множество технологии и **библиотеки от класове** (class libraries) със стандартна функционалност, която програмистите ползват наготово в своите приложения. Например в системната библиотека има класове за работа с математически функции, изчисляване на логаритми и тригонометрични функции, които могат да се ползват наготово (класът System.Math). Друг пример е библиотеката за работа с мрежа (System.Net), която има готова функционалност за изпращане на e-mail (чрез класа System.Net.Mail.MailMessage) и за изтегляне на файл от Интернет (чрез класа System.Net.WebClient).

**.NET технология** наричаме съвкупността от .NET класове, библиотеки, инстру­менти, стандарти и други програмни средства и утвърдени подходи за разработка, които установяват технологична рамка при изграждането на определен тип приложения. **.NET библиотека** наричаме съвкупност от .NET класове, които предоставят наготово определен тип функционалност. Например .NET технология е ADO.NET, която предоставя стандартен подход за достъп до релационни бази от данни (като например Microsoft SQL Server и MySQL). Пример за библиотека са класовете от пакета System.Data.SqlClient, които предоставят връзка до SQL Server посредством технологията ADO.NET.

Някои технологии, разработвани от външни софтуерни доставчици, с времето започват да се използват масово и се утвърж­дават като техно­логични стандарти. Част от тях биват забелязани от Microsoft и биват включвани като разширения в следващите версии на .NET Framework. Така .NET платформата постоянно еволюира и се разширява с нови библиотеки и технологии. Например технологиите за обектно-релационна персис­тентност на данни (ORM технологиите) първоначално започнаха да се развиват като независими проекти и продукти (като проекта с отворен код NHibernate), а по-късно набраха огромна популярност и доведоха до нуждата от вграждането им в .NET Framework. Така се родиха технологиите LINQ-to-SQL и ADO.NET Entity Framework съответно в .NET 3.5 и .NET 4.0.

#### Application Programming Interface (API)

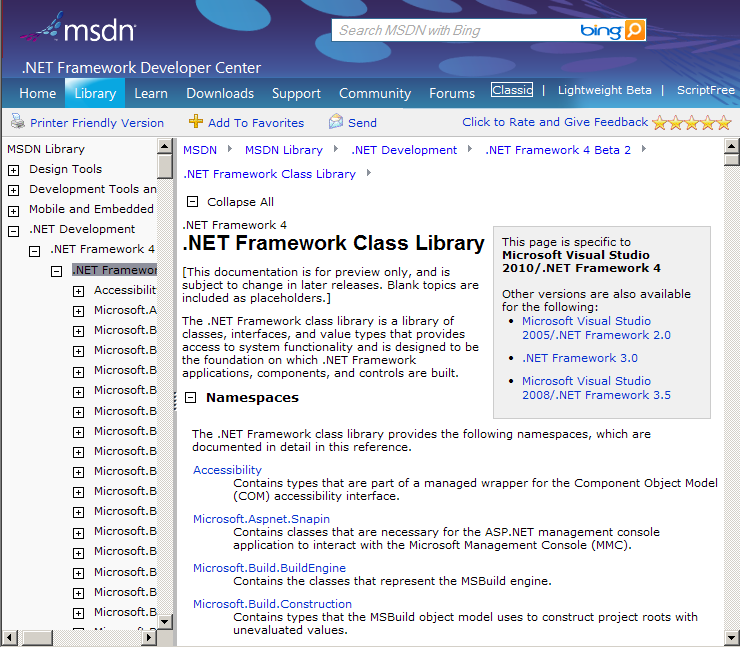
Всеки .NET инструмент или технология се използва, като се създават обекти и се викат техни методи. Наборът от публични класове и методи, които са достъпни за употреба от програмистите и се предоставят от технологиите, се нарича **Application Programming Interface** или просто **API**. За пример можем да дадем самия .NET API, който е набор от .NET библиотеки с класове, разширяващи възможностите на езика, добавяйки функционал­ност от високо ниво. Всички .NET технологии предоставят публичен API. Много често за самите технологии се говори просто като за API, предоста­вящ определена функционалност, като например API за работа с файлове, API за работа с графика, API за работа с принтер, уеб API и т.н. Голяма част от съвременния софтуер използва множество видове API, обособени като отделно ниво в софтуерните приложения.

#### .NET документацията

Много често се налага да се документира един API, защото той съдържа множество пространства от имена и класове. Класовете съдържат методи и параметри, смисълът на които не винаги е очевиден и трябва да бъде обяснен. Съществуват вътрешни зависимости между отделните класове и за правилната им употреба са необходими разяснения. Такива разяснения и технически инструкции за използване на дадена технология, библио­тека или API и се наричат **документация**. Документацията представлява съвкупност от документи с техническо съдържание.

.NET Framework също има документация, разработвана и поддържана официално от Майкрософт. Тя е достъпна публично в Интернет и се разпространява заедно с .NET платформата като съвкупност от документи и инструменти за преглеждането им и за търсене.

Библиотеката **MSDN Library** представлява официалната документация на Microsoft за всички техни продукти за разработчици и софтуерни технологии. В частност документацията за .NET Framework е част от MSDN Library и е достъпна в Интернет на адрес: <http://msdn.microsoft.com/en-us/library/ms229335(VS.100).aspx>. Ето как изглежда тя:



### Какво ви трябва, за да програмирате на C#?

След като разгледахме какво представляват .NET платформата, .NET библиотеките и .NET технологиите, можем да преминем към писането, компилирането и изпълнението на C# програми.

Базовите изисквания, за да можете да програмирате на C# са **инсталиран .NET Framework и текстов редактор**. Текстовият редактор служи за създаване и редактиране на C# кода, а за компи­лиране и изпълнение се нуждаем от самия .NET Framework.

#### Текстов редактор

Текстовият редактор служи за писане на изходния код на програмата и за записването му във файл. След това кодът се компилира и изпълнява. Като текстов редактор можете да използвате вградения в Windows редактор Notepad (който е изключително примитивен и неудобен за работа) или да си изтеглите по-добър безплатен редактор като например някой от редакторите Notepad++ (<http://notepad-plus.sourceforge.net>) или PSPad ([www.pspad.com](http://www.pspad.com)).

### Компилация и изпълнение на C# програми

Дойде време да компилираме и изпълним вече разгледания теоретичен пример на проста програма, написана на C#. За целта трябва да направим следното:

* Да създадем файл с име HelloCSharp.cs;
* Да запишем примерната програма във файла;
* Да компилираме HelloCSharp.cs до файл HelloCSharp.exe;
* Да изпълним файла HelloCSharp.exe.

А сега, нека да го направим на компютъра!

|  |  |
| --- | --- |
|  | Не забравяйте преди започването с примера, да инста­лирате .NET Framework на компютъра си! В противен случай няма да можете да компилирате и да изпълните програмата. |

По принцип .NET Framework се инсталира заедно с Windows, но в някои ситуации все пак би могъл да липсва. За да инсталирате .NET Framework на компютър, на който той не е инсталиран, трябва да го изтеглите от сайта на Майкрософт (<http://download.microsoft.com>). Ако не знаете коя версия да изтеглите, **изберете последната версия**.

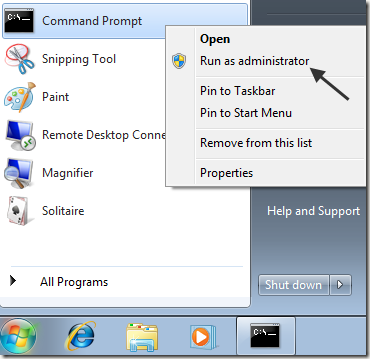
Горните стъпки варират на различните операционни системи. Тъй като програмирането под Linux е малко встрани от фокуса на настоящата книга, ще разгледаме подробно какво ви е необходимо, за да напишете и изпълните примерната програма под Windows. За тези от вас които искат да се опитат да програмират на C# в Linux среда ще споменем необхо­димите инструменти и те ще имат възможност да си ги изтеглят и да експери­ментират.

Ето го и кодът на нашата първа C# програма:

|  |
| --- |
| HelloCSharp.cs |
| class HelloCSharp  {  static void Main()  {  System.Console.WriteLine("Hello, C#!");  }  } |

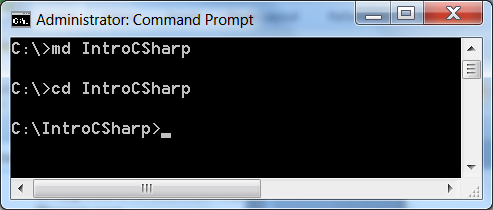
#### Компилиране на C# програми под Windows

Първо стартираме конзолата за команди на Windows, известна още като **Command Prompt** (това става от главното меню на Windows Explorer – Start -> Programs -> Accessories -> Command Prompt):



За предпочитане е в конзолата да се работи с администраторски права, тъй като при липсата им някои операции не са позволени. Стартирането на **Command Prompt** с администраторски права става от контекстното меню, което се появява при натискане на десния бутон на мишката върху иконката на **Command Prompt** (вж. картинката).

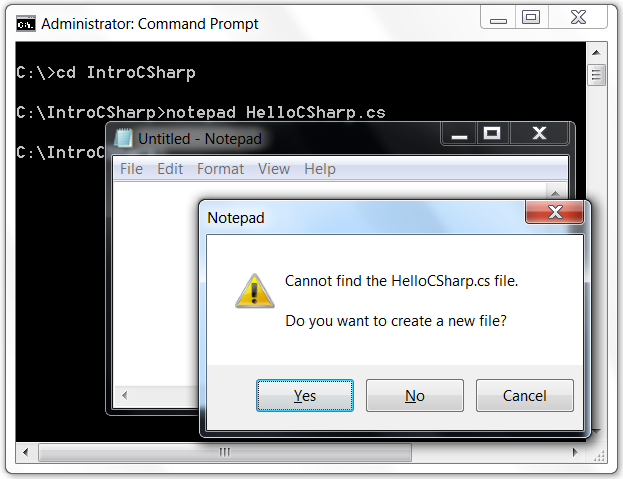
Нека след това от конзолата създадем директория, в която ще експери­ментираме. Използваме командата md за създаване на директория и командата cd за влизане в нея:



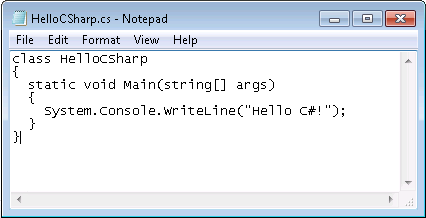
Директорията се казва IntroCSharp и се намира в C:\. Променяме текущата директория на C:\IntroCSharp и създаваме нов файл HelloCSharp.cs**,** като за целта използваме вградения в Windows текстов редактор Notepad. За да създадем файла, на конзолата изписваме следната команда:

|  |
| --- |
| **notepad HelloCSharp.cs** |

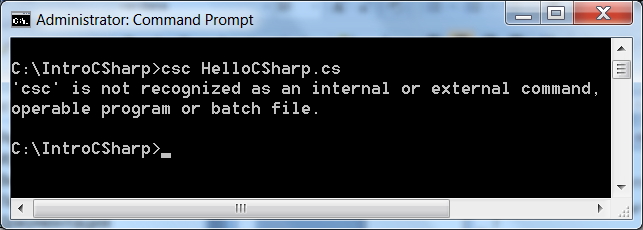
Това стартира Notepad и той показва следния диалогов прозорец за създаване на несъществуващ файл:



Notepad ни пита дали искаме да бъде създаден нов файл, защото такъв в момента липсва. Отговаряме с "Yes". Следващата стъпка е да препишем програмата или просто да прехвърлим текста чрез копиране (Copy / Paste):



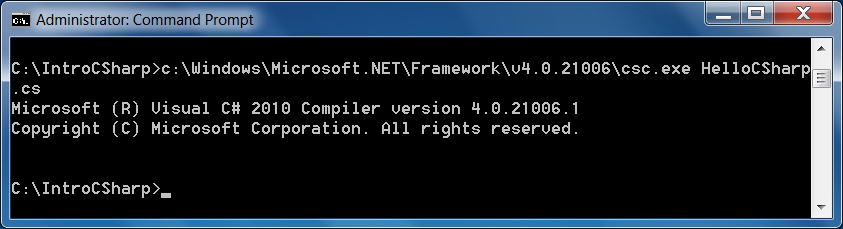
Записваме чрез [Ctrl-S] и затваряме редактора Notepad с [Alt-F4]. Вече имаме изходния код на нашата примерна C# програма, записан във файла C:\IntroCSharp\HelloCSharp.cs. Остава да го компилираме и изпълним. Компилацията се извършва с компи­латора csc.exe.



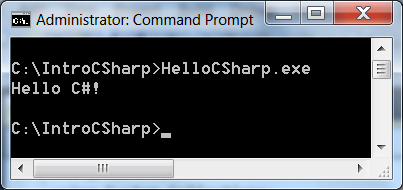
Ето, че получихме грешка – Windows не може да намери изпълним файл или вградена команда с име csc. Това е често срещан проблем и е нормално да се появи, ако сега започваме да работим с C#. Причините за него може да са следните:

* Не е инсталиран .NET Framework;
* Инсталиран е .NET Framework, но директорията Microsoft.NET\ Framework\v4.0 не е в пътя за търсене на изпълними файлове и Windows не намира csc.exe, въпреки че той е наличен на диска.

Първият проблем се решава като се инсталира .NET Framework (в нашия случай – версия 4.0). Другият проблем може да се реши с промени в системния път (ще го направим след малко) или чрез използване на пълния път до csc.exe, както е показано на картинката долу. В нашия случай пълният път до C# компилатора на нашия твърд диск е следният: c:\Windows\Microsoft.NET\Framework\v4.0.21006\csc.exe. Да извикаме компилатора и да му подадем като параметър файла, който трябва да бъде компилиран (HelloCSharp.cs):



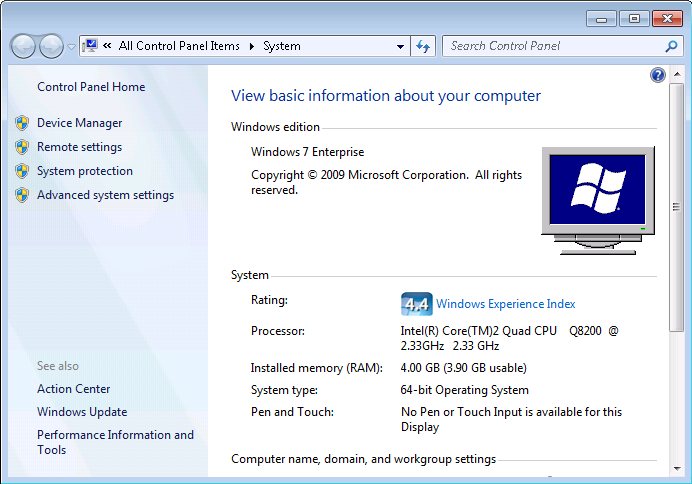
След изпълнението си csc приключва без грешки, като произвежда в резултат файла C:\IntroCSharp\HelloCSharp.exe. За да го изпълним, просто трябва да изпишем името му. Резултатът от изпълнението на нашата първа програма е съобщението "Hello, C#!", отпечатано на конзолата. Не е нещо велико, но е едно добро начало:



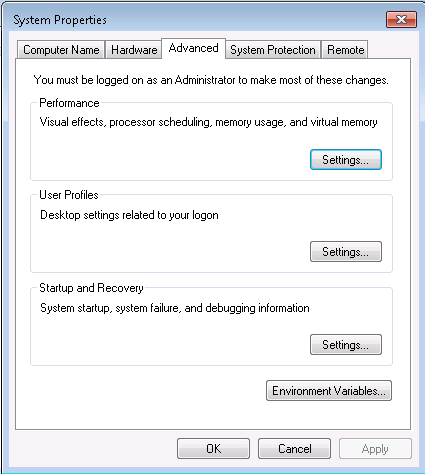
#### Промяна на системните пътища в Windows

Може би ви е досадно всеки път да изписвате пълния път до csc.exe, когато компилирате C# програми през конзолата. За да избегнете това, можете да редактирате системните пътища в Windows и след това да затворите конзолата и да я пуснете отново. Промяната на системните пътища в Windows става по следния начин:

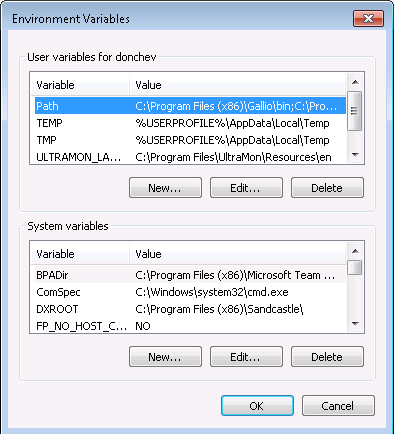
1. Отиваме в контролния панел и избираме "System". Появява се следният добре познат прозорец:



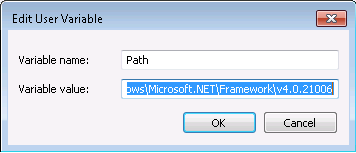
1. Избираме "Advanced System Settings". Появява се диалоговият прозорец "System Properties":



1. Натискаме бутона "Environment Variables" и се показва прозорец с всички променливи на средата:



1. Избираме "Path" от списъка с променливите, както е показано на горната картинка и натискаме бутона "Edit". Появява се малко прозорче, в което добавяме пътя до директорията, където е инсталиран .NET Framework:



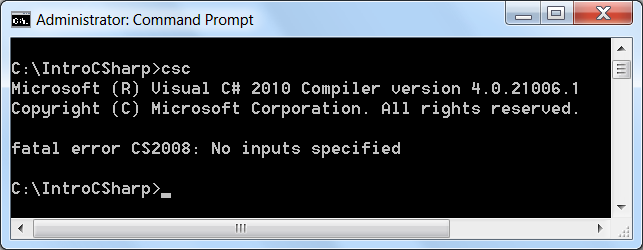
Разбира се, първо трябва да намерим къде е инсталиран .NET Framework. Стандартно той се намира някъде в директорията C:\Windows\Microsoft.NET, например в следната директория:

|  |
| --- |
| C:\Windows\Microsoft.NET\Framework\v.4.0.21006 |

Добавянето на допълнителен път към вече съществуващите пътища от променливата на средата Path се извършва като новият път се долепи до съществуващите и за раздели­тел между тях се използва точка и запетая (;).

|  |  |
| --- | --- |
|  | Бъдете внимателни, защото ако изтриете някой от вече същест­вуващите системни пътища, определени функции в Windows или част от инсталирания софтуер могат да спрат да функцио­нират нормално! |

1. Когато сме готови с пътя, можем да се опитаме да извикаме csc.exe, без да посочваме пълния път до него. За целта отваряме cmd.exe (Command Prompt) и пишем командата "csc". Би трябвало да се изпише версията на C# компилатора и съобщение, че не е зададен входен файл:



## Средата за разработка Visual Studio

До момента разгледахме как се компилират и изпълняват C# програми през конзолата (Command Prompt). Разбира се, има и по-лесен начин – чрез използване на **интегрирана среда за разработка**, която може да изпълнява вместо нас всички команди, които използвахме. Нека разгле­даме как се работи със среди за разработка и с какво ни помагат те, за да си вършим по-лесно работата.

### Интегрирани среди за разработка

В предходните примери разгледахме компилация и изпълнение на прог­рама от един единствен файл. Обикновено програмите са съставени от много файлове, понякога дори десетки хиляди. Писането с текстов редактор, компилирането и изпълнението на една програма от командния ред е сравнително проста работа, но да направим това за голям проект, може да се окаже сложно и трудоемко занимание. За намаляване на сложността, улесняване на писането, компилирането и изпълнението на софтуерни приложения чрез един единствен инструмент съществуват визуални приложения, наречени **интегрирани среди за разработка** (Integrated Development Environment, IDE). Средите за разработка най-често пред­лагат множество допълнения към основните функции за разработка, като например дебъгване, изпълнение на unit тестове, проверка за често срещани грешки, достъп до хранилище за контрол на версиите и други.

### Какво е Visual Studio?

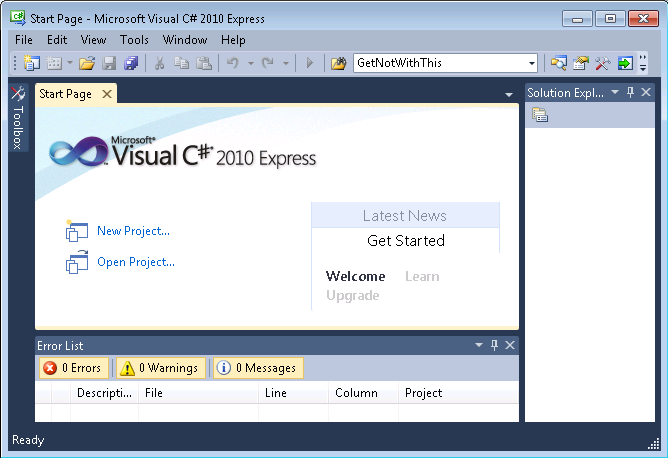
**Visual Studio** (VS) е мощна интегрирана среда за разработка на софту­ерни приложения за Windows и за платформата .NET Framework. VS поддържа различни езици за програмиране (например C#, VB.NET и C++) и различни технологии за разработка на софтуер (Win32, COM, ASP.NET, ADO.NET Entity Framework, Windows Forms, WPF, Silverlight и още десетки други Windows и .NET технологии). Visual Studio предоставя мощна интегрирана среда за писане на код, компилиране, изпълнение, дебъгване и тестване на приложения, дизайн на потребителски интер­фейс (форми, диалози, уеб страници, визуални контроли и други), моделиране на данни, модели­ране на класове, изпълнение на тестове, пакетиране на приложения и стотици други функции.

Visual Studio има безплатна версия наречена **Visual Studio Community Edition**, която може да се изтегли безплатно от неговия официален сайт: <https://www.visualstudio.com>.

В рамките на настоящата книга ще раз­гледаме само най-важните функции на Visual Studio – свързаните със самото програмиране. Това са функциите за създаване, редактиране, компили­ране, изпълнение и дебъгване на програми.

Преди да преминем към примера, нека разгледаме малко по-подробно структурата на визуалния интерфейс на **Visual Studio**. Основна съставна част са прозорците. Всеки прозорец реализира различна функция, свързана с разработката на приложения. Да разгле­даме как изглежда **Visual Studio** след начална инсталация и конфигурация по подразбиране. То съдържа няколко прозореца:

* **Start Page** – от началната страница можете лесно да отворите някой от последните си проекти или да стартирате нов, да направите първата си C# програма, да получите помощ за използването на C#.
* **Solution Explorer** – при незареден проект този прозорец е празен, но той ще стане част от живота ви като C# програмист. В него ще се показва структурата на проекта ви – всички файлове, от които се състои, независимо дали те са C# код, картинки, които ползвате, или някакъв друг вид код или ресурси.



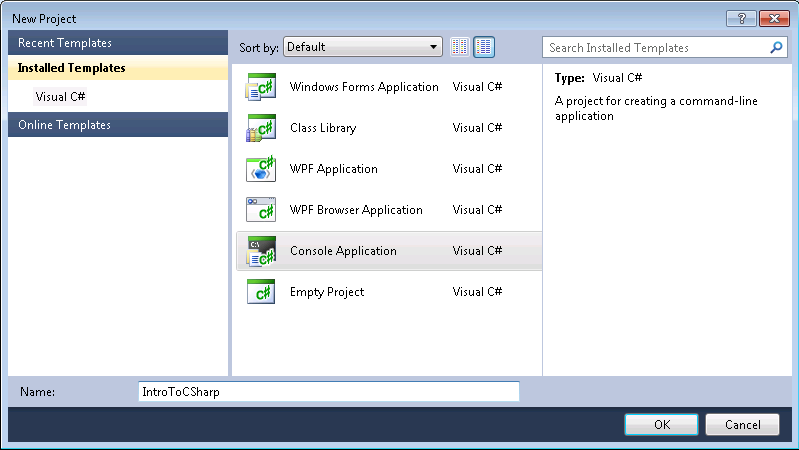
Съществуват още много други прозорци във Visual Studio с помощно предназ­начение, които няма да разглеждаме в момента.

### Създаване на нов C# проект

Преди да направим каквото и да е във Visual Studio, трябва да създадем нов проект или да заредим съществуващ. Проектът логически групира множество файлове, предназначени да реализират някакво софтуерно приложение или система. За всяка програма е препоръчително да се създава отделен проект.

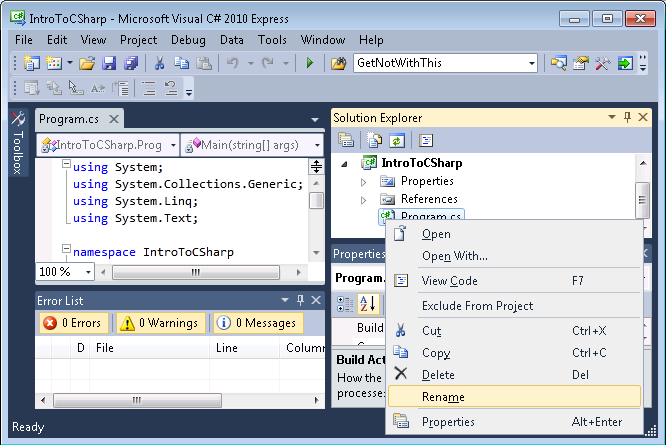
Проект във Visual Studio се създава чрез следване на следните стъпки:

* File -> New Project ...
* Появява се помощникът за нови проекти и в него са изброени типовете проекти, които можем да създадем. Имайте предвид, че понеже използваме **безплатна** **версия** на Visual Studio, предназна­чена главно за учащи, ще видим доста **по-малко видове проекти**, отколкото в стандартните, платени версии на VS:

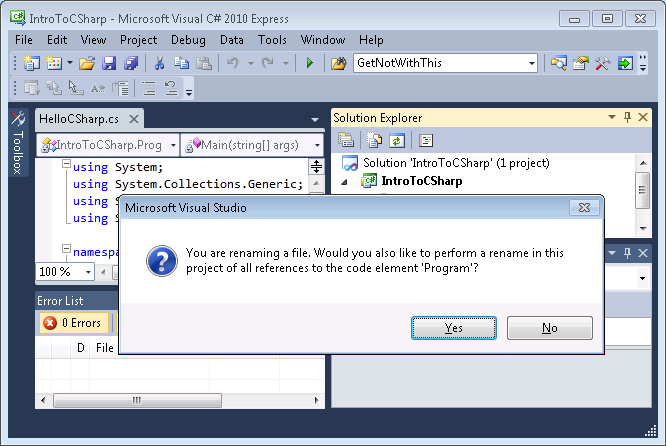


* Избираме **Console Application**. Конзолните приложения са прог­рами, които ползват за вход и изход конзолата. Когато е необходимо да се въведат данни, те се въвеждат от клавиатурата, а когато се отпечатва нещо, то се появява в конзолата (т.е. като текст на екрана в прозореца на програмата). Освен конзолни приложенията могат да бъдат с графичен потребителски интерфейс (GUI), уеб приложения, уеб услуги, мобилни приложения, RIA приложения и други.
* В полето "Name" пишем името на проекта. В нашия случай избираме име IntroToCSharp.
* Натискаме бутона [OK].

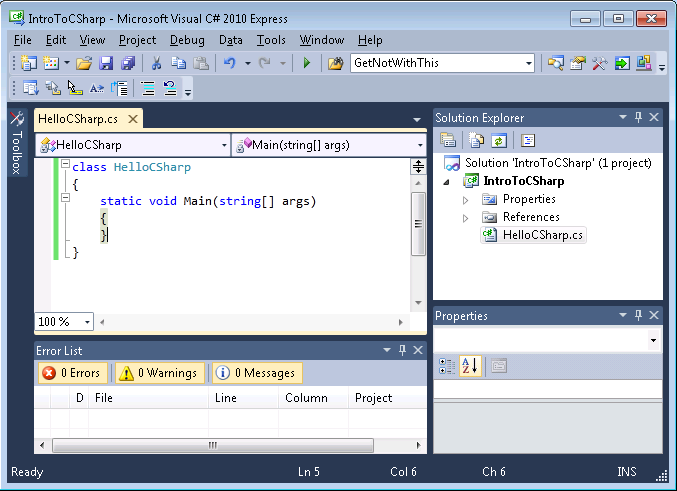
Новосъздаденият проект се показва в Solution Explorer. Автоматично е добавен и първият ни файл, съдържащ кода на програмата. Той носи името Program.cs. Важно е да задаваме смислени имена на нашите файлове, класове, методи и други елементи от програмата, за да можем след това лесно да ги намираме и да се ориентираме с кода. За да преименуваме файла Program.cs, щракваме с десен бутон върху него в Solution Explorer и избираме "Rename". Може да зададем за име на основния файл от нашата C# програма HelloCSharp.cs. Преименува­нето на файл можем да изпълним и с клавиша [F2], когато е избран съответния файл от Solution Explorer:



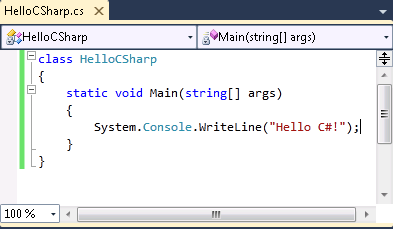
Появява се диалогов прозорец, който ни пита дали искаме освен името на файла да преименуваме и името на класа. Избираме "Yes".



След като изпълним горните стъпки вече имаме първото конзолно приложение носещо името IntroToCSharp и съдържащо един единствен клас HelloCSharp:



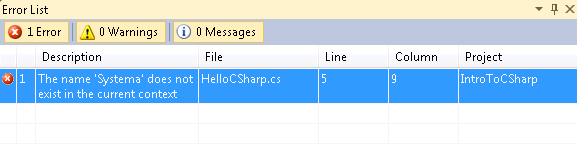
Остава да допълним кода на метода Main(). По подразбиране кодът на HelloCSharp.cs би трябвало да е зареден за редактиране. Ако не е, щракваме два пъти върху файла HelloCSharp.cs в Solution Explorer прозореца, за да го заредим. Попълваме сорс кода:



### Компилиране на сорс кода

Процесът на компилация във Visual Studio включва няколко стъпки:

* Проверка за синтактични грешки;



* Проверка за други грешки, например липсващи библиотеки;
* Преобразуване на C# кода в изпълним файл (.NET асембли). При конзолни приложения се получава .exe файл.

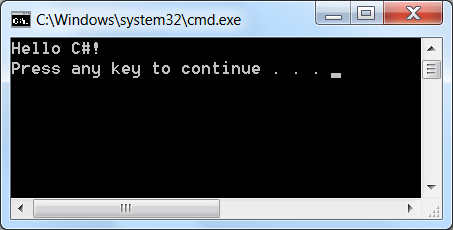
За да компилираме нашия примерен файл във Visual Studio, натискаме клавиша [F6]. Обикновено още докато пишем и най-късно при компи­лация наме­рените грешки се подчертават в червено, за да привличат вниманието на програмиста, и се показват във визуализатора "Error List" (ако сте го изключили можете да го покажете от менюто "View" на Visual Studio).

Ако в проекта ни има поне една грешка, то тя се отбелязва с малък червен "х" в прозореца "Error List". За всяка грешка се визуализира кратко описание на проблема, име на файл, номер на ред и име на проект. Ако щракнем два пъти върху някоя от грешките в "Error List", Visual Studio ни прехвърля автоматично в съответния файл и на съответния ред в кода, на мястото в кода, където е възникнала грешката.

### Стартиране на проекта

За да стартираме проекта натискаме [Ctrl+F5] (задържаме клавиша [Ctrl] натиснат и в това време натискаме клавиша [F5]).

Програмата се стартира и резултатът се изписва в конзолата, следван от текста "Press any key to continue . . .":



Последното съобщение не е част от резултата, произведен от програмата, а се показва от Visual Studio с цел да ни подсети, че програмата е завършила изпълнението си и да ни даде време да видим резултата. Ако стартираме програмата само с [F5], въпросното съобщение **няма да се появи** и резултатът ще изчезне веднага след като се е появил, защото **програмата ще приключи** и нейният прозорец ще бъде затворен. Затова използвайте [Ctrl+F5] за да стартирате своите конзолни програми.

Не всички типове проекти могат да се изпълняват. За да се изпълни C# проект, е необходимо той да съдържа точно един клас с Main() метод, деклариран по начина, описан [в началото на настоящата тема](#_Дефиниция_на_метод).

### Дебъгване на програмата

Когато програмата ни съдържа грешки, известни още като **бъгове**, трябва да ги намерим и отстраним, т.е. да **дебъгнем** програмата. Процесът на дебъгване включва:

* Забелязване на проблемите (бъговете);
* Намиране на кода, който причинява проблемите;
* Оправяне на кода, така че програмата да работи правилно;
* Тестване, за да се убедим, че програмата работи правилно след нанесените корекции.

Процесът може да се повтори няколко пъти, докато програмата заработи правилно.

След като сме забелязали проблем в програмата трябва да намерим кода, който го причинява. Visual Studio може да ни помогне с това, като ни позволи да проверим постъпково дали всичко работи, както е планирано.

За да спрем изпълнението на програмата на някакви определени места, можем да поставяме **точки на прекъсване**, известни още като **стопери (breakpoints)**. Стоперът е асоцииран към ред от програмата. Програ­мата спира изпълнението си на тези редове, където има стопер и позволява постъп­ково изпълнение на останалите редове. На всяка стъпка може да проверяваме и дори да променяме стойностите на текущите променливи.

Дебъгването е един вид **постъпково изпълнение на програмата** на забавен кадър. То ни дава възможност по-лесно да вникнем в детайлите и да видим къде точно възникват грешките и каква е причината за тях.

Нека направим **грешка в нашата програма** умишлено, за да видим как можем да се възползваме от стоперите (breakpoints). Ще добавим един ред в прог­рамата, който ще предизвика изключение (exception) по време на изпълнение (на изклю­ченията ще се спрем подробно в главата "[Обработка на изключения](#_Глава_12._Обработка)".

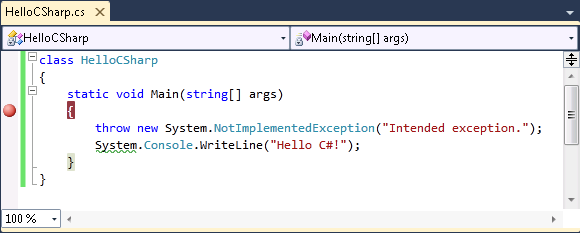
Засега нека направим програмата да изглежда по следния начин:

|  |
| --- |
| HelloCSharp.cs |
| class HelloCSharp  {  static void Main()  {  throw new NotImplementedException("Intended exception.");  System.Console.WriteLine("Hello, C#!");  }  } |

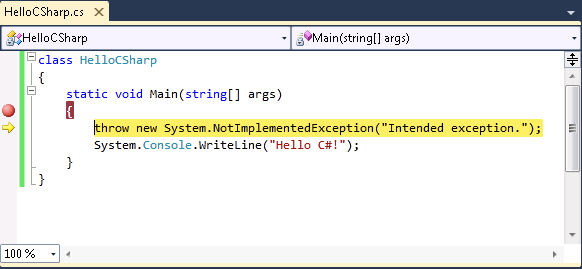
Когато отново стартираме програмата с [Ctrl+F5], ще получим грешка и тя ще бъде отпечатана в конзолата:



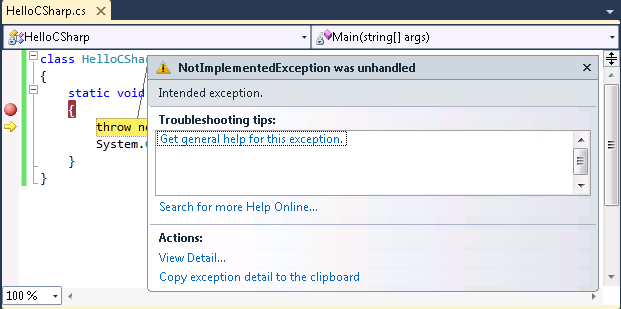
Да видим как стоперите ще ни помогнат да намерим къде е проблемът. Преместваме курсора на реда, на който е отварящата скоба на метода, и натискаме [F9] (така поставяме стопер на избрания ред). Появява се точка на прекъсване, където програмата ще спре изпълнението си, ако е стартирана в режим на дебъгване:



Сега трябва да стартираме програмата в режим на отстраняване на грешки (в режим на дебъгване). Избираме Debug -> Start Debugging или натискаме [F5]. Програмата се стартира и веднага след това спира на първата точка на прекъсване, която срещне. Кодът се оцветява в жълто и можем да го изпълняваме постъпково. С клавиша **[F10]** преминаваме на следващия ред:



Когато сме на даден ред и той е жълт, неговият код **все още не е изпълнен**. Изпълнява се след като го подминем. В случая все още не сме получили грешка, въпреки че сме на реда, който добавихме и който би трябвало да я предизвиква. Натискаме **[F10]** още веднъж, за да се изпълни текущият ред. Този път Visual Studio показва прозорец, който сочи реда, където е възникнала грешката, както и някои допълни­телни детайли за нея:



След като вече знаем точно къде е проблемът в програмата, можем да го отстраним. За да стане това трябва първо да **спрем изпълнението** на програмата преди да е завършила. Избираме Debug –> Stop Debugging или натискаме [Shift + F5]. След това изтриваме проблемния ред и стартираме програмата в нормален режим (без проследяване) с [Ctrl+F5].

## Алтернативи на Visual Studio

Както вече видяхме, въпреки че можем да минем и без Visual Studio на теория, това на практика **не е добра идея**. Работата по компилиране на един голям проект, отстраняването на грешки в кода и много други действия биха отнели много време извън Visual Studio.

От друга страна Visual Studio е платена среда за разработка на софтуер (в пълната му версия). Много хора трудно могат да си позволят да си закупят професионалните му версии (дори това може да е непосилно за малки фирми и отделни лица, които се занимават с програмиране). За щастие Community версията на Visual Studio е безплатна и е достатъчно добра.

### SharpDevelop

Една от тях е SharpDevelop (#Develop). Можете да го намерите на следния сайт: [http://www.icsharpcode.NET/OpenSource/SD/](http://www.icsharpcode.net/OpenSource/SD/). #Develop е IDE за C# и се разработва като софтуер с отворен код. Той поддържа голяма част от функционалностите на Visual Studio 2010, но работи и под Linux и други операционни системи. Няма да го разглеждаме подробно, но го имайте предвид в случай, че ви е необходима среда за C# разработка и не можете да ползвате Visual Studio.

### MonoDevelop

MonoDevelop е интегрирана среда за разработка на софтуер за .NET платформата. Той е напълно безплатен (с отворен код) и може да бъде свален от: [http://monodevelop.com](http://monodevelop.com/). С MonoDevelop могат бързо и лесно да се пишат напълно функционални десктоп, мобилни и уеб приложения за Linux, Mac OS X и Windows. С него програмистите могат лесно да прехвърлят проекти, създадени с Visual Studio, към Mono платформата и да ги направят напълно функциониращи под други платформи.

## Декомпилиране на код

Понякога на програмистите им се налага да видят кода на даден модул или програма, които не са писани от тях самите и за които не е наличен сорс код. Процесът на генерирането на сорс код от същест­вуващ изпълним бинарен файл (.NET асембли – .exe или .dll) се нарича **декомпилация**.

Декомпилацията на код може да ви се наложи в следните случаи:

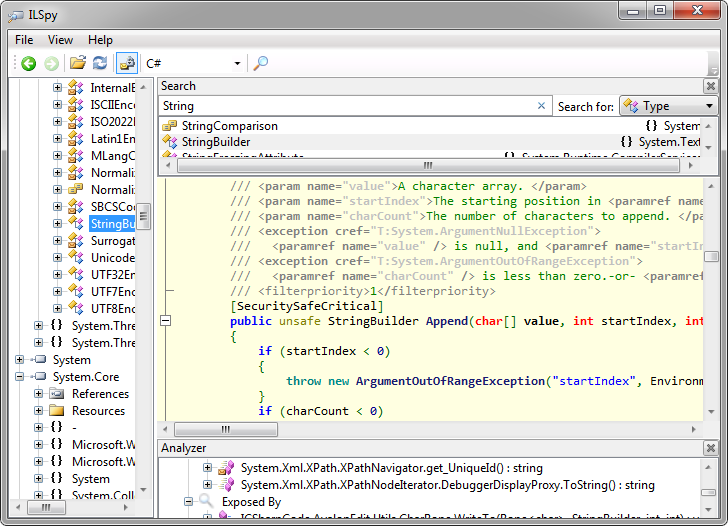
* Искате да видите как е реализиран даден алгоритъм, за който знаете, че работи добре.
* Имате няколко варианта, когато използвате нечия библиотека и искате да изберете оптималния.
* Нямате информация как работи дадена библиотека, но имате компилиран код (асембли), който я използва и искате да разберете как точно го прави.

Декомпилацията се извършва с помощни инструменти, които не са част от Visual Studio. Най-използваният декомпилатор беше Red Gate’s **Reflector** (преди да стане платен в началото на 2011). Добра алтернатива на Reflector е **Just Decompile** (декомпилаторът на Telerik), който е безплатен, с отворен код и работи доста добре: <http://telerik.com/products/decompiler>.

Друг много добър инструмент за декомпилация е инструментът с отворен код **ILSpy**, който може да бъде изтеглен от <http://ilspy.net>.

ILSpy няма нужда от инсталация. След като бъде стартиран, зарежда някои от стандартните библиотеки от .NET Framework. Чрез менюто File -> Open можете да отворите избрано от вас .NET асембли. Можете да заредите и асембли от GAC (Global Assembly Cache).

Ето как изглежда програмата по време на работа:



По два начина можете да видите как е реализиран даден метод. Ако искате да видите например как работи статичния метод System.Currency.  
ToDecimal(), първо можете да използвате дървото вляво и да намерите класа Currency в пространството от имена System, след това да изберете метода ToDecimal. Достатъчно е да **натиснете върху даден метод**, за да видите неговия C# код. Друг вариант да намерите даден клас е чрез търсене с търсачката на ILSpy. Тя търси в имената на всички класове, интерфейси, методи, свойства и т.н. от заредените в програмата асемблита.

JustDecompile и ILSpy са изключително полезни инструменти, които се използват почти ежедневно при разработката на .NET софтуер и затова задължително трябва да си изтеглите поне едно от тях и да си поиграете с него. Винаги, когато се чудите как работи даден метод или как е имплементирано дадено нещо в някое асембли, можете да разчитате на **декомпилатора**, за да научите.

## C# под Linux

В момента в България (а може би и на световно ниво) програмирането на C# за Linux е доста по-слабо развито от това за Windows. Въпреки всичко не искаме да го пропуснем с лека ръка и затова ще ви дадем отправни точки, от които можете да тръгнете сами, ако ползвате Linux.

### Програмиране на C# под Linux?

Най-важното, което ни трябва, за да програмираме на C# под Linux, е имплементация на .NET Framework. Microsoft .NET Framework не се поддържа за Linux, но има друга .NET имплементация, която се нарича **Mono**. Можете да си изтеглите Mono (който се разработва и разпро­странява като свободен софтуер) от неговия официален уеб сайт: [http://www.mono-project.com](http://www.mono-project.com/). Mono позволява да компилираме и изпълняваме програми на C# в Linux среда и върху други операционни системи. Той съдържа C# компилатор, CLR, garbage collector, стандартните .NET библиотеки и всичко останало, което имаме в Microsoft .NET Framework под Windows.

Разбира се, **Visual Studio за Linux също няма**, но можем да използваме аналога на #Develop – monoDevelop. Него можете да изтеглите от: [http://www.monodevelop.com](http://www.monodevelop.com/).

## Упражнения

1. Запознайте се с Microsoft Visual Studio, Microsoft Developer Network (MSDN) Library Documentation. Инсталирайте си Visual Studio.
2. Да се намери описанието на класа System.Console в стандартната .NET API документация (MSDN Library).
3. Да се намери описанието на метода System.Console.WriteLine() с различ­ните негови възможни параметри в MSDN Library.
4. Да се компилира и изпълни примерната програма от примерите в тази глава през командния ред (конзолата) и с помощта на Visual Studio.
5. Да се модифицира примерната програма, така че да изписва различно поздравление, например "Добър ден!".
6. Напишете програма, която изписва вашето име и фамилия на конзолата.
7. Напишете програма, която извежда на конзолата числата 1, 101, 1001 на нов ред.
8. Напишете програма, която извежда на конзолата текущата дата и час.
9. Напишете програма, която извежда корен квадратен от числото 12345.
10. Напишете програма, която извежда първите 100 члена на редицата 2, -3, 4, -5, 6, -7, 8.
11. Направете програма, която прочита от конзолата вашата възраст и изписва (също на конзолата) каква ще бъде вашата възраст след 10 години.
12. Опишете разликите между C# и .NET Framework.
13. Направете списък с най-популярните програмни езици. С какво те се различават от C#?
14. Да се декомпилира примерната програма от задача 5.

## Решения и упътвания

1. Ако разполагате с DreamSpark акаунт или вашето училище или университет предлага безплатен достъп до продуктите на Microsoft, си инсталирайте пълната версия на Microsoft Visual Studio. Ако нямате възможност да работите с пълната версия на Microsoft Visual Studio, можете безплатно да си изтеглите Visual Studio Community Edition, коeто е напълно безплатно за малки екипи и с учебна цел.
2. Използвайте адреса, даден в раздела [.NET документация](#_.NET_документацията) към тази глава. Отворете го и търсете в йерархията вляво. Може да направите и търсене в Google – това също работи добре и често пъти е най-бързият начин да намерим документацията за даден .NET клас.
3. Използвайте същия подход като в предходната задача.
4. Следвайте инструкциите от раздела [Компилация и изпълнение на C# програми](#_Компилация_и_изпълнение).
5. Използвайте кода на примерната C# програма от тази глава и променете съобщението, което се отпечатва. Ако имате проблеми с кирилицата, сменете т. нар. System Locale с български от прозореца "Region and Language" в контролния панел на Windows.
6. Потърсете как се използва метода System.Console.Write().
7. Използвайте метода System.Console.WriteLine().
8. Потърсете какви възможности предлага класа System.DateTime.
9. Потърсете какви възможности предлага класа System.Math.
10. Опитайте се сами да научите от интернет как се ползват цикли в C#.
11. Използвайте методите System.Console.ReadLine(), int.Parse() и System.DateTime.AddYears().
12. Направете проучване в интернет и се запознайте детайлно с разликите между тях.
13. Проучете най-популярните езици и вижте примерни програми на тях. Сравнете ги с езика C#.
14. Първо изтеглете и инсталирайте JustDecompile или ILSpy (повече информация за тях можете да намерите в тази глава). След като ги стартирате, отворете компилирания файл от вашата програма. Той се намира в поддиректория bin\Debug на вашия проект. Например, ако вашият проект се казва TestCSharp и се намира в C:\Projects, то компилираното асембли на вашата програма ще е файлът C:\Projects\TestCSharp\bin\Debug\TestCSharp.exe.