فهرست

سرآغاز

با ظهور دات نت و محیط توسعه‌گر VS و پس از آن به حاشیه رفتن تمامی محیط‌های تولید نرم‌افزار همچون دلفی محبوب، مایکروسافت این بار با خود به رقابت می‌پردازد و در هر نسخه ای از دات نت ما شاهد پیدایش تکنولوژی‌ها و ایده‌های مختلف در جهت تسریع و تنظیم کیفیت تولید که منجر به نگهداری و توانایی توسعه‌پذیری آینده‌ی نرم‌افزارها می‌باشد، هستیم.

در نسخه Visual Studio 2008 این بار یکی از افراد نابغه که سابقاً در شرکت بورلند کار می‌کرد، به نام آقای آندرس هلسبرگ، زبان غیر روالمند LINQ را اختراع کرده و در زبان‌های برنامه‌نویسی دات نت همچون C#، Vb.Net قرار داد.

LINQ شاهکاری بود که امروزه آن را آینده دسترسی به داده ها می‌دانند.آقای هلسبرگ در یکی از مجلات مشهور جهانی، مرد نابغه سال ۲۰۰۹ نامیده شد. اما LINQ تنها پیشکشی دات نت ۲۰۰۸ نبود. در این نسخه تکنولوژی هایی همچون WCF، WF، WPF و Cardshape افزوده شد.

در Visual Studio 2010 ضمن تکمیل و توسعه‌ی تکنولوژی‌ها در جهت توانمندسازی آنها ،امکانات محیط VS در مواردی چون خطایابی، ترسیم دیاگرام کلاس‌ها و UML نیز ارتقا پیدا کرده است. لازم به ذکر است که تغییراتی در زبان VB.Net نیز اعمال شده است که در طول کتاب با آنها آشنا خواهید شد.

در اینجا لازم می‌دانم شما را با بخش‌هایی از کتاب آشنا کنم:

بررسی مقدمات زبان برنامه‌نویسی VB.NET:

در این بخش ضمن مرور بر سنت زبان VB.NET، ساختارها و مفاهیم جدیدی که در VB2010 بوجود آمده نیز مورد بررسی قرار می‌گیرد.

جهش به دنیای برنامه نویسی شی گرا:

یک برنامه‌نویس کار خود را با آموختن الگوریتم و فلوچارت جهت توانائی در حل مسئله آغاز می کند و پس از خبرگی در حل مسائل، به سراغ یک زبان برنامه‌نویسی جهت کد کردن الگوریتم‌ها می‌رود. اما باید توجه داشته باشید که پس از طی این مراحل و رسیدن به مرحله‌ی توانائی در کد کردن، هنوز آموزه‌های پایه‌ای ملزوم برای برنامه‌نویس کامل نشده است، بلکه این بار یک برنامه‌نویس امروزی در جهت تولید نرم‌افزار، نیاز به یک جهش دارد و آن آشنائی و توانائی درک شی‌گرایی و برنامه نویسی شی‌گرا می‌باشد. زیرا شی‌گرا امروزه پایه و اساس تولید نرم‌افزار و تمامی تکنولوژی‌های دات نت است. از آنجا که همیشه تاکید بنده بر آن است که برنامه‌نویسان مفاهیم شی‌گرا را خوب یاد بگیرند تا بتوانند آنها را به درستی بکار بندند، فصلی کامل به زبانی ساده، همراه با مثال‌هایی از پیرامون زندگی واقعی نوشته‌ام و توصیه می شود به هیچ وجه تا این بخش را خوب درک نکرده و آن را یاد نگرفته‌اید، به سراغ

گام‌های بعدی نروید. زیرا مدامی که برنامه‌نویسی شی‌گرا را خوب بلد نباشید، نخواهید توانست با تکنولوژی‌های جدید همچون LINQ To SQL، Entity Framework کار کنید. ضمناً در صورت عدم توانائی شما در برنامه‌نویسی شی‌گرا، در استفاده از خود محیط ویژوال استدیو نیز توانمند نخواهید شد. همچنین در صورت ناتوانی در این بخش، امکان پیاده‌سازی برنامه‌نویسی سه لایه که در این کتاب به سادگی بیان شده است نیز برایتان میسر نخواهد بود.

ابزارهای اضافه:

در هنگام اجرای پروژه‌ها، برنامه‌نویسان به امکاناتی بیش از امکانات فعلی موجود در محیط دات نت نیازمندند. لذا در گام کار با اکتویکس‌ها ، موضوعاتی چون کار با فایلهای PDF، SWF، استفاده از MsAgent،AnalogClock، تقویم شمسی و برنامه نویسی مدیریت ارسال و دریافت SMS، برنامه‌نویسی استفاده خودکار از نرم افزارهای Excel،Word و آشنائی با ابزارهای زیبا و تکمیل شده‌ی کنترل‌های قبلی دات نت به نام DevComponent آشنا خواهید شد.

زبان LINQ:

استفاده از زبان LINQ بسیار ساده است، زیرا این زبان نیازهای شما را بی‌آنکه الگوریتم آن را بدانید انجام می‌دهد. این زبان جهت استفاده در منابع داده‌ای مختلف مهیا شده است. لذا جهت جلوگیری از سخت شدن آموزش، کاربرد LINQ را پس از معرفی هر منبع داده‌ای، در فصلی جداگانه قرار داده ایم. در این فصل‌ها سعی شده زوایا و کاربرد این زبان در پروژه‌های بزرگ و واقعی مورد برررسی قرار بگیرد. در صورت مشکل در این زبان سعی کنید به فصل‌های مقدماتی آن مراجع نمائید.
در این کتاب امکان انجام همه‌ی موضوعات هم در زمان طراحی و هم در زمان کدنویسی بیان شده است.

تولید نرم افزار اتوماسیون:

با توجه به نیاز روز افزون کشور در جهت تولید نرم‌افزارهای تجاری و رشد فوق العاده VS در این زمینه و با توجه به تغییر نگرش دسترسی به داده ها در NET. و کمبود منابع لازم جهت توجیه برنامه‌نویسان در استفاده از این معماری‌های جدید، لازم بر آن دانستم که گام کاملی مجتمع بر هشت فصل جهت بررسی بانک‌های اطلاعات و تکنولوژی‌های دسترسی به آنها را جهت آموزش در کتاب به گنجانم.
در این بین با توجه به عدم وجود منابع فارسی در موضوعاتی همچون Entity Framework و برنامه نویسی سه لایه، سعی بر آن شده که مطالب به طور تئوری و عملی ارائه شوند.

پروژه:

مطابق عهدی که با خود بسته‌ام،همانند کتاب قبلی در این کتاب نیز یک پروژه‌ی دیگر جهت اجرای مطالب آموزش داده شده تولید کردیم. به نظر می‌رسد این پروژه، راهنما و منبعی بسیار مناسب برای برنامه‌نویسان در زمینه‌ی برنامه‌نویسی شی‌گرا و سه لایه باشد.

در انتها از بزرگوارانی که مرا در این کتاب یاری داده‌اند به ترتیب زیر تشکر و قدردانی می کنم:

از سرکار گرامی خانم مهندس منصوره حاجی‌زاده که در تولید پروژه، فصل اکتویکس‌ها و ویراست علمی بسیار کمک نموده‌اند تشکر و کمال قدردانی را دارم.

از استاد بزرگوار جناب آقای مهندس حمزه پارسی نژاد جهت طراحی جلد خلاقانه و زیبایشان ممنون و سپاسگذارم.

از آقای مهندس محمد رفیع نبی‌زاده جهت عکس‌برداری بسیار ممنونم.

در انتها امیدوارم توانسته باشم، گامی کوچک جهت تولید نرم‌افزارهای شی‌گرا و با استفاده از تکنولوژی‌های جدید برای برنامه‌نویسان عزیز ایرانی برداشته باشم.

بی‌صبرانه منتظر پیشنهادها و انتقادات شما خوانندگان عزیز هستم.

MohsenModhej@gmail.com

WWW.HowPrg.com

محسن مدحج

کتاب راهنمای کاربردی Visual Basic.Net 2010

سال ۱۳۸۹

chapter 1

فصل ۱

بررسی مقدمات زبان برنامه نویسی VB.NET

Visual Studio

صحبت کردن ساده است. کدت رو نشون بده.	(Linus Torvalds)

مقدمات زبان برنامه نویسی VB.NET2010 :

هر زبان برنامه‌نویسی دارای قوانین و ساختار مربوط به خود در رابطه با تعریف متغیرها، شرط‌ها، حلقه‌ها و... می‌باشد که در ابتدای شروع به کار با آن زبان باید به سراغ آموختن این موضوعات برویم.

کلمات کلیدی :

کلمات ذخیره شده‌ای هستند که برای اهداف خاص مورد استفاده قرار می‌گیرند که به آنها کلمات کلیدی[1] یا کلمات دخیره شده می‌گویند، مانند کلمه کلیدی Dim که برای تعریف متغیرها به کار می‌رود.

درجدول زیرتعدادی از کلمات کلیدی را که در قسمت‌کلاسها، زیربرنامه‌ها، شرط‌ها، ساختارهای تکرار، نوع داده‌ها، استثناها، رویدادها و آنهایی که به صورت عمومی بکار می‌روند را لیست کرده‌ایم:

شناسه: منظور از شناسه نام‌هایی است که به توابع، متغیرها و ... داده می‌شود (درحقیقت هویت یک چیز را بیان می‌کند). شناسه‌ها به دو دسته‌ی شناسه‌های استاندارد و غیر استاندارد تقسیم می‌شوند:

الف) شناسه‌های استاندارد:

به منظورهای خاص توسط کامپایلر زبان VB.Net استفاده می‌شوند، مانند:

دیتا تایپها	توابع	
Int16	Abs	Round
Int32	Atan	Pow
Int64	Cos	Sqrt
UInt16	Sin	Asc
UInt32	Tan	InStr
UInt64	Exp	Left

[1] Reserved Word

ب) شناسه‌های غیر استاندارد:
کلمات معتبری هستند که نه جزء کلمات کلیدی و نه جزء شناسه‌های استاندارد هستند و برنامه نویس می‌تواند برای نام متغیرها ،کنترل‌ها، توابع، برچسب‌ها و ... از آنها استفاده کند.
با دانستن مفاهیم فوق، مقدمات زبان برنامه نویسی VB.NET را آغاز می‌کنیم.

متغیر چیست؟

محلی از حافظه‌ی اصلی (Ram) است که برای نگهداری انواع داده‌ها به کار می‌رود. همانطورکه از نام متغیر مشخص است، امکان تغییر مقادیر درون آن در طول اجرای برنامه فراهم خواهد بود. برای تعریف متغیر دو مرحله‌ی اصلی در پیش رو داریم:

۱. نامگذاری متغیر (طبق قوانین)
۲. تعیین نوع متغیر

۱- نامگذاری متغیر طبق قوانین:
هر متغیر دارای نامی است که به وسیله‌ی آن می‌توان به متغیر مورد نظر دسترسی پیدا کرد. همانند انسان‌ها که برای دسترسی به آنها از نامشان استفاده می‌کنیم. اما این نامگذاری در هر زبان دارای قواعدی است که باید بر طبق آنها انجام شود. در ادامه، قوانین تعریف متغیر را بررسی می‌کنیم:

- اولین کاراکتر باید حرف یا زیر خط باشد. از کاراکتر اول به بعد می‌توانیم از اعداد و حروف نیز استفاده کنیم.
- نام متغیر نباید جزء کلمات کلیدی و شناسه‌های استاندارد باشد.
- در نام متغیر نباید از Blank (فاصله) و علائم خاص مانند ؟ ، ! ، $ و ... استفاده کرد و حداکثر طول یک شناسه ۱۰۲۳ کاراکتر است.

علاوه بر قوانین نامگذاری ذکر شده بهتر است به هنگام نامگذاری متغیرها به نکات زیر نیز توجه کنید. با استفاده از این نکات می‌توانید خوانایی برنامه‌هایتان را افزایش دهید.

- در نام متغیر از پیشوندی استفاده کنید که نوع متغیر را مشخص کند. بدین ترتیب به وسیله‌ی نام متغیر از نوع آن هم مطلع خواهید شد و نیازی نیست به قسمت تعریف متغیر مراجعه کنید. در جدول زیر پیشوندهای برخی از نوع داده‌ها نشان داده شده است:

نوع داده	پیشوند	مثال
Boolean	Bln	Bln_flag
Byte	Byte	Byte_len
Short	Short	Short_number
Integer	Int	Int_code
Long	Lng	Lng_price
Single	Sng	Sng_avg
Double	Dbl	Dbl_scientific
String	Str	Str_firstname
Char	Chr	Chr_symbol

• از نام‌هایی استفاده کنید که متناسب با عملکرد متغیر باشند. به این ترتیب برنامه قابل فهم تر خواهد بود و نیاز کمتری به مستند سازی خواهد داشت. به طور مثال اگر متغیری را برای نگهداری نام افراد استفاده می‌کنید، می‌توانید از str_firstname برای نام آن استفاده نمایید.

برای جدا کردن قسمت‌های نام یک متغیر می‌توانید از حروف بزرگ و یا زیرخط استفاده کنید، مانند StrFirstName (به این روش نامگذاری شتری گفته می‌شود چون نام متغیر مانند کوهان شتر دارای پستی و بلندی می‌گردد.) و یا str_first_name. (به این ترتیب کاملا مشخص است که یک متغیر رشته‌ای، برای نگهداری نام کوچک افراد تعریف نموده‌ایم.)

نامهای درست برای متغیر:

```
a2 , a2b , _2c
```

نامهای نادرست برای متغیر:

```
2a , a  b , -2c , a?b
```

۲- تعیین نوع متغیر:

داده‌ها انواع مختلفی دارند. در این مرحله مشخص می‌کنیم که متغیر چه نوع داده‌ای (Data type)[۲] را می‌تواند در خود جای دهد.

[۲] Data type : داده‌هایی که در کامپیوتر ذخیره می‌شوند، انواع مختلفی دارند. این نوع‌ها را Data type می‌نامیم.

انواع داده‌ها:

الف- داده‌های عددی

ب- داده‌های غیر عددی

الف- داده‌های عددی:

داده‌های عددی بر دو قسم می‌باشند:

۱. داده‌های عددی صحیح
۲. داده‌های عددی اعشاری

جدول نوع داده‌های عددی صحیح:

نوع داده (data type)	مقدار حافظه اشغال شده (برحسب بایت)	محدوده داده ای
Integer, Int32	4	-2'147'483'648 u 2'147'483'647
Short, Int16	2	-32'768 u 32'767
Long, Int64	8	-9'223'372'036'854'775'808 u 9'223'372'036'854'775'807
Byte	1	0 u 255
Sbyte	1	-128 u 127
UInteger,UInt32	4	0 u 4,294,967,295
ULong ,UInt64	8	0 u 18,446,744,073,709,551,615 (1.8...E+19)
UShort,UInt16	2	0 u 65,535

جدول نوع داده‌های عددی اعشاری:

نوع داده (data type)	مقدار حافظه اشغال شده (برحسب بایت)	محدوده داده‌ای
Single	۴	برای مقادیر مثبت 1.401298E-45 تا 3.402823E+38 برای مقادیر منفی -3.402823E+38 تا -1.401298E-45
Double	۸	اعداد بزرگ
Decimal	۱۶	اعداد بسیار بزرگ

ب- داده‌های غیر عددی:

داده‌های غیر عددی عبارتند از:

۱- داده‌های کاراکتری

۲- داده‌های رشته‌ای

۳- داده‌های منطقی

جدول نوع داده‌های کاراکتری و رشته‌ای:

محدوده داده‌ای	نوع داده (data type)
۱ کاراکتر	Char
۲میلیون کاراکتر	String

جدول نوع داده‌های منطقی و آبجکت:

محدوده داده‌ای	مقدار حافظه اشغال شده (برحسب بایت)	نوع داده (data type)
T or F	۱	Boolean
هرنوع داده‌ای	۴ یا ۸	Object

اکنون با دانستن مفاهیم فوق می‌توان گفت: تعریف یک متغیر یعنی نامگذاری صحیح آن و تعیین نوع داده‌ای که می‌تواند ذخیره کند.

چگونگی تعریف متغیر:

- شکل کلی تعریف یک متغیر به صورت زیر است:

```
Dim  نام متغیر  as  نوع متغیر
یا
Dim  نام متغیر۱ , نام متغیر ۲ , ... as  نوع متغیر
و یا
Dim  نام متغیر۱ as  نوع متغیر , نام متغیر ۲ as  نوع متغیر , ...
```

```
Dim  shr_len  as short
```

در این مثال shr_len خانه‌ای از حافظه است که فقط می‌تواند اعداد ۳۲,۷۶۸- تا ۳۲,۷۶۷ را نگهـداری کند.

- 32768 =< a < 32767

```
Dim StrFirstName , StrLastName as String
Dim StrFirstName as String , ShrLen as Short
```

برخی از نوع داده‌های معروف و پرکاربرد، دارای کاراکتر شناسه نوع داده[۳] مخصوصی هستند.

به عنوان مثال :

Integer	%
Long	&
Single	!
Double	#
Decimal	@
String	$

و شکل کلی استفاده از آنها در تعریف متغیر به صورت زیر است:

```
Dim  کاراکتر شناسه نوع داده+ نام متغیر
یا
Dim  کاراکتر شناسه نوع داده + نام متغیر , کاراکتر شناسه نوع داده+ نام متغیر , ...
```

[۳] Data-Type Suffix

```
Dim   StrLastName$ , SngAvg! , IntCode%
```

گاهی این روش بهترین راه حل برای تعریف چند متغیر از نوع‌های مختلف است.

همیشه باید بسته به نوع داده، Data type ای را در نظر بگیریم که کمتـرین میـزان حافظه را اشغال نماید.

دو متغیر با شرایط مشخص شده تعریف نمائید:

0=<a<255 , b>255

پاسخ درست :

```
Dim a as byte
Dim b as short
```

پاسخ نادرست :

```
Dim a,b as short
```

وقتی نوع متغیر مشخص نشود به صـورت خودکـار از نـوع داده‌ای Object در نظـر گرفته خواهد شد و در این صورت امکان نگهداری هر نوع داده‌ای در آن فراهم می‌شود.

به شکل زیر:

Dim نام متغیر

```
Dim a
A=100
A="sss"
A=2.33
```

انتساب داده به متغیر:

عمل مقداردهی به متغیرها را دستور انتساب داده می‌نامیم. این عمل به دو شکل انجام می‌شود:

الف) در زمان تعریف (مقداردهی اولیه) به شکل زیر:

Dim نام متغیر as نوع متغیر = مقدار یا متغیر

```
Dim a As Byte = 100
```

ب) در طول برنامه به شکل کلی زیر:

نام متغیر = مقدار یا متغیر

```
Dim a As Byte
a = 100
```

داده‌ها باید با نوع متغیرها و محدوده‌ی آنها سازگاری داشته باشند. در مثال زیر یک نمونه از مقداردهی نادرست متغیر را می‌بینیم:

```
Dim A as byte
A = 1000
```

در مثال بالا a را در محدوده‌ی byte تعریف کرده‌ایم، پس نمی‌توانیم به آن عدد ۱۰۰۰ را نسبت دهیم.

اگر نکته شماره ۱ رعایت نشود دو نوع خطا به وجود می‌آید:

۱- اگر داده با نوع متغیر هماهنگ نباشد خطای زمان[۴] اجرا را مشاهده می‌کنیم.

```
Dim A as short
A = "salam"
```

[۴] خطای زمان اجرا، خطایی است که در هنگام اجرای برنامه رخ داده، باعث توقف برنامه می‌شود و سپس خطی که خطا در آن رخ داده است مشخص می‌گردد.

در این حالت خطای زمان اجرا را در هنگام مقداردهی به این متغیر مشاهده می‌کنیم. علت این خطا، آن است که در VB.Net ، متغیرها در زمان اجرا ساخته و مقداردهی می‌شوند.

۲- اگر داده در محدوده‌ی متغیر نباشد، دو حالت ممکن است پیش بیاید:

حالت اول : مستقیماً به یک متغیر مقداری خارج از محدوده بدهیم که در این صورت خطای کامپایلری[۵] را مشاهده می‌کنیم.

```
Dim A as byte
A = 4000
```

حالت دوم : متغیری را به آن انتساب دهیم که در محدوده متغیر اولیه نباشد، در اینصورت خطای زمان اجرا رخ می‌دهد.

```
Dim B as integer , A as byte
B=3276
A=B
```

داده‌های صحیح در متغیر اعشاری می‌تواند قرار بگیرد، اما عکس این قضیه یعنی قرار دادن داده‌های اعشاری در متغیر صحیح باعث می‌شود که قسمت اعشاری داده حذف گردد. به این نوع تبدیلات که خود کامپایلر VB.Net انجام می‌دهد، تبدیلات ضمنی می‌گویند.

```
Dim A As Single = 9.987
A = 99
Dim b as byte
B=9.987          /عدد ۹ می‌شود/
```

انواع متغیرهایی که در محدوده یکدیگرند و هم‌نوع هستند می‌توانند در یکدیگر قرار گیرند.

[۵] خطای کامپایلری، خطایی است که در زمان کدنویسی رخ می‌دهد و با کشیدن خط در زیر کد، محل خطا مشخص می‌شود .

```
Dim  a , b  as  short
A=20
B=A
```

در vb.net اعـدادی کـه نوشـته مـی‌شـوند بـه طـور پـیش فـرض از نـوع Integer می‌باشند.

استفاده از نوع داده‌های کاراکتری و رشته‌ای:

داده‌های کاراکتری[۶]:

داده‌های کاراکتری فقط شامل یک کاراکتر هستند و می‌توانند شامل اعداد، حروف و علامتها باشند. اگر بیش از یک کاراکتر در یک متغیر کاراکتری قرار دهیم، بقیه‌ی کاراکترها حذف می‌شوند.

```
Dim  chr_symbol as  char
chr_ symbol = "?"
chr_ symbol = "b"
chr_ symbol = "2"
chr_ symbol ="Iran"        →        chr_ symbol ="I"
```

در صورتی که قصد داشته باشید خود کاراکتر دابل کتیشـن را بـه عنـوان یـک داده نگهداری نمائید باید دو بار این کاراکتر را تایپ کنید.

```
Dim  chr_symbol as  char
chr_ symbol = """"
```

داده‌های رشته‌ای[۷]:

داده‌های رشته‌ای مجموعه‌ای از کاراکترهای به دنبال هم هستند. این کاراکترها می‌توانند حرف، علامت

---

[۶] Character

[۷] String

و حتی عدد باشند، مقادیر رشته‌ای و کاراکتری همیشه بین دو علامت دابل کوتیشن (" ") قرار می‌گیرند.

example

```
Dim str$
Str = "mohsen"
Str = "4537"
```

اگر تعداد کاراکترها بیش از دو میلیون، شود بقیه کاراکترها حذف می‌شوند.

ثابتها :

ثابت‌ها نیز مانند متغیرها برای نگهداری داده‌ها استفاده می‌شوند با این تفاوت که پس از تعریف مقدار ثابت، دیگر مقدار آن در طول برنامه تغییر نمی‌کند و ثابت در حقیقت نامی مجازی برای یک داده است که به طور مکرر در برنامه مورد استفاده قرار می‌گیرد و به شکل‌های کلی زیر قابل استفاده می‌باشد:

```
Const نام متغیر as نوع داده = مقدار مورد نظر
یا
Const نام متغیر = مقدار مورد نظر
```

در شکل اول باید همان اصول انتساب داده به متغیر رعایت شود، اما در شکل دوم می‌توان هر مقداری را نسبت داد.

example

```
Const a as byte = 19
Const a! = 20
Const a = 21
```

ثابت‌ها به شما امکان جایگزینی نام‌هایی را که به خاطرآوردنشان آسان است به جای مقادیر لیترالی که به خاطرآوردنشان دشوار است را می‌دهند. این کار می‌تواند نوشتن و نگهداری کد شما را تسهیل نماید.

کلمه کلیدی Nothing :

هنگام تعریف یک متغیر، آن متغیر بسته به نوع داده‌ای که از آن تعریف شده است دارای یک مقدار پیش فرض خواهد بود. به عنوان مثال متغیر نوع Boolean دارای مقدار False و Integer دارای مقدار صفر(۰) می‌باشد.

پس از تعریف یک متغیر تا قبل از مقداردهی، آن متغیر دارای مقدار پیش فرض می‌باشد.

در هنگام کدنویسی در شرایطی که قصد ندارید مقدار خاصی را به یک متغیر (حال از هر نوعی) انتساب دهید کافی است، کلمه کلیدی Nothing را به آن انتساب دهید تا آن متغیر مقدار پیش فرض خود را ذخیره نماید.

example

```
Dim a As Integer
a=8
a = Nothing
Dim s As String
s="Ahmad"
s = Nothing
Dim b As Boolean
b = Nothing
Dim c As Font = Nothing
```

نوع Nullable :

متغیرهایی از نوع Integer، Double و... ، در صورتی که مقداری درون آنها وجود نداشته باشد، مقدار صفر را به صورت پیش فرض نگهداری می‌کنند. اما توجه داشته باشید که خود صفر یک عدد است و مقدار تهی نیست حال جهت آنکه این امکان پدید آید که اینگونه متغیرها مقدار تهی را نیز بتوانند نگهداری کنند از نوع Nullable به شکل کلی زیر استفاده می‌کنیم:

```
Dim نام متغیر As New Nullable(Of نوع داده)
```

example

```
Dim a As New Nullable(Of Integer)
```

در شکل فوق متغیر a در صورتی که هیچ مقداری به آن نسبت داده نشده باشد دیگر مقدار صفر را به طور پیش فرض نگهداری نمی‌کند بلکه دارای مقدار تهی یا هیچ خواهد شد.
جهت دادن مقدار تهی به اینگونه متغیرها از کلمه کلیدی Nothing استفاده می‌شود.

```
a=Nothing
```

در صورتی که متغیری عددی به صورت معمول (غیر Nullable) تعریف شود و مقدار Nothing را به آن اختصاص دهیم مقدار تهی را نگهداری نمی‌کند. مثلاً در نوع Integer بازهم مقدار صفر نگهداری خواهد شد.

روشی دیگر جهت تعریف یک متغیر از نوع Nullable وجود دارد، برای این منظور به راحتی می‌توانید با اضافه کردن یک علامت سوال در پایان نوع داده اینکار را انجام دهید.

```
Dim a As Integer?
Dim b As Double?
Dim c As Byte?
```

داده‌های شمارشی :

جهت درک راحت‌تر اینگونه داده‌ها تصور کنید، تعدادی از ثابت‌های عددی مرتبط با یکدیگر را در یک گروه قرار داده‌اید.
به شکل کلی زیر:

```
Enum نام As نوع داده عددی
مقدار عددی = عضو۱
مقدار عددی = عضو۲
End Enum
```

حال جهت استفاده از آنها کافی است که نام گروه و سپس نام ثابت مورد نظر را صدا بزنید:
به شکل کلی زیر:

```
عضو مورد نظر. نام داده شمارشی
```

```
Enum روزهای_هفته As Integer
    شنبه = 0
    یکشنبه = 1
    دوشنبه = 2
    سه_شنبه = 3
End Enum
```

همانطور که مشاهده می‌کنید در مواردی که مجبور به قرار دادن فاصله بودیم از علامت زیرخط (_) استفاده کردیم.

در این مثال جهت اشاره به این مطلب که امکان تعریف شناسه‌های فارسی نیز وجود دارد از حروف فارسی استفاده کردیم لذا پیشنهاد نمی‌شود در کد برنامه‌ها از حروف فارسی استفاده شود.

داده‌های شمارشی باید خارج از زیربرنامه (Sub, Function) تعریف شوند.

یک ساختار داده‌ی شمارشی را می‌توان به عنوان یک نوع داده استفاده نمود و از آن متغیر تعریف کرد.

example

```
Enum روزهای_هفته As Integer
    شنبه = 0
    یکشنبه = 1
    دوشنبه = 2
    سه_شنبه = 3
End Enum

Sub Main()
    Dim a As روزهای_هفته
    a = دوشنبه.روزهای_هفته
End Sub
```

نوع داده ساختاری :

این نوع داده‌ها توسط برنامه نویسان تعریف می‌شوند و توسط آن می‌توان یک مجموعه از متغیرها با نوع‌های مختلف را در یک گروه با یک عنوان ذخیره نمود به عنوان مثال می‌توان یک ساختار به نام Personel ایجاد کرده و متغیرهای با عناوین نام، نام خانوادگی و سن را در آن قرار داد.

جهت تعریف یک ساختار از شکل کلی زیر استفاده می‌شود:

```
Structure نام ساختار
  Dim نام متغیر As نوع متغیر
  Dim نام متغیر As نوع متغیر
...
End Structure
```

example

```
Structure Personel
  Dim Name As String
  Dim Fmily As String
  Dim age As integer
End Structure
```

از Structure همانند داده‌های شمارشی نمی‌توان به طور مستقیم استفاده نمود بلکه باید آن را به عنوان یک نوع داده تصور کرده و متغیری از این نوع داده جدید تعریف نمائیم:

```
Dim prsl As Personel
   prsl.age = 1
   prsl.Name = "fardin"
   prsl.Fmily = "Salami"
```

همانطور که مشاهده می‌کنید ابتدا یک متغیر از این نوع داده‌ی جدید تعریف کرده و جهت استفاده از اعضای درون آن، پس از نام متغیر یک نقطه گذاشته و سپس عضو مورد نظر را انتخاب می‌کنیم.

کلمه کلیدی With :

همانطور که در مثال فوق مشاهده می‌کنید برای دسترسی به عضوهای ساختار پرسنل هر بار باید ابتدا نام آن را تایپ کرده تا بتوانیم به عضو مورد نظر دسترسی پیدا کنیم جهت سهولت کار و جلوگیری از تایپ مجدد نام ساختار از عمگر With به شکل کلی زیر استفاده می‌کنیم:

```
With نام متغیر ساختار
      مقدار مورد نظر = عضو.
      مقدار مورد نظر = عضو.
End With
```

```
Dim prsl As Personel
  With prsl
     .age = 1
     .Name = "fardin"
     .Fmily = "Salmi"
  End With
```

آشنایی با محیط Console Application :

حال وقت آن است که آموخته‌های خود را در زبان VB.Net عملی کنیم اما به علت آنکه هنوز محیط ویژوال را نشناخته‌ایم، نمی‌توانیم در محیطی که حاوی اشیاست برنامه نویسی کنیم. بنابراین از Console که در VS.Net وجود دارد و جهت ایجاد برنامه‌های شبیه به سیستم عامل DOS[۸] با کدنویسی VB.Net است استفاده می‌نماییم.

[۸] - منظور از DOS همان Command Prompt ویندوز است که برنامه‌های تحت داس را شبیه سازی می‌کند. اینگونه برنامه‌ها فقط دارای خروجی متنی هستند و هیچگونه عنصری در آنها وجود ندارد.

نخستین باری که VS را اجرا می کنید پنجره‌ای نمایش داده می شود (تصویر ۱-۱). مطابق با شکل زبان Visual Basic Development Settings را انتخاب کنید. اینکار سبب می شود یک سری پیش فرض‌ها در محیط VS برای زبان مورد نظرتان در نظر گرفته شود.

تصویر ۱-۱

اکنون مراحل زیر را برای ایجاد برنامه Console دنبال کنید.

برای این کار به ترتیب File → New → Project را برگزینید (تصویر۲-۱).

تصویر۲-۱

سپس از کادر نمایش داده شده گزینه Console Application را انتخاب کنید.

با فشردن کلید ok شکل زیر را مشاهده می‌کنید.

```
Module Module1
   Sub Main ()


   End Sub
End Module
```

شروع کدنویسی در Console :

توجه داشته باشید که همه‌ی کدها را باید بین Sub Main و End Sub بنویسید.

در ابتدا باید کد زیر را وارد کنید:

```
Console.ReadLine
```

علت نوشتن این کد آن است که پنجره Console بعد از اجرا بلافاصله بسته می‌شود و در واقع شما با نوشتن این کد به پنجره دستور می‌دهید که تا کاربر Enter نکرده است بسته نشود.

دستورهای ورودی/ خروجی :

دستور ورودی:

دستور Console.ReadLine برای گرفتن ورودی از کاربر استفاده می‌شود. در شکل زیر توسط این دستور متغیر a مقداردهی شده است:

```
a = console.readline ()
```

example

```
Module Module1
   Sub Main ()
         a = console.readline ()
   End Sub
End Module
```

در این مثال a مقداردهی می‌شود و بعد پنجره بسته می‌شود. برای اینکه این شرایط پیش نیاید باید بلافاصله کد مربوطه (یعنی Console.readline()) را تایپ کنید. به کد زیر دقت کنید:

```
Module Module1
   Sub Main ()
         a = console.readline ()
         Console.readline()
   End Sub
End Module
```

برای اجرای برنامه دکمه‌ی (Start Debugging) یا کلید F5 را کلیک کنید. بدین ترتیب شکل زیر را مشاهده خواهید کرد:

نکته مقداری را که وارد می‌کنید باید هم‌نوع و هم‌محدوده با نوع متغیر باشد.

example

```
Dim a as integer=console.readline()
```

برای وارد کردن مقادیر رشته‌ای و کاراکتری، همانند دستور انتساب نیاز نیست از علامت دابل کوتیشن("") استفاده شود.

دستور خروجی:

برای چاپ کردن مقادیر و متغیرها بر روی صفحه‌ی نمایش از شکل کلی زیر استفاده می‌کنیم:

Console.Write (مقدار یا متغیر مورد نظر)

example

```
Console.Write ("Hello My Friend")
```

عملگرها :

یک عبارت شامل مجموعه‌ای از مقادیر، متغیرها و ... می‌باشد. برای ارتباط هر یک از این اجزا از عملگرها استفاده می‌کنیم و آن وقت همین متغیرها عملوند به حساب می‌آیند (به عبارت دیگر به طرفین هر عملگر، عملوند گفته می‌شود). عملگرهای مختلفی وجود دارند که کارهای مختلفی انجام می‌دهند مانند عملگر (+) که برای جمع دو عدد در مقادیر عددی استفاده می‌شود.

عملگرها به چند دسته تقسیم می‌شوند :

۱- عملگرهای محاسباتی : جمع (+)، منها (-)، ضرب (*)، توان (^)، تقسیم با اعشار(/)، تقسیم صحیح (\)، mod (باقی مانده)

۲- عملگرهای منطقی و بیتی : And , Or , Xor , Not , AndAlso , OrElse

۳- عملگرهای رابطه‌ای یا مقایسه‌ای : <,>,<=,>=,<>,=

۴- عملگرهای ترکیبی : =+ , =* , =/ , =\ , =&

۵- عملگرهای رشته‌ای : &,+,=+

عملگر تقسیم صحیح (\) : مقدار یا متغیر سمت چپ را به مقدار یا متغیر سمت راست تقسیم می‌کند که جواب همواره عددیست صحیح (از قسمت اعشاری صرف نظر می‌شود). اما در صورت استفاده از عملوندهای اعشاری، آنها گرد می‌شوند.

```
dim A as byte
A=12 \ 10                    →      A=1
```

عملگر تقسیم اعشاری (/) : مقدار یا متغیر سمت چپ را به مقدار یا متغیر سمت راست تقسیم می‌کند و جواب همواره عددیست اعشاری.

```
dim A as single
A=12/10                      →      A=1.2
```

عملگر mod : عدد سمت چپ را بر عدد سمت راست تقسیم می‌کند و باقی مانده را به عنوان جواب بازمی‌گرداند. جواب آن همواره عددیست صحیح.

```
Dim A as byte
A=12 mod 10                  →      A=2
```

• مثالی از عملگرهای ترکیبی:

```
Dim A as byte
A=5
A+=5                                →      A=10
A - = 2                 →           A=8
A *= 2
A \= 2
A /= 2
A ^= 2
```

عملوندها باید با یکدیگر و نیز با عملگرها سازگاری داشته باشند. همچنین باید توجه داشته باشید که مقدار بازگشتی از عبارات همواره با نوع و محدوده متغیر در برگیرنده‌اش سازگاری داشته باشد.

نوع هر عبارت هم‌نوع با عملوندهای همان عبارت تعیین خواهد شد.

اگر نوع داده عملوندها در یک عبارت یکسان باشد:

Byte + Byte➔Byte
Integer * Integer➔ Integer

تفاوتی نمی‌کند که عملگر در عبارات چه چیزی باشد.

اگر نوع داده عملوندها در عبارت متفاوت باشد آنگاه نوع داده با محدوده بیشتر به عنوان نوع داده کل عبارت در نظر گرفته می‌شود:

Integer +byte➔ Integer
Integer * byte➔ Integer

نتیجه‌ی عبارات عددی باید در محدوده‌ی عملوندهایشان (اعضای عبارت) باشد.

```
 Dim b, b1 As Byte, i As Integer
b = 100
b1 = 200
i = b + b1              Error
```

به صورت ساده می‌توان گفت که در خط آخر، خطا در سمت راست عبارت اتفاق افتاده است یعنی قبل از آنکه نتیجه در متغیر I قرار گیرد خطا رخ می‌دهد، علت آن است که عبارت دارای عملوندهای b و b1 از نوع Byte است و با توجه به نکته فوق نوع داده‌ای کل عبارت همان Byte در نظر گرفته می‌شود. حال آنکه خروجی عبارت برابر با ۳۰۰ بوده و از محدوده‌ی ۲۵۵ خارج است.

حال خطای بالا را به شکل زیر برطرف می‌کنیم:

```
i = 1 * b * b1
```

در این جا عملوند ۱ در حاصل عبارت بی‌تاثیر بوده اما با توجه به آنکه در VB.Net اعداد به تنهایی از نوع Integer در نظر گرفته می‌شود این روش باعث رفع خطا خواهد شد.

نوع داده عبارت قبل به صورت زیر می‌شود:

عملگرهای مقایسه‌ای :

نتیجه عباراتی با عملگرهای مقایسه‌ای جوابیست همواره منطقی (Trueیا False) و معمولا در دستورات شرطی استفاده می‌شوند. در جدول زیر عملگرهای مقایسه‌ای لیست شده اند:

عملگر	کاربرد
=	مساوی
<	کوچکتر از
< =	کوچکتر یا مساوی
< >	نامساوی
>	بزرگتر
> =	بزرگتر یا مساوی

example

```
2>33  → False
2<34 → True
2<>34 → true
2=34 → false
```

می‌توان خروجی عباراتی با این عملگرها را در متغیر Boolean ذخیره کرد.

```
Dim b as Boolean = 2<=2
B=2<>5
```

عملگرهای منطقی :

نتیجه‌ی عباراتی با عملگرهای منطقی جوابی‌ست همواره منطقی.

عملگر NOT : مقدار مخالف ارزش عملوند را بر می‌گرداند. در صورتی که شرط مورد نظر ارزش درست را داشته باشد، NOT آن مقدار نادرست را بر می‌گرداند و برعکس.

نتیجه‌ی عبارات با عملوند NOT:

X	Not (X)
T	F
F	T

عملگرهای ANDALSO یا AND:

عملگر AND در صورت درست بودن تمامی عملوندها مقدار TRUE را می‌دهد. نتیجه‌ی عملگر ANDALSO نیز همانند عملگر AND است با این تفاوت که اگر اولین عملوند ارزش False داشته باشد مقدار False را می‌دهد و دیگر دومین عملوند را بررسی نمی‌کند. بنابراین سرعت آن بیشتر از عملگر AND می‌باشد.

نتیجه‌ی عبارات با عملگر ANDALSO یا AND:

X	Y	X AND Y	X ANDALSO
F	F	F	F
F	T	F	F
T	F	F	F
T	T	T	T

عملگرهای ORELSE و OR:

عملگر OR درصورت درست بودن یکی از عملوندها، مقدار TRUE را می‌دهد. نتیجه‌ی عملگرORELSE همانند عملگر OR است با این تفاوت که اگر اولین عملوند ارزش True داشته باشد مقدار True را می‌دهد و دیگر دومین عملوند را بررسی نمی‌کند و سرعت آن بیشتر از عملگر OR می‌باشد.

X	Y	X OR Y	X ORELSE Y
F	F	F	F
F	T	T	T
T	F	T	T
T	T	T	T

نتیجه‌ی عباراتی با عملگر XOR:

در صورتی که تعداد عملوندهای صحیح، فرد باشد خروجی آن TRUE و در غیر اینصورت FALSE می‌باشد:

X	Y	X XOR Y
F	F	F
F	T	T
T	F	T
T	T	F

اولویت عملگرها :

عملگرها نسبت به یکدیگر دارای تقدم و اولویت هستند. در عباراتی که شامل چندین عملگر می‌باشند، این مسئله حائز اهمیت است. اگر در یک عبارت چند عملگر هم اولویت باشند عملیات از چپ به راست صورت می‌گیرد.

ترتیب اولویت عملگرها به شکل زیر است:

۱- توابع و پرانتزها
۲- توان
۳- علامت مثبت (+) و منفی (-) و not
۴- ضرب (*) و تقسیم اعشاری (/)
۵- تقسیم صحیح (\)
۶- Mod (باقی مانده تقسیم)
۷- جمع و منها
۸- عملگرهای مقایسه‌ای شامل : =,<>,=>,=<,>,<

دستورهای شرطی :

هر برنامه‌ای ممکن است سه نوع ساختار کنترلی داشته باشد تا مراحل اجرای آن را کنترل نمایند. این سه نوع ساختار عبارتند از : ترتیب – تصمیم – تکرار

ساختار دستور IF :

هنگامی که نیاز به تصمیم گیری باشد از دستورات شرطی استفاده می‌شود.
در این ساختار e یک عبارت یا متغیر شرطی است.

```
If e Then
دستور یا دستورات 1
Else
دستور یا دستورات 2
End If
```

وجود قسمت ۲ کاملاً اختیاری و بر حسب نیاز می باشد.

برنامه‌ای بنویسید که دو عدد بگیرد و اگر مجموع آنها کمتر از ۱۰ بود "One" را چاپ کند و در غیر این صورت "More" را چاپ نماید.

```
Module Module1
   Sub Main()
      Dim a As Byte
      Dim b As Byte
      Console.Write("Please type First Number:")
      a = Console.ReadLine
      Console.Write("Please type Next Number:")
      b = Console.ReadLine
      If a + b < 10 Then
         Console.Write("One")
      Else
         Console.Write("More")
      End If
      Console.ReadLine()
   End Sub
End Module
```

تعریف متغیرها

چاپ پیغام

مقدار دهی متغیر

ساختار دستور IF

برای اینکه تا Enter نزده‌اید پنجره بسته نشود.

برنامه‌ای بنویسید که اگر اعداد بزرگتر از ۱۰ در متغیر قرار داده شود، پیغام long و در غیر این صورت پیغام "small" را نمایش دهد (با استفاده از یک متغیر منطقی).

```
   Sub Main()
      Dim A As Byte
      Dim b As Boolean
      A = Console.ReadLine
      b = A > 10
      If  b=True Then
         Console.WriteLine("long")
      Else
         Console.WriteLine("small")
      End If
      Console.ReadLine()
   End Sub
End Module
```

ساختار If – Else نردبانی :

یک ساختار معروف به شرح زیر است:

```
If  e1  then
 S1
Else
    If  e2  then
     S2
    Else
        If  e3  then
         S3
        End If
    End If
End If
```

ساختار If-Elseif :

با استفاده از این روش ساختار قبلی را به آسانی می‌توان پیاده سازی کرد. همچنین استفاده از این ساختار خوانایی برنامه را نیز افزایش می‌دهد.

```
If  e1  then
S1
ElseIf  e2  then
S2
ElseIf  e3  then
S3
End if
```

program

برنامه‌ای بنویسید که یک عدد را از ورودی گرفته و اگر برابر ۱۰ بود چاپ کند "Ten"، اگر برابر ۲۰ بود چاپ کند "Twenty" و اگر برابر ۳۰ بود چاپ کند "Thirty" و از برنامه خارج شود.

```
Module Module1
   Sub Main()
      Dim a As Byte
      a = Console.ReadLine
      If a = 10 Then
         Console.WriteLine("Ten ")
      Else
         If a = 20 Then
            Console.WriteLine("Twenty ")
         Else
            If a = ۳۰ Then
               Console.WriteLine("Thirty ")
               End                    } ——————————▶ دستورخروج از برنامه
            End If
         End If
      End If
      Console.ReadLine()
   End Sub
End Module
```

دستور End سبب خروج از برنامه می‌شود.

دستور Case :

زمانی که چندین شرط داشته باشیم یا به عبارتی بیش از سه دستور If بخواهیم به کار ببریم منطقی‌تر آن است که از دستور Case به شکل کلی زیر استفاده نماییم:

```
Select Case e
Case مقدار مورد نظر
دستور یا دستورات
Case مقدار مورد نظر
دستور یا دستورات
Case Else
دستور یا دستورات
End Select
```

```
Dim i As Integer
  i = Console.ReadLine
    Select Case i
      Case 1
        Console.Write("Rayane")
      Case 2
        Console.Write("Markazi")
      Case 3
        Console.Write("VB.NET")
      Case Else
         Console.Write("Linq")
    End Select
```

درصورتی که هیچ کدام از مقادیر ۱یا ۲ یا ۳ نباشد مقدار "Linq" چاپ می‌شود.

در صورت تمایل می‌توانید از الگوهای زیر نیز استفاده کنید.

بررسی اعداد در محدوده‌ای معین:

Case محدوده پایین To محدوده بالا

```
Case 1 To 22
```

جهت تعیین مقدار در وضعیت مورد نظر:

Case is شرط

Case Is <> 22	در صورتی که مخالف عدد ۲۲ باشد.
Case Is > 22	در صورتی که برابربا عدد ۲۲ باشد.

حلقه‌های تکرار :

به کمک حلقه‌ها می‌توان تعدادی دستورالعمل را به دفعات معین و یا تا پدید آمدن شرایطی خاص تکرار کرد. حلقه‌ها را می‌توان به کمک دستور if و دستور پرش (goto) ساخت. اما تاکید زبان‌های ساخت یافته بر عدم استفاده از این نوع حلقه‌هاست.

حلقه For (با تعداد تکرار مشخص) :

برای تکرار تعدادی دستور به دفعات معین از حلقه For استفاده می‌شود. فرم کلی حلقه For به دو شکل زیر است:

الف) تعریف متغیر شمارنده قبل از حلقه:

```
Dim Counter as نوع متغیر
For Counter = مقدار اولیه To مقدار نهایی [Step مقدار]

    دستور یا دستورات حلقه

Next Counter
```

ب) تعریف متغیر شمارنده در ساختار حلقه:

```
For Counter as نوع متغیر = مقدار اولیه To مقدار نهایی [Step مقدار]

    دستور یا دستورات حلقه

Next Counter
```

Counter : شمارنده حلقه می‌باشد.

بدیهی است که در شکل ب پس از پایان کار حلقه متغیر شمارنده از بین خواهد رفت.

- در صورتی که مقدار اولیه در حلقه کوچکتر از مقدار نهایی باشد، به این حلقه، حلقه‌ی افزایشی گفته می‌شود.

```
For i=1 to 10
  Console.WriteLine(i)
Next i
```

در این مثال اعداد از ۱ تا ۱۰ به ترتیب نمایش داده می‌شوند.

- در صورتی که مقدار اولیه در حلقه بزرگتر از مقدار نهایی باشد، به این حلقه، حلقه‌ی کاهشی گفته می‌شود.

```
For i=10 to 1 step -1
  Console.WriteLine(i)
Next i
```

در این مثال اعداد از ۱۰ تا ۱ نشان داده می‌شوند. در حلقه‌های کاهشی مقدار Step عددی منفی می‌باشد.

درحلقه افزایشی اگر مقدار نهایی از مقدار اولیه کوچکتر باشد حلقه اصلا اجرا نمی‌شود. به همین ترتیب در حلقه for کاهشی نیز اگر مقدار نهایی بزرگتر از مقدار اولیه باشد حلقه اجرا نمی‌شود. در این مورد خطایی هم صورت نمی‌گیرد و فقط کنترل اجرای برنامه به خط بعد از حلقه می‌رود.

اگر مقدار اولیه شمارنده حلقه For با مقدار نهایی آن برابر باشد (چه حلقه افزایشی باشد چه کاهشی) دستورات درون این حلقه فقط یک بار اجرا می‌شود.

در VB.NET هنگامی که حلقه تمام می‌شود شمارنده حلقه یکی بیشتر از مقدار نهایی حلقه است.

مقدار نهایی حلقه فقط یکبار آنهم در ابتدای حلقه مشخص می‌شود اما مقدار اولیه حلقه در گردش‌های بعدی تغییر خواهد کرد، بنابراین مقدار نهایی حلقه در بدنه حلقه قابل تغییر نخواهد بود اما مقدار اولیه حلقه در بدنه قابل تغییر است.

در صورت آنکه قصد داشته باشید تعداد تکرار یک حلقه در زمان اجرای برنامه تعیین شود به دو شکل این عمل صحیح خواهد بود:

```
Dim k as integer
K=console.readline()
For I as integer=0 to k
دستورات
Next
```

```
For I as integer=0 to console.readline()
دستورات
Next
```

همانطور که در نکته‌ی قبل گفتیم، مقدار نهایی حلقه فقط یکبار تعیین شده و بنـابراین دسـتور ورودی فقط یکبار اجرا می‌شود.

برنامه ای بنویسید که اعداد ده تا بیست را چاپ کند.

```
Module Module1
   Sub Main()
      Dim a As Byte
      For a = 10 To 20
         Console.WriteLine(a)
      Next
      Console.ReadLine()
   End Sub
End Module
```

شمارنده حلقه یکی‌یکی افزایش می‌یابد و ده بار عدد a را نمایش می‌دهد.

تعریف متغیر شمارنده در ساختار حلقه:

```
For i As نوع متغیر = مقدار اولیه To مقدار نهایی [Step مقدار]
دستور یا دستورات حلقه
Next
```

```
For i As Byte = 0 To 10 Step 2
  Console.WriteLine("hello")
Next
```

پس از پایان حلقه دیگر امکان استفاده مجدد از این متغیر شمارنده وجود ندارد.

حلقه While :

در نوع دیگری از حلقه‌ها (به نام حلقه‌های شرطی) تعداد دفعات تکرار معلوم نیست و حلقه تا پدید آمدن شرط خاصی تکرار می‌شود، هر چند که گفته شد می‌توان این حلقه‌ها را با if و پرش goto ساخت، اما راحت تر این است که از حلقه While به شکل کلی زیر استفاده کنیم:

```
While e
دستور یا دستورات حلقه
End While
```

e : در این ساختار یک عبارت یا متغیر شرطی است.

progrom

برنامه‌ای بنویسید که پیغام hello را ۱۰ بار چاپ کند (از حلقه while استفاده کنید).

```
Module Module1
  Sub Main()
    Dim i As Byte
    While i <> 10
      Console.WriteLine("hello")
      i += 1
    End While
    Console.ReadLine()
  End Sub
End Module
```

ساختار حلقه While

تا زمانی که I مخالف ده می باشد

شمارنده حلقه

progrom

برنامه ای بنویسید که تا عدد ۱۷ وارد نشده است برنامه بسته نشود.

```
Module Module1
  Sub Main()
    Dim a As Byte
    While a <> 17
      a = Console.ReadLine
    End While
  End Sub
End Module
```

در ابتدای حلقه a مخالف ۱۷ است و به همین دلیل تا زمانی که عدد ۱۷ وارد نشده است، دستور درون حلقه (a=Console.Readline) تکرار می‌شود. در نتیجه نیازی به استفاده از دستور Console.Readline نیست و a مرتباً مقادیر ورودی را در خود جای می‌دهد.

آرایه چیست؟

تعدادی متغیر از یک نوع و تحت یک نام می‌باشند. هر یک از متغیرهای درون آرایه با یک شماره که به آن اندیس[۹] می‌گوییم شناسایی می‌شود. متغیرهای درون آرایه را عناصر آرایه می‌نامیم.

شکل کلی تعریف آرایه به صورت زیر است:

```
Dim نام آرایه (تعداد اندیس) As نوع متغیر
```

example

```
Dim a (4) as Byte
```

همانطور که می‌دانید شماره اندیس آرایه از صفر شروع می‌شود، پس دقت داشته باشید که تعداد عناصر آرایه یکی بیشتر از اندیسی است که شما داده‌اید.

برای دسترسی به خانه‌های آرایه از الگوی زیر پیروی کنید:

```
نام آرایه(شماره اندیس)
```

example

```
a (0)
a (1)
```

برای مقداردهی به هر خانه‌ی آرایه از الگوی زیر پیروی کنید:

```
نام آرایه(شماره اندیس) = "مقدار مورد نظر"
```

[۹] Index.

example

```
a (0)=1
a (2)=20
```

۱		۲۰		

دراینجا نیز باید اصول مقداردهی رعایت شود. چرا که هر خانه در واقع یک متغیر است.

progress

برنامه‌ای بنویسید که اعداد ۱ تا ده را در یک آرایه قرار داده و نمایش دهد.

```
Dim a(9) As Byte
Dim i As Byte
      For i = 0 To 9
         a(i) = i + 1
    Console.Write(a(i))
Next
```

آرایه بدون پهنای باند :

برای تعریف آرایه بدون پهنای باند به شکل زیر عمل می‌کنیم:

```
Dim نام آرایه () As نوع متغیر
```

example

```
Dim a() As Byte
```

جهت مقداردهی به این نوع آرایه‌ها از شکل زیر استفاده می‌شود:

```
Dim نام آرایه () As نوع متغیر = {مقداراول و مقداردوم...}
```

example

```
Dim A( ) As Byte = {10, 20, 9, 6, 4}
```

حال با توجه به آنکه پنج مقدار به آرایه‌ی بدون پهنای باند داده‌ایم، این آرایه حاوی پنج عنصر و دارای اندیسهای ۰ تا ۴ می‌باشد.

در VB10 آرایه‌های بدون پهنای باند را به شکل زیر نیز می‌توان تعریف نمود:

```
Dim نام آرایه = {مقدار۱ ,مقدار۲ ,مقدار۳ , ...}
```

example

```
Dim a = {1, 2, 3}
```

جهت اطلاع از طول آرایه های بدون پهنای باند، می توان از متد Count استفاده نمود.

example

```
Console.Write(a.Count)
```

در زبانهای VB9 و ما قبل از آن در صورتی که نوع متغیر یا آرایه همانند بالا مشخص نشود، به صورت خودکار، نوع آن Object در نظر گرفته می‌شود و این در حالی است که نوع آرایه‌ی بالا در vb10 از نوع Integer خواهد بود.

در موارد دیگر به مثالهای زیر توجه کنید:

```
Dim b = {1, 2, 3.5}          'از نوع Double تعریف می‌شود
Dim c = {"1", "2", "3"}      'از نوع String تعریف می‌شود
Dim d = {1, "A"}             'از نوع Object تعریف می‌شود
```

حال با استفاده از یک حلقه‌ی For به عناصر این آرایه دسترسی پیدا می‌کنیم:

```
For i As Byte = 0 To 4
   Console.WriteLine(A(i))
Next
```

حلقه For Each :

این حلقه جهت گردش در یک مجموعه به کار می‌رود، به این طریق که در هر بار گردش حلقه، به ترتیب تک تک اعضا در شمارنده حلقه قرار می‌گیرد.

به شکلی کلی زیر:

```
For Each شمارنده in مجموعه
NEXT
```

example

```
Dim A() As Integer = {10, 20, 9, 6, 4}

    For Each d In A
      Console.WriteLine(d)
    Next
```

در گردش اول مقدار ۱۰ در شمارنده d و در گردش دوم مقدار ۲۰ در شمارنده d قرار می‌گیرد و در ادامه نیز اینکار مرتب انجام می‌شود تا آنکه مقدار ۴ در متغیر d قرار گرفته و کار حلقه به پایان می‌رسد.

در اکثر مواقع نیاز نیست متغیر شمارنده از قبل تعریف شده باشد، زیرا شمارنده به صورت خودکار هم نوع با مجموعه تعریف می‌شود. اما در صورت نیاز متغیر شمارنده را به دو شکل زیر می‌توان تعریف کرد:

الف) در ساختار حلقه:

```
For Each نام متغیر شمارنده As نوع متغیر شمارنده In مجموعه
Next
```

example

```
Dim s() As String = {"من ","ی","ا","د","خ"}

 For Each j As String In s
    Console.WriteLine(j)
Next
```

تک تک اعضای مجموعه‌ی S که از نوع رشته هستند نمایش داده می‌شود.

ب) قبل از تعریف بدنه حلقه:

```
Dim s() As String = {"کی ","تا","کجا","جور","چه"}
 Dim j As String
 For Each j In s
   Console.WriteLine(j)
 Next
```

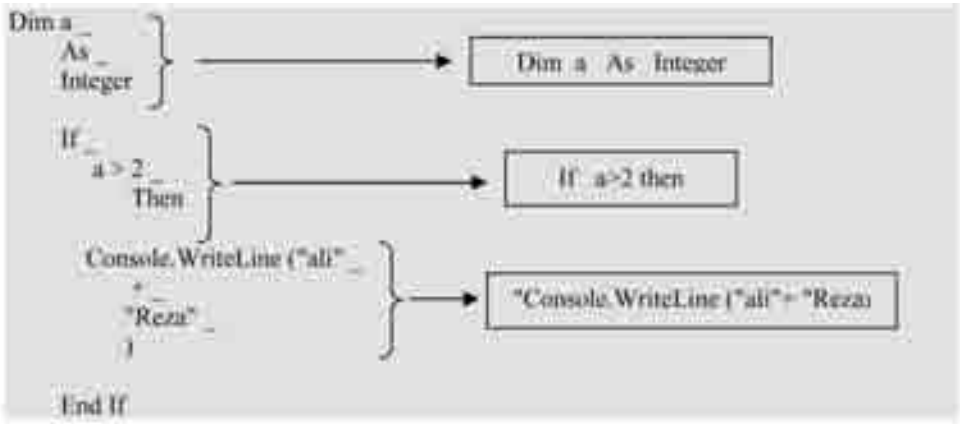
شکستن دستورات :

درصورتی که دستورات طولانی شوند و قصد داشته باشید یک سری دستورات بهم پیوسته را در چند خط بشکنید، می‌توانید از علامت _ استفاده نمائید، برای درک این موضوع به مثال‌های زیر توجه نمائید.

ابتدا کدهای زیر را در نظر بگیرید:

```
Dim a As Integer
    If a > 2 Then
      Console.WriteLine("ali" + "Reza")
    End If
```

حال کدهای بالا را به چندین خط تقسیم می‌کنیم:



ادامه خطوط به صورت ضمنی:

در VB10(Visual Studio 2010) این امکان پدید آمده است که در برخی از قطعه دستورات بی آنکه علامت _ را استفاده نمود بتوان کدها را در چندین خط شکست.

علائم و کلمات کلیدی‌که‌قبل یا بعد از آن می‌توان‌خطوط را بی‌آنکه از علامت‌زیر خط (_) استفاده نمود شکست، به شرح زیر است:

علائم	قبل	بعد	علائم	قبل	بعد
,		X	/ Operator		X
(		X	\ Operator		X
)	X		Mod Operator		X
{		X	+ Operator (unary and binary)		X
}	X		^= Operator		X
<%=		X	*= Operator		X
%>	X		/= Operator		X
<		X	\= Operator		X
>	X		+= Operator		X
Aggregate	X	X	-=Operator		X
Distinct	X	X	&= Operator		X
From (in a query)	X	X	< Operator		X
From (in a collection initializer)		X	<= Operator		X
Group By	X	X	> Operator		X
Group Join	X	X	>= Operator		X
Join	X	X	= Operator		X
Let	X	X	<> Operator		X
Order by	X	X	Is Operator		X
Select (in query context)	X	X	IsNot Operator		X
Skip	X	X	Like Operator		X
Skip While	X	X	& Operator		X
Take	X	X	And Operator		X
Take While	X	X	Or Operator		X
Where	X	X	Xor Operator		X
In	X	X	AndAlso Operator		X
Into	X	X	OrElse Operator		X
On	X	X			
Ascending	X	X			
Descending	X	X			
^ Operator		X			
* Operator		X			

به مثال‌های زیر توجه نمائید:

```
For Each j
In
  A
```

```
For Each j In A
```

```
Console.Write(
      "Ali" &
      "Reza"
      )
```

```
Console.Write("Ali" & "Reza")
```

توضیحات (comment) :

برنامه نویسان معمولا در قسمت‌های مختلف کدهایشان توضیحاتی را جهت یادآوری عملکرد آن قطعه کد و یا جهت خوانایی بیشتر برای سایر برنامه نویسان به برنامه اضافه می‌کنند.

این توضیحات گاهاً می‌تواند یک قطعه کد ساده نیز باشد.

ابتدا یک تک کتیشن(') تایپ کرده و سپس توضیحات مورد نظر را تایپ می‌نمائیم.

به شکل زیر:

```
'توضیحات
```

```
'dar in ghesmat process marbot be search anjam mishavad
'in ghesmat hamanand code For I as byte=0 to 5  amal khahad kard
```

توضیحات توسط مترجم کامپایل نمی‌شود.