به نام خدا

جزوه برنامه سازی پیشرفته 2

فاطمه احمدی

فهرست مطالب

۱- مقدمه

به #C و دنیایی از برنامه سازی تحت ویندوز، وب و اینترنت با ویژوال استودیو و پلتفرم .NET. خوش آمدید. این جزوه برگرفته از کتب مختلف[۱]، برنامه سازی #C را در سطح پایه ای و مقدماتی ارائه می دهد.

#C در سیر تکامل C و C++ ارائه شده است و صراحتا برای پلتفرم .NET. مایکروسافت توسعه یافته است. #C ویژگی‌های مهمی را برای برنامه نویسان فراهم می‌کند مانند برنامه سازی شی‌گرا[۲]، رشته‌ها[۳]، گرافیک[۴]، اجزای رابط گرافیکی کاربر[۵]، اداره کردن استثناها[۶]، برنامه سازی چندنخی[۷]، چندرسانه ای[۸]، پردازش فایل[۹]، پردازش پایگاه داده[۱۰]، برنامه سازی شبکه و اینترنت[۱۱] و محاسبات توزیعی[۱۲].

برنامه نویسان دستورات برنامه را در زبانهای برنامه سازی متفاوتی می نویسند. تعدادی از زبانهای برنامه سازی با کامپیوترها قابل فهم اند، اما تعدادی از آنها به مراحلی از ترجمه نیاز دارند. امروزه صدها زبان برنامه سازی استفاده می شود، به طور کلی زبانها را در سه دسته زیر میتوان دسته بندی نمود:

- **زبانهای ماشین**: هر کامپیوتر می تواند فقط زبان ماشین خود را به طور مستقیم بفهمد که بوسیله سخت افزار آن طراحی می شود. زبانهای ماشین به طورکلی شامل توالی از اعداد هستند که در نهایت به ۰ و ۱ تبدیل می شوند. زبانهای ماشین وابسته به ماشین هستند، به عبارت دیگر هر زبان ماشین خاص روی یک نوع کامپیوتر استفاده می شود.
- **زبانهای سطح پایین (اسمبلی)**: به جای استفاده از رشته ای از اعداد که کامپیوترها می توانند مستقیما بفهمند، برنامه نویسان از اختصارنویسی به زبان انگلیسی استفاده کردند که عملیات ابتدایی کامپیوتر را ارائه دهند. زبانهای اسمبلی جزء این دسته از زبانها هستند که زبان سمبلیک زبان ماشین نامیده می شوند.

---

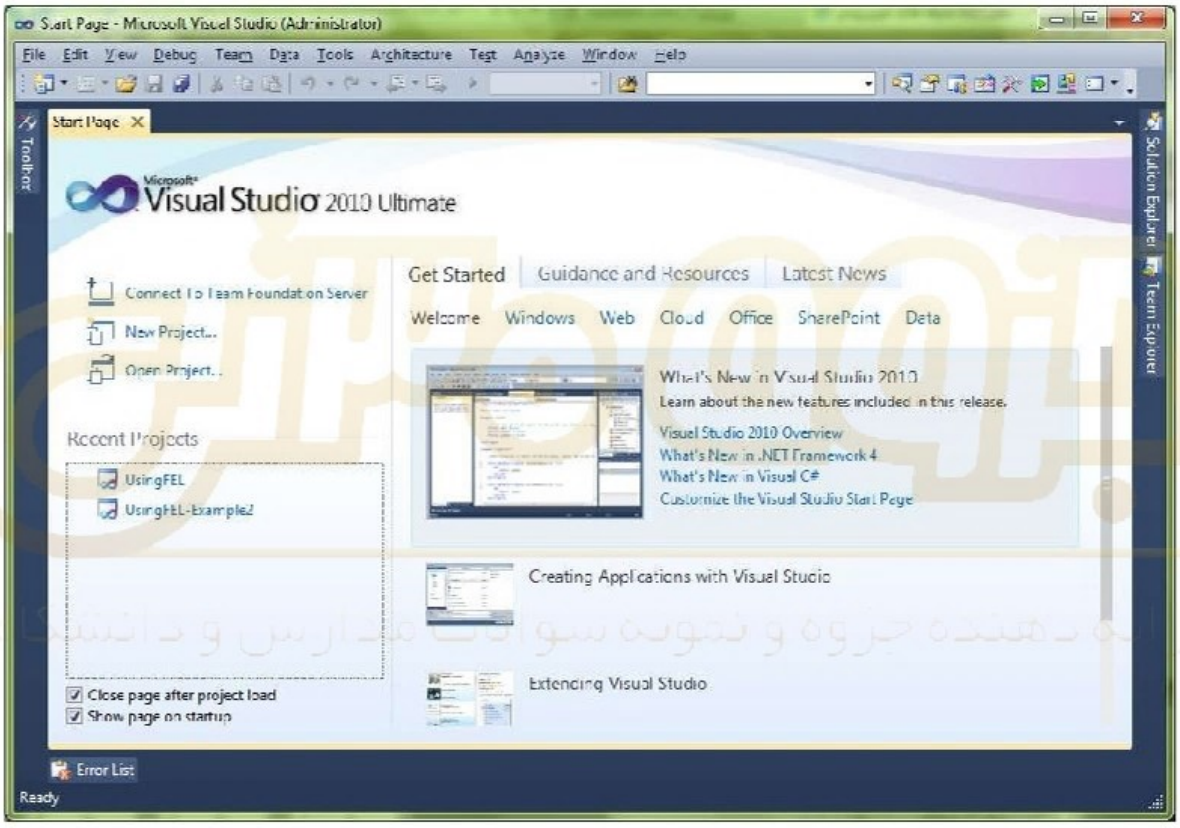
[۱] بخصوص برگرفته از کتاب "C# How to program" نوشته Dietel and Deitel.

[2] Object-Oriented Programming (OOP)
[3] strings
[4] graphics
[5] Graphical User Interface (GUI)
[6] exception handling
[7] multithreading programming
[8] multimedia
[9] file processing
[10] database processing
[11] network and Internet programming
[12] distributed computing

- **زبانهای سطح بالا:** برنامه نویسان زبانهای سطح بالا را به زبانهای ماشین و اسمبلی ترجیح می دهند زیرا این زبانها به زبان انسان نزدیکند. اما تبدیل زبان سطح بالا به زبان ماشین توسط مترجم ها صورت می گیرد بنابراین اجرای این برنامه ها از برناهه های دسته های دیگر زمانبرتر است.

۲- انواع برنامه ها به زبان C#

شکل زیر محیط برنامه سازی ویژوال استودیو (نسخه ۲۰۱۰) را نشان می دهد.

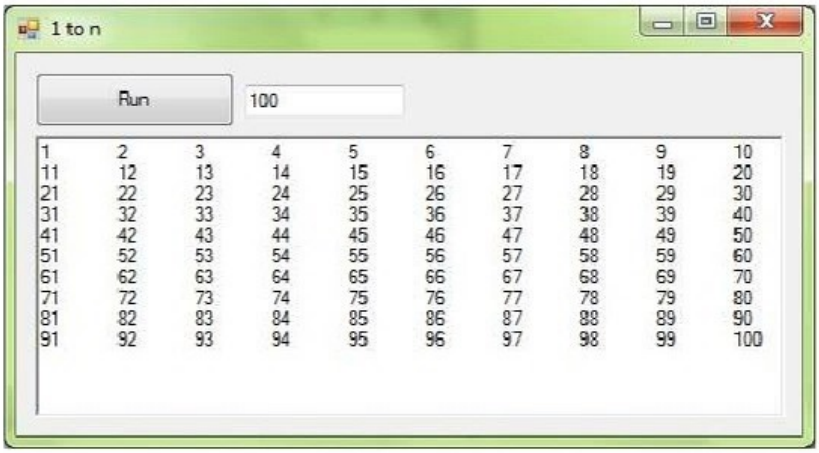


به کمک زبان C# می توان برنامه هایی با انواع متفاوت خروجی ها ایجاد کرد. با توجه به نوع خروجی برنامه، به طور کلی می توان برنامه های C# را به دو دسته تقسیم کرد:

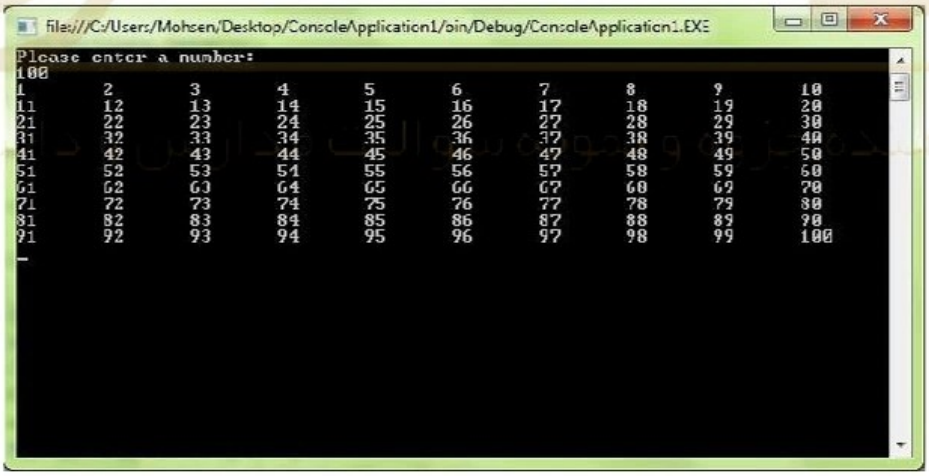
۱. برنامه‌های کاربردی ویندوز[1]: خروجی این برنامه ها می تواند یک تصویر، نمودار، جدول و غیره باشد. تعامل کاربر با برنامه از طریق فرمها انجام می شود که تعدادی کنترل دیداری (دکمه، ابزار متن، منو و ...) روی آنها قرار گرفته

[1] windows application

است. شکل زیر مثالی از خروجی یک برنامه کاربردی ویندوز است که اعداد ۱تا عدد موردنظر کاربر را روی فرم نشان می دهد.



۲. برنامه‌های کاربردی کنسول[1]: این برنامه ها خالی از کنترل های دیداری هستند. خروجی این برنامه ها متن است که در پنجره کنسول نمایش داده می شود و تعامل کاربر با برنامه از طریق متن و پنجره کنسول انجام می گیرد. این نوع خروجی در ویندوزهای ۹۵/۹۸، MS-DOS prompt و در ویندوزهای NT/2000/XP/Vista/Seven ، command prompt نامیده می شود. شکل زیر خروجی یک برنامه کاربردی کنسول است که اعداد ۱ تا عدد موردنظر کاربر را روی پنجره کنسول نمایش می دهد.



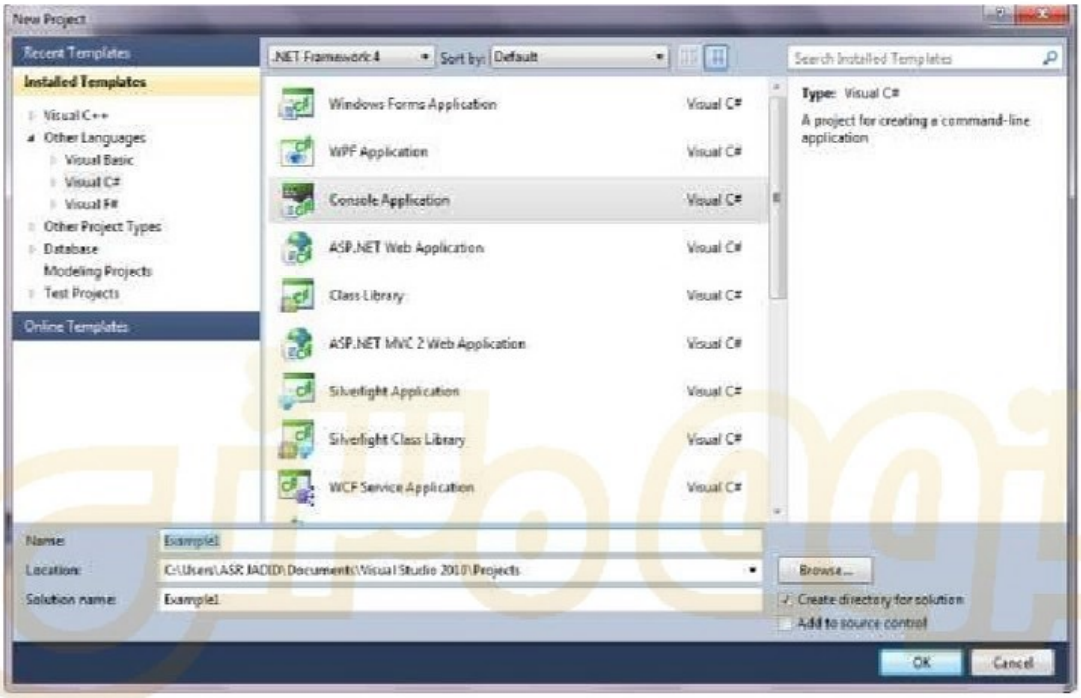
۳. چارچوب یک برنامه کاربردی کنسول

به کمک مراحل زیر یک برنامه کاربردی کنسول در ویژوال استودیو ایجاد و اجرا می کنیم.

۱- ایجاد برنامه کاربردی کنسول

[1] console application

به منوی file بروید و New را انتخاب کنید، سپس Project را انتخاب کنید، یک دیالوگ ظاهر می شود در سمت چپ، از قسمت other languages، Visual C# و در سمت راست Console Application را انتخاب کنید. میتوانید در قسمت پایین، اطلاعاتی مانند نام (مانند Example1) و محل پروژه را تغییر دهید. در نهایت دکمه OK را برای ایجاد پروژه کلیک نمایید.



۲- در متد Main کد مورد نظر را بنویسید. مانند زیر:

```
Console.WriteLine("Welcome to C# Programming.");
```

۳- برنامه را اجرا کنید.

برای اجرا برنامه به منوی Build بروید و گزینه Build Solution را انتخاب کنید. اگر برنامه خطای گرامری نداشته باشد می توانید آن را اجرا کنید. برای اجرا از منوی Debug گزینه Start Without Debugging را انتخاب کنید.

انتخاب Debug > Start Without Debugging باعث می شود که پنجره کنسول از کاربر بخواهد که برای خاتمه اجرای برنامه دکمه ای را فشار دهد و قبل از آن به کاربر اجازه دهد خروجی برنامه را مشاهده کند. درغیر این صورت اگر برنامه را با انتخاب Debug > Start اجرا کنید همین که پنجره کنسول باز می شود و پیام را چاپ می کند فورا بسته می شود.

در آینده خواهید دید که روش دیگر برای جلوگیری از بسته شدن فوری پنجره کنسول استفاده از متد ReadKey (از کلاس Console) می باشد که به صورت زیر در انتهای دستورات نوشته می شود. متد ReadKey برای دریافت یک کلید از ورودی استفاده می شود.

```
Console.Readkey();
```

بنابراین برنامه زیر یک مثال ساده از برنامه سازی C# از سری برنامه های کاربردی کنسول است که جمله "Welcome to C# Programming." را در پنجره کنسول چاپ می کند. (در این جزوه، خروجی تمام برنامه ها در کادر سبزرنگ مانند زیر، نمایش داده می شود).

```
1  // Fig. 3.1: Welcome1.cs
2  // A first program in C#.
3
4  using System;
5
6  class Welcome1
7  {
8     static void Main( string[] args )
9     {
10       Console.WriteLine( "Welcome to C# Programming!" );
11    }
12 }
```

```
Welcome to C# Programming!
```

خط ۱ و ۲ برنامه که با // شروع شده اند، توضیح هستند و دستور اجرایی نمی باشند و برای افزایش خوانایی برنامه اضافه شده اند.

نکته: برای اضافه کردن توضیحات تک خطی (همانند زبان C++)، از // (جفت slash) در ابتدای خط استفاده می شود. برای اضافه کردن توضیحات چندخطی به برنامه (همانند زبان C)، به جای استفاده از // در ابتدای هر خط، می توان از /* در ابتدای متن و */ در انتهای متن استفاده کرد. مانند زیر:

```
/* This is a multiple-line
   comment. It can be
   split over many lines */
```

نکته: توضیحات در برنامه سبزرنگ می باشند.

خط ٤ با ویژوال استودیو ایجاد شده است و نشان می دهد که برنامه از ویژگیهای فضای نام[1] system استفاده می کند. ویژگی های C# در گروههایی به نام "فضای نام" دسته بندی می شوند که بوسیله قالب کاری .NET[2] ایجاد شده اند. یکی از ویژگیها در فضای نام system، console می باشد که برای ورود و خروج اطلاعات از برنامه از متدهای آن استفاده می شود.

دستورات در خطوط ٦ تا ١٢ متعلق به کلاس welcome1 هستند.

نکته: یک برنامه C# شامل کلاسها و متدها است که یا توسط برنامه نویس ایجاد شده اند یا از کتابخانه کلاس قالب کار .NET. استفاده شده اند. هر کلاس در خود اعضایی دارد، متد هم می تواند عضوی از یک کلاس باشد. هر متد، کاری را انجام می دهد و اطلاعاتی را برمی گرداند.

آکولاد چپ ({) در خط ٧، شروع بدنه تعریف کلاس Welcome1 است و آکولاد راست (}) در خط ١٢، پایان تعریف کلاس Welcome1 است.

نکته: هر کلاس و متد با یک آکولاد چپ ({) شروع می شود و با یک آکولاد راست (}) خاتمه می یابد.

خط ٨ هم در برنامه کاربردی کنسول و هم در برنامه کاربردی ویندوز وجود دارد. این برنامه های کاربردی، اجرا را در تابع Main شروع می کنند. آکولاد چپ ({) در خط ٩، شروع بدنه متد Main را نشان می دهد و آکولاد راست (}) بدنه متد Main را خاتمه می دهد. پرانتزهای باز و بسته بعد از تابع Main نشان دهنده این است که Main یک متد است. بدون تابع Main برنامه های کاربردی C# قابل اجرا نمی باشند.

خط ١٠، به کامپیوتر دستور می دهد که کاراکترهایی که مابین دابل کوتیشن[3] (") قرار دارند را چاپ کند. به این کاراکترها، رشته یا رشته ای از کاراکترها گفته می شود. کلاس Console به برنامه این توانایی را می دهد که اطلاعاتی را به خروجی استاندارد کامپیوتر بفرستند یا اطلاعاتی را از ورودی دریافت کند این کار را با استفاده از متدها انجام می دهد. در زیر توابع ورودی و خروجی مهم معرفی می شوند.

٤- توابع ورودی/خروجی

توابع ورودی/خروجی مهمی که در کلاس Console از فضای نام System قرار دارند عبارتند از:

---

[1] namespace
[2] .NET Framework
[3] double quotation

- Write: نوشتن در خروجی
- WriteLine: نوشتن در خروجی و رفتن به خط بعد
- Clear: پاک کردن صفحه
- Read: خواندن یک کاراکتر از ورودی
- ReadLine: خواندن یک خط به صورت یک رشته
- ReadKey: خواندن یک کلید

متد WriteLine از کلاس Console که با استفاده از علامت . (dot) به صورت Console.WriteLine فراخوانی می شود، متنی(رشته) که به عنوان ورودی به آن داده شده است را در پنجره کنسول نمایش می دهد. همانطور که در بالا ذکر شد، علاوه بر متد WtileLine، متد Write نیز برای چاپ اطلاعات در خروجی استفاده می شود. اما تفاوت دو متد در این است که متد WriteLine بعد از چاپ اطلاعات، نشانگر را در شروع خط بعد قرار می دهد. اما متد Write اینگونه نیست و نشانگر را به خط بعد نمی فرستد به عبارت دیگر کاراکترهای بعدی بلافاصله بعد از کاراکترهای چاپ شده چاپ خواهند شد.

بنابراین می توان رشته "Welcome to C# Programming." را به کمک دو متد Write و WriteLine به صورت زیر، در یک خط چاپ نمود.

```
1  // Fig. 3.4: Welcome2.cs
2  // Printing a line with multiple statements.
3
4  using System;
5
6  class Welcome2
7  {
8     static void Main( string[] args )
9     {
10       Console.Write( "Welcome to " );
11       Console.WriteLine( "C# Programming!" );
12    }
13 }
```

```
Welcome to C# Programming!
```

نکته: تمام خط ۱۰، از ابتدا تا انتها یک دستور نامیده می شود. دقت کنید که هر دستور در زبان C# به نقطه ویرگول[1] (;) ختم می شود.

می توان تنها با یک دستور، چندین خط را در خروجی ایجاد کرد. این کار با استفاده از کاراکترهای newline (\n) انجام می شود. استفاده از "\n" در رشته ورودی تابع Write یا WriteLine باعث می شود که به محض رسیدن اجرا به این رشته، نشانگر به شروع خط بعد برود و کاراکترهای بعدی در آن خط چاپ شوند. مثال زیر کاربرد \n را به خوبی نشان می دهد. با استفاده کردن از \n بعد از هر کلمه باعث شدیم هر کلمه در یک خط جداگانه چاپ شود.

```
1  // Fig. 3.5: Welcome3.cs
2  // Printing multiple lines with a single statement.
3
4  using System;
5
6  class Welcome3
7  {
8     static void Main( string[] args )
9     {
10       Console.WriteLine( "Welcome\nto\nC#\nProgramming!" );
11    }
12 }
```

```
Welcome
to
C#
Programming!
```

به کاراکتر \ (backslash) ، escape گفته می شود زیرا به کمک آن می توان یک کاراکتر خاص را به خروجی فرستاد. هنگامی که یک کاراکتر در کنار \ قرار می گیرد به آن escape sequence گفته می شود. به عنوان مثال \n به عنوان یک escape sequence، کاراکتر Newline است. در جدول زیر چندین escape sequence معرفی می شود.

Escape sequence	توصیف
\n	Newline. نشانگر را به ابتدای خط بعد منتقل می کند.
\t	نشانگر را به اندازه tab (سه فاصله) به سمت راست منتقل می کند.
\r	نشانگر را به ابتدای خط جاری منتقل می کند.
\\	کاراکتر \ ، (backslash) را در خروجی چاپ می کند.
\"	کاراکتر " ، (double quotation) را در خروجی چاپ می کند.

[1] Semicolon

۵- اعلان متغیر

همانند C و C++، برای استفاده از متغیرها در برنامه های C# باید آنها را تعریف نمود. متغیر، به محلی از حافظه گفته می شود که مقداری برای استفاده در برنامه در آن ذخیره می شود. بنابراین هر متغیر دارای نام، نوع، اندازه و مقدار می باشد.

نام متغیر: نام متغیرها، شناسه نامیده می شود. هر شناسه ترکیبی از حروف، ارقام، علامت زیرخط (_) و امپرسند (@) می‌باشد که با ارقام شروع نشود و شامل فاصله نباشد. شناسه نمی‌تواند از کلمات کلیدی (رزروی) C# باشد. کلمات کلیدی (Keyword) در جدول زیر معرفی شده است.

C# Keywords

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

نوع متغیر: با توجه به مقداری که به متغیر نسبت داده خواهد شد، از بین انواع داده ای در جدول زیر، یک نوع برای متغیر در نظر گرفته می شود. شکل کلی تعریف متغیر به صورت زیر می باشد:

DataType VariableName;

نوع متغیر نام متغیر ;

که به جای DataType ، یکی از انواع داده ای (از جدول زیر) قرار می گیرد. و به جای VariableName، یک شناسه معتبر قرار می گیرد. دقت کنید که اعلان متغیر با نقطه ویرگول یا سمی کالن (;) خاتمه می یابد.

اندازه متغیر: با توجه به نوع داده ای انتخاب شده، فضایی از حافظه (برحسب بایت) به متغیر تخصیص داده می شود. در جدول زیر برای هر نوع داده ای، اندازه فضای اشغالی در حافظه (در ستون اندازه) مشخص شده است.

مقدار متغیر: مقدار متغیر، داده ای است که در طول اجرای برنامه در محلی از حافظه ذخیره می شود. ممکن است به یک متغیر از ابتدای تعریف آن، مقداری نسبت داده شود و یا در حین اجرای برنامه به آن مقدار داده شود. محدوده مقادیری که به هر نوع داده ای می توان اختصاص داد در جدول زیر (در ستون مقادیر) مشخص شده است.

نماد در C#	توصیف	اندازه (بایت)	مقادیر	Class / Struct
int	صحیح	4	۲- میلیارد تا ۲ میلیارد	System.Int32
uint	صحیح مثبت	4	۰ تا ٤ میلیارد	System.UInt32
short	صحیح کوتاه	2	۳۲۷٦۸- تا ۳۲۷٦۷	System.Int16
ushort	صحیح کوتاه مثبت	2	۰ تا ٦٥٥۳٥	System.UInt16
long	صحیح بلند	8		System.Int64
ulong	صحیح بلند مثبت	8		System.UInt64
float	اعشاری	4		System.Single
double	اعشاری با دقت بالا	8		System.Double
sbyte	بایت علامت دار	1	۱۲۸- تا ۱۲۷	System.SByte
byte	بایت بدون علامت	1	۰ تا ۲٥٥	System.Byte
char	نویسه یا کاراکتر	2		System.Char
bool	بولی (true/false)	1	true یا false	System.Boolean

System.String		*	رشته	*string*
System.Object		*	نوع پایه در C#	*object*

٦- تبدیل نوع

در تبدیل نوع، اگر نوع اولیه کوچکتر از نوع نهایی باشد، به طور ضمنی می توان تبدیل نوع را انجام داد، یعنی به طور مستقیم متغیر نوع اولیه را در متغیر نوع نهایی کپی نمود. مثلا اگر داشته باشیم double d; و int a; می توان براحتی نوشت: d=a . اما عکس آن امکان پذیر نیست، زیرا اگر بخواهیم متغیر اعشاری را در یک متغیر صحیح کپی کنیم قسمت اعشاری آن از بین می رود بنابراین نمی توان آن را به طور ضمنی انجام داد بلکه باید تبدیل نوع صریح انجام گیرد. برای تبدیل نوع صریح نوع نهایی را داخل پرانتز قبل از نوع اولیه قرار می دهیم مانند a= (int) d. به این ترتیب قسمت صحیح عدد اعشاری d را داخل متغیر صحیح a قرار می دهیم و قسمت اعشاری آن از بین می رود.

۷- دریافت اعداد از ورودی

متدهای Read، ReadLine و ReadKey، اطلاعات ورودی را به صورت رشته دریافت می کنند. بنابراین برای دریافت اعداد از ورودی باید به صورت زیر عمل کرد:

۱. استفاده از دستور ReadLine (که عدد را به صورت یک رشته دریافت می کند)
۲. استفاده از تابع Parse یا TryParse نوع داده مورد نظر:

برای تبدیل رشته دریافتی به عدد باید از توابع تبدیل استفاده کرد. تابع parse یک تابع تبدیل است که در کلاسهای انواع داده int، float و غیره تعریف شده است. برای تبدیل به صورت زیر عمل می کنیم:

```
string  s;
int  n;
s = Console.ReadLine();
n = Int.Parse(s);
```

مثال: در برنامه کاربردی کنسول زیر قصد داریم جمع دو عدد که توسط کاربر از طریق صفحه کلید وارد می شود را محاسبه کنیم. همانند قبل یک برنامه کاربردی کنسول ایجاد می کنیم. همانطور که قبلا هم توضیح داده شد هر برنامه #C دارای حداقل یک کلاس است. در این مثال نام کلاس اصلی برنامه Addition می باشد. تابع Main برنامه (که

اجرای برنامه از آن شروع می شود) داخل کلاس Addition وجود دارد. هم کلاس Addition و هم تابع Main با یک آکولاد چپ ({) شروع می شوند و با یک آکولاد راست (}) خاتمه می یابند.

در این مثال نیاز داریم دو متغیر تعریف کنیم که اعداد ورودی توسط کاربر در آنها ذخیره شود سپس آن اعداد را با هم جمع کنیم. دو عدد را از نوع صحیح (int) در نظر می گیریم. بنابراین دو متغیر از نوع صحیح به نام های number1 و numbe2 (مطابق شکل زیر) به کمک نوع صحیح (int) تعریف می کنیم. اما همانطور که قبلا ذکر شد ورودی برنامه توسط متد ReadLine به صورت رشته دریافت می شود. بنابراین باید به ازای هر عدد یک متغیر رشته ای نیز تعریف نماییم و سپس به کمک متد Parse عمل تبدیل رشته به عدد صحیح را انجام دهیم. بنابراین دو متغیر رشته ای به نام های firstNumber و secondNumber (مطابق شکل زیر) به کمک نوع رشته (string) تعریف می کنیم. به این ترتیب هر عدد توسط متد ()ReadLine به صورت رشته از ورودی دریافت می شود و در متغیر رشته ای قرار می گیرد.

```
// prompt for and read first number from user as string
Console.Write( "Please enter the first integer: " );
firstNumber = Console.ReadLine();

// read second number from user as string
Console.Write( "\nPlease enter the second integer: " );
secondNumber = Console.ReadLine();
```

سپس با استفاده از متد Parse از کلاس Int32 به عدد صحیح تبدیل شده و در متغیر صحیح قرار می گیرد:

```
// convert numbers from type string to type int
number1 = Int32.Parse( firstNumber );
number2 = Int32.Parse( secondNumber );
```

حال می توانیم این دو عدد صحیح را با هم جمع نموده و حاصل را در یک متغیر از نوع صحیح (sum) که قبلا تعریف کرده ایم قرار دهیم:

```
// add numbers
sum = number1 + number2;
```

در نهایت حاصلجمع دو عدد را به کمک متد WriteLine از کلاس Console، در خروجی چاپ می نماییم:

```
// display results
Console.WriteLine( "\nThe sum is {0}.", sum );
```

نکته: با توجه به اینکه ورودی متد WriteLine (یا Write) از نوع رشته است. برای چاپ اعداد در خروجی، یک روش این است که برای هر عدد یک {i} در محلی از رشته که می خواهیم عدد چاپ شود قرار دهیم سپس بعد از اتمام رشته ورودی (بعد از دابل کوتیشن) ، یک کاما (,) قرار داده سپس نام متغیری که میخواهیم مقدار آن نمایش داده شود بنویسیم. i شماره عددی است که می خواهیم چاپ شود. این شماره از صفر شروع می شود.

```
1  // Fig. 3.11: Addition.cs
2  // An addition program.
3
4  using System;
5
6  class Addition
7  {
8     static void Main( string[] args )
9     {
10       string firstNumber,    // first string entered by user
11              secondNumber;   // second string entered by user
12
13       int number1,           // first number to add
14           number2,           // second number to add
15           sum;               // sum of number1 and number2
16
17       // prompt for and read first number from user as string
18       Console.Write( "Please enter the first integer: " );
19       firstNumber = Console.ReadLine();
20
21       // read second number from user as string
22       Console.Write( "\nPlease enter the second integer: " );
23       secondNumber = Console.ReadLine();
24
25       // convert numbers from type string to type int
26       number1 = Int32.Parse( firstNumber );
27       number2 = Int32.Parse( secondNumber );
28
29       // add numbers
30       sum = number1 + number2;
31
32       // display results
33       Console.WriteLine( "\nThe sum is {0}.", sum );
34
35    } // end method Main
36
37 } // end class Addition
```

```
Please enter the first integer: 45

Please enter the second integer: 72

The sum is 117.
```

نکته: می توان از متغیرهای رشته ای استفاده نکرد و به طور مستقیم ورودی که از طریق متد ReadLine دریافت می شود به متد Parse داده شود و به نوع صحیح تبدیل شود. مانند زیر:

```
int number1;
number1 = Int32.Parse( Console.ReadLine() );
```

۸- عملگرهای محاسباتی

بسیاری از برنامه ها محاسبات ریاضی را انجام می دهند. در جدول زیر عملگرهای محاسباتی در زبان C# ، معرفی شده است. این عملیات دودویی هستند زیرا به دو عملوند نیاز دارند. عملیاتی که به یک عملوند نیاز داشته باشند یکانی نامیده می شوند.

نام عملگر	نمایش عملگر در C#	مثال
جمع	+	x + y
تفریق	-	x – y
ضرب	*	x * y
تقسیم	/	x / y
باقیمانده	%	x % y

نکته ۱ : اگر هر دو عملوند عمل تقسیم (/) عدد صحیح باشند حاصل تقسیم نیز عدد صحیح خواهد بود و به آن تقسیم صحیح گفته می شود. اما اگر حداقل یک عملوند آن اعشاری باشد حاصل تقسیم اعشاری خواهد بود.

نکته ۲ : عمل باقیمانده (%) در C# معمولا با عملوندهای صحیح استفاده می شوند اما می تواند با عملوندهای اعشاری نیز استفاده شود. (در زبان C یا C++ فقط با عملوندهای صحیح استفاده می شود)

تمرین در کلاس: برنامه کاربردی کنسول ایجاد کنید که دو عدد از ورودی دریافت کند و جمع، تفریق، ضرب و تقسیم دو عدد را در خروجی چاپ کند.

۹- اختصارنویسی عملگرهای محاسباتی

در C#، عملگرهای محاسباتی به همراه عمل انتساب را می توان اختصارنویسی کرد. به عنوان مثال می توانیم عمل a=a+5 را با اختصار به صورت a+=5 بنویسیم. به عبارت دیگر اگر بخواهیم عملیاتی روی یک متغیر اعمال کنیم سپس مقدار بدست آمده مجددا در آن متغیر قرار بگیرد از اختصارنویسی استفاده می کنیم. این اختصارنویسی معمولا شامل عملگرهای + ، - ، * ، / و % می باشد. در جدول زیر مثالی از این نوع اختصارنویسی ارائه شده است.

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g %= 9`	`g = g % 9`	3 to g

۱۰-عملگرهای کاهش و افزایش

C# ، عملگر افزایش (++) و کاهش (--) یکانی را در اختیار می گذارد. این عملگر برای افزایش یک واحد (یا کاهش یک واحد) متغیر به کار می رود. به عنوان مثال برای یک متغیر به نام c ، c++ معادل c=c+1 یا c+=1 می باشد. این عملگرها اگر قبل از متغیر به کار روند پیش افزایش (یا پیش کاهش) نامیده می شوند و به این معناست که ابتدا مقدار متغیر یک واحد افزایش (یا کاهش) یابد سپس مقدار جدید آن در عبارت جاری استفاده شود. و اگر بعد از متغیر به کار روند پس‌افزایش (یا پس‌کاهش) نامیده می شوند که به این معناست که ابتدا مقدار فعلی متغیر در عبارت جاری استفاده شود سپس مقدار متغیر یک واحد افزایش (یا کاهش) یابد.

مثال زیر تفاوت پیش افزایش و پس افزایش را نشان می دهد.

```
1  // Fig. 4.14: Increment.cs
2  // Preincrementing and postincrementing
3
4  using System;
5
6  class Increment
7  {
8     static void Main( string[] args )
9     {
10       int c;
11
12       c = 5;
13       Console.WriteLine( c );   // print 5
14       Console.WriteLine( c++ ); // print 5 then postincrement
15       Console.WriteLine( c );   // print 6
16
17       Console.WriteLine();      // skip a line
18
19       c = 5;
20       Console.WriteLine( c );   // print 5
21       Console.WriteLine( ++c ); // preincrement then print 6
22       Console.WriteLine( c );   // print 6
23
24    } // end of method Main
25
26 } // end of class Increment
```

```
5
5
6

5
6
6
```

۱۱-عملگرهای رابطه ای و تساوی

عملگرهای رابطه ای و تساوی برای ساخت تصمیم در دستورات تصمیم استفاده می شوند. انواع عملگرهای رابطه ای و تساوی در جدول زیر معرفی شده است.

نام عمل	نماد عملگر در C#	مثال	توصیف
تساوی	==	x == y	x با y مساوی است؟
نامساوی	!=	x != y	x با y مساوی نیست؟
بزرگتر	>	x > y	x بزرگتر از y است؟
کوچکتر	<	x < y	x کوچکتر از y است؟
بزرگتر مساوی	>=	x >= y	x بزرگتر مساوی y است؟
کوچکتر مساوی	<=	x <= y	x کوچکتر مساوی y است؟

اولویت عملگرهای محاسباتی و رابطه ای در جدول زیر ارائه شده است. همانطور که دیده می شود، اولویت پرانتز از همه عملگرهای بالاتر است بنابراین می توان از پرانتز در تغییر اولویت عملگرها استفاده کرد.

نوع عملگر	عملگر	شرکت پذیری
پرانتز	( )	از چپ به راست
ضرب	* / %	از چپ به راست
جمع	+ -	از چپ به راست
رابطه ای	< <= > >=	از چپ به راست
تساوی	== !=	از چپ به راست
انتساب	=	از راست به چپ

تمرین در کلاس: بعد از اجرای عملیات زیر محتوای متغیرها را مشخص کنید. مقدار اولیه تمام متغیرها ٤ می باشد.

```
P *= x++;

q += ++x;
```

۱۲-ساختارهای کنترلی

C#، هشت ساختار کنترلی در اختیار برنامه نویس قرار می دهد. سه نوع ساختار انتخاب و چهار نوع ساختار تکرار می باشند.

ساختارهای انتخاب

ساختارهای انتخاب شامل if ، if/else و switch می باشند.

if

ساختار انتخاب if، تک انتخابی نامیده می شود زیرا **یک دستور** یا گروهی از دستورات را یا انتخاب می کند یا رد می کند. جلوی دستور if یک شرط قرار می گیرد. این شرط معمولا بوسیله عملگرهای رابطه ای و تساوی ایجاد می شود. اگر شرط برقرار باشد (درست باشد)، دستورات داخل بدنه if اجرا می شوند. در غیر این صورت اجرا به دستورات داخل بدنه if نمی رسد.

```
if (شرط)
{
دستورات
}
```

نکته: اگر دستور if فقط یک دستور در بدنه خود داشته باشد نیازی به آکولاد چپ و راست برای شروع و خاتمه بدنه خود ندارد. مانند مثال زیر که هر دستور if فقط یک دستور چاپ (Console.WriteLine) در بدنه خود دارند.

مثال زیر دو متغیر را با استفاده از ٦ دستور if تک انتخابی با هم مقایسه می کند، برای هر دستور if که شرط آن برقرار باشد دستورات بدنه آن اجرا خواهد شد.

```
1  // Fig. 3.19: Comparison.cs
2  // Using if statements, relational operators and equality
3  // operators.
4
5  using System;
6
7  class Comparison
8  {
9     static void Main( string[] args )
10    {
11       int number1,           // first number to compare
12           number2;           // second number to compare
13
14       // read in first number from user
15       Console.Write( "Please enter first integer: " );
16       number1 = Int32.Parse( Console.ReadLine() );
17
18       // read in second number from user
19       Console.Write( "\nPlease enter second integer: " );
20       number2 = Int32.Parse( Console.ReadLine() );
21
22       if ( number1 == number2 )
23          Console.WriteLine( number1 + " == " + number2 );
24
25       if ( number1 != number2 )
26          Console.WriteLine( number1 + " != " + number2 );
27
28       if ( number1 < number2 )
29          Console.WriteLine( number1 + " < " + number2 );
30
31       if ( number1 > number2 )
32          Console.WriteLine( number1 + " > " +  number2 );
33
34       if ( number1 <= number2 )
35          Console.WriteLine( number1 + " <= " + number2 );
36
37       if ( number1 >= number2 )
38          Console.WriteLine( number1 + " >= " + number2 );
39
40    } // end method Main
41
42 } // end class Comparison
```

```
Please enter first integer: 2000

Please enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

```
Please enter first integer: 1000

Please enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Please enter first integer: 1000

Please enter second integer: 1000
1000 == 1000
1000 <= 1000
1000 >= 1000
```

if/else

ساختار انتخاب if/else ، دو انتخابی نامیده می شود. زیرا بین دو گروه دستورات، یک گروه را انتخاب می کند و گروه دیگر را رد می کند. اگر شرط برقرار باشد (درست باشد)، فقط دستورات داخل بدنه if اجرا می شوند. اگر شرط برقرار نباشد (نادرست باشد)، دستورات داخل بدنه else اجرا می شوند و دستورات داخل بدنه if اجرا نخواهند شد.

```
if ( شرط )
{
        دستورات ۱
}
else
{
        دستورات ۲
}
```

قطعه کد زیر با یک مثال ساختار انتخاب if/else دو انتخابی را نشان می دهد.

```
if ( studentGrade >= 60 )
   Console.WriteLine( "Passed" );
else
   Console.WriteLine( "Failed" );
```

if/else چندگانه

در این ساختار انتخابی، یک مجموعه دستورات انتخاب می شوند و مجموعه دستورات دیگر رد خواهند شد.

```
if ( شرط ۱ )
{
        دستورات ۱
}
else if ( شرط ۲ )
{
        دستورات ۲
}
else if ( شرط ۳ )
{
        دستورات ۳
}
```

قطعه کد زیر با یک مثال ساختار انتخاب if/else چندگانه را نشان می دهد.

```
if ( studentGrade >= 90 )
   Console.WriteLine( "A" );
else if ( studentGrade >= 80 )
   Console.WriteLine( "B" );
else if ( studentGrade >= 70 )
   Console.WriteLine( "C" );
else if ( studentGrade >= 60 )
   Console.WriteLine( "D" );
else
   Console.WriteLine( "F" );
```

switch

ساختار انتخاب switch با ساختار انتخاب if/else چندگانه معادل است. ساختار انتخاب switch ، چند انتخابی نامیده می شود. زیرا از بین چندین گروه دستورات، یک گروه دستورات را انتخاب می کند و بقیه را رد می کند.

اگر انتخاب گروهی از دستورات وابسته به مقدار یک متغیر یا عبارت باشد می توانیم از ساختار چندانتخابی switch در برنامه استفاده کنیم.

ساختار کلی دستور switch به صورت زیر می باشد. در جلوی کلمه کلیدی switch داخل پرانتز نام متغیر یا عبارتی که میخواهیم مقدار آن را بررسی کنیم قرار می دهیم. ساختار switch (مانند ساختارهای دیگر) دارای نقطه شروع {) (و نقطه پایان (}) می باشد. در این ساختار با استفاده از کلمه کلیدی case مقدار متغیر جلوی switch بررسی می شود. به تعداد مقادیر موردنظر از case استفاده می کنیم. بعد از case دستورات موردنظر قرار می گیرد. هر case با کلمه کلیدی break خاتمه می یابد. break باعث می شود که اجرا از تمام case های بعدی رد شود و فقط دستورات داخل یک case اجرا شود. اگر اجرا از تمام caseها عبور کند، یعنی مقدار متغیر در هیچ کدام از caseها نیامده باشد آنگاه دستورات داخل default اجرا می شوند.

```
switch ( متغیر یا عبارت )
{
 case …. :
        { دستورات
          break;
        }
 case …. :
        { دستورات
          break;
        }
default :
        { دستورات
```

```
            break;
        }
    }
```

ساختار switch در قطعه کد زیر هم قابل مشاهده است. در مثال زیر، محتوای متغیر grade بررسی می شود با توجه به محتوای آن، متغیر مربوطه یک واحد افزایش می یابد.

```
switch ( grade )
{
   case 'A':   // grade is uppercase A
   case 'a':   // or lowercase a
      ++aCount;
      break;

   case 'B':   // grade is uppercase B
   case 'b':   // or lowercase b
      ++bCount;
      break;

   case 'C':   // grade is uppercase C
   case 'c':   // or lowercase c
      ++cCount;
      break;

   case 'D':   // grade is uppercase D
   case 'd':   // or lowercase d
      ++dCount;
      break;

   case 'F':   // grade is uppercase F
   case 'f':   // or lowercase f
      ++fCount;
      break;

   default:    // processes all other characters
      Console.WriteLine(
         "Incorrect letter grade entered." +
         "\nEnter a new grade" );
      break;

} // end switch
```

نکته : اگر بخواهیم دستورات موردنظر به ازای چند مقدار اجرا شوند می توانیم از چند case پشت سرهم استفاده کنیم. مانند قطعه کد بالا که مقدار متغیر grade را بررسی می کند و در صورتی که حرف a یا A باشد (در case اول) با دستور پیش‌افزایش یک واحد به متغیر aCount اضافه می کند.

ساختارهای تکرار

C# چهار ساختار تکرار (for , while , do/while , foreach) در اختیار برنامه نویس قرار می دهد که در ادامه معرفی خواهند شد.

for

ساختار کلی دستور for به صورت زیر است :

for(عبارت ۱ ; عبارت ۲ ; عبارت ۳)

{

دستورات

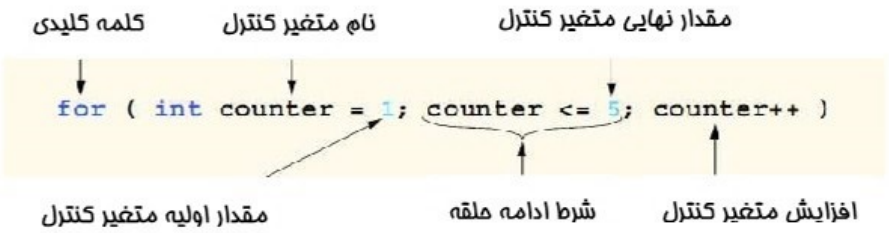
}

در عبارت ۱، مقداردهی اولیه متغیر کنترل کننده حلقه انجام می گیرد. در عبارت ۲، شرط ادامه حلقه قرار می گیرد که معمولا شامل مقدار پایانی متغیر است. در عبارت ۳، افزایش یا کاهش متغیر کنترل کننده انجام می گیرد.

مثال: برنامه ای بنویسید که با استفاده از حلقه for اعداد ۱ تا ۵ را در خطوط جداگانه چاپ نماید.

```
1  // Fig. 5.2: ForCounter.cs
2  // Counter-controlled repetition with the for structure.
3
4  using System;
5
6  class ForCounter
7  {
8     static void Main( string[] args )
9     {
10       // initialization, repetition condition and incrementing
11       // are all included in the for structure
12       for ( int counter = 1; counter <= 5; counter++ )
13          Console.WriteLine( counter );
14    }
15 }
```

```
1
2
3
4
5
```



مثالهای استفاده از حلقه for:

۱) متغیر کنترل را از ۱ تا ۱۰۰ با افزایش یک واحد تغییر دهید.

```
for ( int i = 1; i <= 100; i++ )
```

۲) متغیر کنترل را از ۱۰۰ تا ۱ با کاهش یک واحد (یا افزایش ۱-) تغییر دهید.

```
for ( int i = 100; i >= 1; i-- )
```

۳) متغیر کنترل را از ۷ تا ۷۷ با افزایش ۷ تایی تغییر دهید.

```
for ( int i = 7; i <= 77; i += 7 )
```

٤) متغیر کنترل را از ۲۰ تا ۲ با کاهش ۲ تایی تغییر دهید.

```
for ( int i = 20; i >= 2; i -= 2 )
```

٥) متغیر کنترل را به مقادیر زیر (به ترتیب از راست به چپ) تغییر دهید: ۲، ٥، ۸، ۱۱، ۱٤، ۱۷، ۲۰

```
for ( int j = 2; j <= 20; j += 3 )
```

٦) متغیر کنترل را به مقادیر زیر (به ترتیب از راست به چپ) تغییر دهید: ۹۹، ۸۸، ۷۷، ٦٦، ٥٥، ٤٤، ۳۳، ۲۲، ۱۱

```
for ( int j = 99; j >= 0; j -= 11 )
```

تمرین در کلاس: برنامه ی کاربردی کنسول ایجاد کنید که یک عدد از ورودی بگیرد و اعداد ۱ تا آن عدد را در خروجی چاپ کند.

تمرین در کلاس: برنامه ی کاربردی کنسول ایجاد کنید که یک عدد از ورودی بگیرد و اول بودن آن را تشخیص دهد.

while

در ساختار تکرار for، تعداد تکرار باید مشخص باشد. while ساختار تکرار دیگری است که به برنامه نویس این امکان را می دهد که یک سری اعمال را تا زمانی که یک شرط برقرار است تکرار کند.

بدنه ساختار while ممکن است شامل یک دستور یا چندین دستور باشد. همانند ساختار if و for اگر بدنه while بیشتر از یک دستور داشته باشد نیاز به آکولاد چپ برای شروع بدنه و آکولاد راست برای خاتمه بدنه می باشد.

با توجه به ساختار تکرار for، به طور کلی ساختار تکرار while به شکل زیر می باشد:

```
عبارت ۱
while ( عبارت ۲ )
{
دستورات
عبارت ۳
}
```

تمرین در کلاس: برنامه کاربردی کنسول ایجاد کنید که یک عدد از ورودی بگیرد، مقلوب عدد را در خروجی چاپ کند.

مثال: برنامه ای بنویسید که با استفاده از حلقه while اعداد ۱ تا ۵ را در خطوط جداگانه چاپ نماید.

```
1  // Fig. 5.1: WhileCounter.cs
2  // Counter-controlled repetition.
3
4  using System;
5
6  class WhileCounter
7  {
8     static void Main( string[] args )
9     {
10       int counter = 1;          // initialization
11
12       while ( counter <= 5 )    // repetition condition
13       {
14          Console.WriteLine( counter );
15          counter++;             // increment
16
17       } // end while
18
19    } // end method Main
20
21 } // end class WhileCounter
```

```
1
2
3
4
5
```

مثال : با استفاده از حلقه while ، ۱۰ نمره از ورودی دریافت کنید، معدل نمرات را محاسبه کرده و در خروجی چاپ کنید.

```
1  // Fig. 4.7: Average1.cs
2  // Class average with counter-controlled repetition.
3
4  using System;
5
6  class Average1
7  {
8     static void Main( string[] args )
9     {
10       int total,           // sum of grades
11           gradeCounter,    // number of grades entered
12           gradeValue,      // grade value
13           average;         // average of all grades
14
15       // initialization phase
16       total = 0;           // clear total
17       gradeCounter = 1;    // prepare to loop
18
19       // processing phase
20       while ( gradeCounter <= 10 )  // loop 10 times
21       {
22          // prompt for input and read grade from user
23          Console.Write( "Enter integer grade: " );
24
25          // read input and convert to integer
26          gradeValue = Int32.Parse( Console.ReadLine() );
27
28          // add gradeValue to total
29          total = total + gradeValue;
30
31          // add 1 to gradeCounter
32          gradeCounter = gradeCounter + 1;
33       }
34
35       // termination phase
36       average = total / 10;  // integer division
37
38       // display average of exam grades
39       Console.WriteLine( "\nClass average is {0}", average );
40
41    } // end Main
42
43 } // end class Average1
```

```
Enter integer grade: 100
Enter integer grade: 88
Enter integer grade: 93
Enter integer grade: 55
Enter integer grade: 68
Enter integer grade: 77
Enter integer grade: 83
Enter integer grade: 95
Enter integer grade: 73
Enter integer grade: 62

Class average is 79
```

مثال : با استفاده از حلقه while برنامه ای بنویسید که تعدادی نمره از ورودی دریافت کند و معدل نمرات را محاسبه کرده و در خروجی چاپ کند، برنامه زمانی به دریافت نمرات خاتمه دهد که کاربر عدد ۱- را وارد کند.

```
1  // Fig. 4.9: Average2.cs
2  // Class average with sentinel-controlled repetition.
3
4  using System;
5
6  class Average2
7  {
8     static void Main( string[] args )
9     {
10       int total,            // sum of grades
11           gradeCounter,     // number of grades entered
12           gradeValue;       // grade value
13
14       double average;       // average of all grades
15
16       // initialization phase
17       total = 0;            // clear total
18       gradeCounter = 0;     // prepare to loop
19
20       // processing phase
21       // prompt for input and convert to integer
22       Console.Write( "Enter Integer Grade, -1 to Quit: " );
23       gradeValue = Int32.Parse( Console.ReadLine() );
24
25       // loop until a -1 is entered by user
26       while ( gradeValue != -1 )
27       {
28          // add gradeValue to total
29          total = total + gradeValue;
30
31          // add 1 to gradeCounter
32          gradeCounter = gradeCounter + 1;
33
34          // prompt for input and read grade from user
35          // convert grade from string to integer
36          Console.Write( "Enter Integer Grade, -1 to Quit: " );
37          gradeValue = Int32.Parse( Console.ReadLine() );
38
39       } // end while
40
41       // termination phase
42       if ( gradeCounter != 0 )
43       {
44          average = ( double ) total / gradeCounter;
45
46          // display average of exam grades
47          Console.WriteLine( "\nClass average is {0}", average );
48
49       }
50       else
51       {
52          Console.WriteLine( "No grades were entered." );
53       }
54
55    } // end method Main
56
57 } // end class Average2
```

```
Enter Integer Grade, -1 to Quit: 97
Enter Integer Grade, -1 to Quit: 88
Enter Integer Grade, -1 to Quit: 72
Enter Integer Grade, -1 to Quit: -1

Class average is 85.6666666666667
```

do/while

ساختار کلی دستور do/while به صورت زیر است

do

{

دستورات

} while (شرط);

فرق اساسی بین ساختار تکرار do/while با while و for در این است که در do/while دستورات حداقل یکبار اجرا می شوند. یعنی حتی اگر شرط برقرار نباشد دستورات یکبار اجرا می شوند چون بررسی درستی شرط بعد از اجرای دستورات است. اما در while یا for اگر شرط برقرار نباشد دستورات اصلا اجرا نمی شوند.

نکته : در دستور do/while بعد از while و شرط، سمی کالن (;) الزامی است.

مثال : برنامه ای بنویسید که با استفاده از do/while اعداد ۱ تا ۵ را در خطوط جداگانه چاپ نماید.

```
1  // Fig. 5.12: DoWhileLoop.cs
2  // The do/while repetition structure.
3
4  using System;
5
6  class DoWhileLoop
7  {
8     static void Main( string[] args )
9     {
10       int counter = 1;
11
12       do
13       {
14          Console.WriteLine( counter );
15          counter++;
16       } while ( counter <= 5 );
17
18    } // end method Main
19
20 } // end class DoWhileLoop
```

```
1
2
3
4
5
```

۱۳-دستور break

وقتی دستور break در ساختارهای for، while، do/while و switch اجرا می شود باعث می شود کنترل اجرا به خارج از ساختار منتقل شود، بنابراین دستورات بعد از break اجرا نمی شوند.

در قطعه کد زیر، زمانی که count برابر ۵ باشد، شرط دستور if درست است و دستور break اجرا می شود بنابراین کنترل از حلقه for خارج می شود.

```
string output = "";
int count;

for ( count = 1; count <= 10; count++ )
{
   if ( count == 5 )
      break;             // skip remaining code in loop
                         // if count == 5

   output += count + " ";

} // end for loop
```

بعد از اجرای حلقه for در قطعه کد بالا، رشته output برابر است با : 1 2 3 4

۱٤-دستور continue

وقتی دستور continue در ساختارهای for، while و do/while اجرا می شوند، از اجرای دستورات بعد از آن صرف نظر شده و کنترل اجرا به تکرار بعدی حلقه منتقل می شود.

```
 string output = "";

for ( int count = 1; count <= 10; count++ )
{
   if ( count == 5 )
      continue;          // skip remaining code in loop
                         // only if count == 5

   output += count + " ";
}
```

پس از اجرای حلقه for در قطعه کد بالا، رشته output برابر است با : 1 2 3 4 6 7 8 9 10

۱۵- عملگرهای شرطی و منطقی

تا اینجا، شرطهای ساده مورد استفاده قرار گرفتند. مانند count<=10 یا number != 20 . این شرطها با استفاده از عملگرهای رابطه ای (>= , > , <= , <) و عملگرهای مساوی (!= , ==) بیان می شوند.

برای داشتن شرطهای چندتایی می توان از ساختارهای if/else چندگانه استفاده کرد، یا از عملگرهای منطقی و رابطه ای استفاده کرد.

در جدول زیر، عملگرهای شرطی و منطقی که زبان C# در اختیار می گذارد ارائه شده اند. از این عملگرها برای ساختن شرطهای پیچیده از شرطهای ساده استفاده خواهیم کرد.

توصیف عملگر	عملگر
AND شرطی	&&
AND منطقی	&
OR شرطی	||
OR منطقی	|
XOR منطقی	^
NOT منطقی	!

AND شرطی: اگر بخواهیم دو شرط (یا بیشتر) با هم در برنامه درست باشند از عملگر AND شرطی استفاده می کنیم. مانند مثال زیر، که اگر gender==1 و age >= 65 با هم برقرار باشد یک واحد به متغیر seniorFemales اضافه می شود در غیر این صورت عملی انجام نمی گیرد.

```
if ( gender == 1 && age >= 65 )
   ++seniorFemales;
```

جدول زیر، جدول درستی عملگر AND شرطی را نشان می دهد که در آن، فقط در حالتی که هر دو شرط درست (true) باشند عبارت شرطی درست است، در غیر این صورت عبارت شرطی نادرست است.

expression1	expression2	expression1 && expression2
false	false	false
false	true	false
true	false	false
true	true	true

OR شرطی: اگر بخواهیم حداقل یکی از شرطهای مشخص شده درست باشند از عملگر OR شرطی در بین آن شرطها استفاده می کنیم. مانند مثال زیر، که اگر حداقل یکی از متغیرهای تعیین شده بزرگتر مساوی ۹۰ باشند جمله "Student grade is A" چاپ خواهد شد در غیر این صورت عملی انجام نمی شود.

```
if ( semesterAverage >= 90 || finalExam >= 90 )
   Console.WriteLine( "Student grade is A" );
```

جدول زیر، جدول درستی عملگر OR شرطی را نشان می دهد. که در آن فقط زمانی که هر دو شرط نادرست باشند عبارت شرطی نادرست است، در غیر این صورت عبارت شرطی درست است.

expression1	expression2	expression1 || expression2
false	false	false
false	true	true
true	false	true
true	true	true

AND منطقی و OR منطقی : عملگرهای AND منطقی (&) و OR منطقی (||) مشابه عملگرهای AND شرطی (&&) و OR شرطی (||) می باشند با این تفاوت که در عملگرهای منطقی ارزیابی اتصال کوتاه وجود ندارد، به عبارت دیگر تمام شرطها چک می شوند. ارزیابی اتصال کوتاه به این معناست که در AND شرطی اگر سمت چپ ترین شرط نادرست باشد شرطهای بعدی چک نخواهند شد و در OR شرطی اگر سمت چپ ترین شرط درست باشد شرطهای بعدی چک نخواهند شد.

در مثال زیر، اگر gender == 1 نباشد شرط age >= 65 بررسی نخواهد شد به این ویژگی، **ارزیابی اتصال کوتاه** گفته می شود.

```
if ( gender == 1 && age >= 65 )
   ++seniorFemales;
```

اما در مثال زیر ویژگی ارزیابی اتصال کوتاه وجود ندارد و هر دو شرط بررسی خواهند شد.

```
if ( gender == 1 &  age >= 65 )
   ++seniorFemales;
```

بنابراین زمانی که می خواهیم در شرط سمت راست حتما عملی انجام شود (بدون توجه به اینکه عبارت شرطی درست است یا نه)، از عملگرهای منطقی استفاده می کنیم. مانند مثال زیر که در آن بعد از بررسی age >= 65 مقدار آن یک واحد افزایش می یابد. در این مثال افزایش مقدار age حتما باید انجام شود.

```
birthday == true | ++age >= 65
```

XOR منطقی: عبارت شرطی XOR منطقی زمانی درست است که یکی از شرطها درست و شرط دیگر نادرست باشد. جدول زیر، جدول درستی عملگر XOR شرطی را نشان می دهد.

expression1	expression2	expression1 ^ expression2
false	false	false
false	true	true
true	false	true
true	true	false

عملگر NOT منطقی (!) : C# عملگر NOT منطقی را برای معکوس کردن شرط در اختیار برنامه نویس قرار می دهد. برخلاف عملگرهای &&، ||، ^، & و | که عملگرهایی دوتایی هستند (نیاز به دو عملوند دارند)، عملگر ! ، عملگری یکانی است و فقط به یک شرط نیاز دارد. عملگر ! قبل از شرط قرار می گیرد.

مثال زیر کاربرد عملگر ! را نشان می دهد. دستور بعد از if زمانی اجرا می شود که grade == sentinelValue نادرست باشد.

```
if ( ! ( grade == sentinelValue ) )
   Console.WriteLine( "The next grade is " + grade );
```

به جای استفاده از عملگر ! منطقی می توان از عملگرهای رابطه ای یا تساوی استفاده کرد. مثال فوق را می توان با استفاده از عملگر نامساوی به صورت زیر بیان کرد:

```
if ( grade != sentinelValue )
   Console.WriteLine( "The next grade is " + grade );
```

جدول زیر، جدول درستی عملگر ! را نشان می دهد.

expression	!expression
false	true
true	false

تاکنون عملگرهای محاسباتی، اختصارنویسی محاسباتی، کاهشی، افزایشی، رابطه ای، تساوی، شرطی و منطقی را بیان کردیم. جدول زیر به طور کامل اولویت بندی عملگرهای ذکر شده را نشان می دهد.

نوع عملگر	عملگر	شرکت پذیری
پرانتز	( )	از چپ به راست
پسوند یکانی	++ --	از راست به چپ
پیشوند یکانی	++ -- + - ! (type)	از راست به چپ
ضرب	* / %	از چپ به راست
جمع	+ -	از چپ به راست
رابطه ای	< <= > >=	از چپ به راست
تساوی	== !=	از چپ به راست
AND منطقی	&	از چپ به راست
XOR منطقی	^	از چپ به راست
OR منطقی	|	از چپ به راست
AND شرطی	&&	از چپ به راست
OR شرطی	||	از چپ به راست
شرطی	? :	از راست به چپ
انتساب	= += -= *= /=	از راست به چپ

۱۶- متدها

متدها ساختارهایی هستند که به برنامه نویس این امکان را می دهند که برنامه ها را ماژولار کنند. ساختار کلی متدها به شکل زیر می باشد:

(لیست پارامترها) نام متد نوع داده خروجی

{

دستورات

}

نوع داده خروجی با توجه به خروجی که قرار است متد ایجاد کند از بین انواع داده ای در C# انتخاب می شود. مانند int، char، void (تهی)، string، double و غیره.

نام متد همانند نام متغیر یک شناسه است که هر شناسه ترکیبی از حروف، ارقام، علامت زیرخط (_) و امپرسند (@) می‌باشد که با ارقام شروع نشود و شامل فاصله نباشد.

لیست پارامترها، متغیرهای محلی متد می باشند (متغیر محلی فقط در بلوک تعریف شده شناخته شده است بنابراین متغیرهای محلی متد فقط در متد شناخته شده اند) که به کمک آنها مقادیری به متد ارسال می شود. این متغیرها برای اتصال بین متدها و برنامه استفاده میشوند.

مثال: قطعه کد زیر، متدی برای محاسبه مجذور یک عدد می باشد. خروجی این متد، عدد صحیح (int) است. ورودی آن پارامتری به نام y از نوع عدد صحیح (int) است.

نکته: خروجی متد با استفاده از کلمه کلیدی return به بیرون متد باز گردانده می شود.

```
// Square method definition
int Square( int y )
{
   return y * y; // return square of y

} // end method Square
```

تمرین در کلاس: برنامه‌ی کاربردی کنسول ایجاد کنید که یک عدد از ورودی دریافت کند و با استفاده از متدها، مجذور و مکعب اعداد ۱ تا آن عدد را در خروجی چاپ نماید.

تمرین در کلاس: برنامه‌ی کاربردی کنسول ایجاد کنید که با استفاده از متدها، اول بودن عدد را تشخیص دهد.

۱۷- متدهای کلاس Math

متدای کلاس Math به برنامه نویس امکان انجام محاسبات ریاضی را می دهند. در جدول زیر متدهای این کلاس معرفی شده اند. برای فراخوانی متدهای کلاس Math، ابتدا نام کلاس Math، سپس عملگر dot (.) بعد از آن نام متد را به همراه پرانتز باز و بسته که پارامتر داخل آن قرار می گیرد. به طور مثال فراخوانی متد Sqrt کلاس Math برای محاسبه جذر عدد ۱۶ به این صورت است: **Math.Sqrt(16) = 4**

متد	توصیف	مثال
Abs(x)	قدر مطلق x را محاسبه می کند	Abs(23.7) = 23.7 Abs(-23)=23 Abs(0) = 0
Ceiling (x)	X را به کوچکترین عدد بزرگتر از x گرد می کند	Ceiling(9.2) = 10.0 Ceiling(-9.8) = -9.0
Cos(x)	کسینوس x را محاسبه می کند	Cos(0.0) = 1.0
Sin(x)	سینوس x را محاسبه می کند	Sin(0.0) = 0.0
Tan(x)	تانژانت x را محاسبه می کند	Tan(0.0) = 0.0
Exp(x)	e^x را محاسبه می کند	Exp(1.0) = 2.7
Floor(x)	x را به بزرگترین عدد کوچکتر از x گرد می کند.	Floor (9.2) = 9.0 Floor(-9.8) = -10.0
Log(x)	لگاریتم x را محاسبه می کند	
Max(x,y)	بین x و y عدد بزرگتر را پیدا می کند	Max(2.3,12.7) = 12.7 Max(-2,-4) = -2
Min(x,y)	بین x و y عدد کوچکتر را می یابد	Min(2.3,12.7) = 2.3 Min(-2.3,-12.7) = -12.7
Pow(x,y)	x به توان y را محاسبه می کند (x^y)	Pow(2,3) = 8 Pow(9.0,0.5) = 3.0
Sqrt(x)	ریشه دوم عدد x را محاسبه می کند	Sqrt(900.0) = 30.0 Sqrt(9.0) = 3.0

۱۸- ارسال پارامتر به متد با مقدار و آدرس

در ارسال پارامتر با مقدار، یک کپی از متغیر به متد ارسال می شود. بنابراین اگر مقدار پارامتر در متد تغییر کند متغیر اصلی در خارج متد متوجه تغییر نخواهد شد. برای اینکه تغییر پارامتر در داخل متد به متغیر اصلی در خارج متد اعمال شود از ارسال با آدرس (ارجاع) استفاده می کنیم. برای ارسال پارامتر با آدرس از کلمه کلیدی ref در لیست پارامترهای متد استفاده می کنیم. در مثال زیر ارسال پارامتر به متد square با آدرس انجام گرفته است

```
// x passed by reference and method modifies
// original variable's value
void SquareRef( ref int x )
{
   x = x * x;
}
```

۱۹- آرایه ها

به خانه های پشت سر هم حافظه که هم نوع و هم نام هستند آرایه گفته می شود. به هر خانه، یک عنصر از آرایه گفته می شود. برای دسترسی به هر عنصر آرایه از نام آرایه به همراه شماره عنصر استفاده می کنیم.

اعلان آرایه: اعلان یا تعریف آرایه به صورت زیر انجام می شود:

نوع عناصر آرایه [] نام آرایه ;

به عنوان مثال int[] c ; تعریف آرایه ای از نوع صحیح را نشان می دهد. اما برای استفاده از آرایه باید به اندازه موردنظر اشاره گر به خانه های حافظه ایجاد کرد و فضای لازم را از حافظه اشغال کرد. که این عمل با استفاده از عملگر new به صورت زیر انجام می شود:

```
c = new  int [12];
```

بنابراین با استفاده از دو دستور زیر آرایه ای با ۱۲ عنصر از اعداد صحیح ایجاد کردیم، فضای اشغال شده در حافظه برای این آرایه در شکل زیر نشان داده شده است.

```
int[] c;
c = new int[ 12 ];
```

که می توانیم آنها را در یک خط همزمان به کار برد به این صورت :

```
int[] c = new int[ 12 ];
```

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

نکته: شماره عنصر (اندیس) در آرایه از صفر شروع می شود. بنابراین برای آرایه ۱۲ عنصری شماره عناصر از ۰ تا ۱۱ می باشد.

در قسمت نوع داده آرایه، می توان هر نوع داده ای را استفاده کرد مانند string، double، char و غیره. بعنوان مثال، در زیر آرایه هایی از نوع رشته و اعشاری تعریف می کنیم. دقت کنید که می توان با استفاده از کاما (,) بیشتر از یک آرایه را همزمان تعریف کرد:

```
string[] b = new string[ 100 ], x = new string[ 27 ];

double[] array1 = new double[ 10 ],
         array2 = new double[ 20 ];
```

علاوه بر استفاده از عملگر new برای ایجاد آرایه ها، می توان با مقداردهی آرایه ها را ایجاد کرد و فضای لازم را از حافظه اشغال نمود. مثال زیر آرایه پنج عنصری از اعداد صحیح به کمک مقداردهی تعریف و ایجاد می کند.

```
int []  y = { 12 , 23 , 65  , 84 , 84 };
```

قطعه کد زیر، مجموع عناصر آرایه ۱۰ عنصری از اعداد صحیح به نام a را محاسبه می کند. متغیر total جمع عناصر آرایه را در خود نگه می دارد.

```
int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int total = 0;

for ( int i = 0; i < a.Length; i++ )
   total += a[ i ];
```

۲۰- ارسال آرایه به متد

اگر بخواهیم آرایه ای را به متد ارسال کنیم، باید آرایه یکی از پارامترهای متد باشد، بنابراین در لیست پارامترهای متد تعریف آرایه موردنظر انجام می گیرد. سپس در هنگام فراخوانی فقط اسم آرایه را به عنوان یک پارامتر در داخل پرانتز جلوی نام متد می نویسیم. قطعه کد زیر، متدی است که عمل جستجوی خطی را در یک آرایه انجام می دهد.

```
// search array for the specified key value
public int LinearSearch( int[] array, int key )
{
   for ( int n = 0; n < array.Length; n++ )
   {
      if ( array[ n ] == key )
         return n;
   }

   return -1;

} // end method LinearSearch
```

برای فراخوانی متد و پاس دادن آرایه به متد، به صورت زیر عمل می کنیم:

```
int elementIndex = LinearSearch( a, searchKey );
```

عنصر searchKey ، عنصر موردجستجو است و آرایه a، آرایه ای است که می خواهیم عنصر را در آن جستجو کنیم که به صورت زیر تعریف شده اند.

```
int searchKey = 2;
int[] a = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26,
            28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50 };
```

مرتب سازی آرایه : برای مرتب سازی (صعودی) آرایه ها می توان از متد sort در کلاس Array به صورت زیر استفاده کرد.

Array.Sort(a);

برای مرتب سازی نزولی آرایه می توان بعد از استفاده از متد Sort، با استفاده از متد Reverse، آرایه مرتب شده صعودی را برعکس کرد بنابراین با دو دستور زیر مرتب سازی نزولی آرایه a انجام می گیرد:

```
Array.Sort(a);
Array.Reverse(a);
```

۲۱- آرایه دو بعدی

برای اعلان آرایه دو بعدی به صورت زیر عمل می کنیم:

نوع عناصر آرایه [,] نام آرایه = new int [تعداد سطر , تعداد ستون];

دستورات زیر آرایه ای دوبعدی از نوع اعداد صحیح با ۲ سطر و ۲ ستون با نام b تعریف می کند.

```
int[,] b = new int[ 2, 2 ];

b[ 0, 0 ] = 1;
b[ 0, 1 ] = 2;
b[ 1, 0 ] = 3;
b[ 1, 1 ] = 4;
```

به جای دستورات بالا می توان اعلان و مقداردهی را همزمان به صورت زیر انجام داد:

```
int[,] b = { { 1, 2 }, { 3, 4 } };
```

آرایه دوبعدی با این نوع نمایش که معرفی شد، طول تمام سطرها در آن یکسان است. اما در C# میتوان آرایه های دوبعدی با طول سطر متفاوت هم داشت که اعلان آن به شکل زیر انجام می شود:

نوع عناصر آرایه [][] نام آرایه = new int [تعداد سطر][تعداد ستون];

مشخص کردن تعداد ستون در این روش الزامی نیست و می توان برای هر سطر تعداد ستون مربوطه را با مقداردهی مشخص کرد مانند زیر:

```
int[][] array2 = new int[ 3 ][];
array2[ 0 ] = new int[] { 1, 2 };
array2[ 1 ] = new int[] { 3 };
array2[ 2 ] = new int[] { 4, 5, 6 };
```

```
grades = new int[ 3 ][];
grades[ 0 ] =  new int[]{ 77, 68, 86, 73 };
grades[ 1 ] =  new int[]{ 96, 87, 89, 81 };
grades[ 2 ] =  new int[]{ 70, 90, 86, 81 };
```

در این مثال یک آرایه دوبعدی به نام array2 تعریف شده است که ۳ سطر دارد و سطر اول آن ۲ ستون، سطر دوم آن ۱ ستون و سطر سوم آن ۳ ستون دارد.

تمرین در کلاس: برنامه ای بنویسید که آرایه ۱۰ عنصری از اعداد صحیح را از ورودی مقداردهی کند سپس آرایه را به صورت نزولی مرتب کند و در خروجی چاپ کند.

تمرین در کلاس: برنامه ای بنویسید که آرایه ای ۵ عنصری از اعداد صحیح را از ورودی دریافت کند سپس با کمک متد، اول بودن اعداد را بررسی کند و تعداد اعداد اول را در خروجی چاپ کند.

۲۲- ساختار تکرار foreach

foreach، ساختار تکراری است که به جای اندیس از یک متغیر استفاده می کند که مقادیر هر عنصر آرایه در آن قرار می گیرد. ساختار کلی foreach به صورت زیر است:

```
foreach( نام آرایه  in  نام متغیر  نوع عناصر آرایه )
{
دستورات
}
```

مثال زیر با استفاده از foreach مینیمم مقدار یک آرایه را می یابد.

```
1  // Fig. 7.16: ForEach.cs
2  // Demonstrating for/each structure.
3  using System;
4
5  class ForEach
6  {
7     // main entry point for the application
8     static void Main( string[] args )
9     {
10       int[,] gradeArray = { { 77, 68, 86, 73 },
11          { 98, 87, 89, 81 }, { 70, 90, 86, 81 } };
12
13       int lowGrade = 100;
14
15       foreach ( int grade in gradeArray )
16       {
17          if ( grade < lowGrade )
18             lowGrade = grade;
19       }
20
21       Console.WriteLine( "The minimum grade is: " + lowGrade );
22    }
23 }
```

```
The minimum grade is: 68
```

تمرین

۱- در جاهای خالی عبارات مناسب قرار دهید.

۱. ---------- و ---------- ابتدا و انتهای بدنه if ، متد و کلاس را مشخص می کنند.

۲. دستور --------- سازنده تصمیم می باشد.

۳. --------- یک توضیح تک خطی را شروع می کند.

٤. اجرای برنامه های کاربردی #C از ---------- شروع می شود.

٥. متدهای ---------- و ---------- برای نمایش اطلاعات در پنجره کنسول استفاده می شوند.

٦. دستور if به برنامه این امکان را می دهد که براساس ---------- یا ---------- شرط تصمیم گیری کند.

۷. متغیر، محلی از ---------- است که می تواند یک مقدار در خود ذخیره کند.

۸. در عملگر انتساب، ابتدا عبارت ---------- ارزیابی می شود.

۲- درستی یا نادرستی عبارات زیر را با ذکر دلیل تعیین کنید.

۱. توضیحات در برنامه باعث می شوند متن بعد از // ، هنگام اجرای برنامه در پنجره کنسول نمایش داده شوند.

۲. در C# ، همه متغیرها قبل از استفاده باید تعریف شوند.

۳. در C# متغیرهای number و NuMbEr یکسان در نظر گرفته می شوند.

٤. متد Int32.Parse یک عددصحیح را به رشته تبدیل می کند.

۵. خطوط خالی، کاراکترهای فاصله، کاراکترهای خط جدید و کاراکترهای تب هنگامی که در بیرون رشته ها قرار بگیرند توسط کامپایلر نادیده گرفته می شوند.

٦. هر برنامه کاربردی C# باید شامل یک متد Main باشد.

۳- یک برنامه کاربردی کنسول ایجاد کنید که اهداف زیر را برآورده سازد:

۱. عبارت "Enter two numbers:" را نمایش دهد، سپس دو عدد صحیح از ورودی دریافت نماید و در متغیرهای a و b ذخیره نماید.

۲. حاصلضرب دو متغیر a و b را در متغیر c قرار دهد.

۳. مقدار حاصلضرب را به کمک عبارت "The product of ---- in ---- is ------." نمایش دهد. که به ترتیب در جاهای خالی از سمت چپ به راست مقدار متغیرهای a، b و c قرار می گیرد.

٤- یک برنامه کاربردی کنسول بنویسید که اعداد ۱ تا ٤ را روی یک خط نمایش دهد بطوریکه بین هر زوج اعداد مجاور هم یک فاصله باشد. برنامه را به دو صورت زیر بنویسید:

۱- یکبار از دستور Console.Write استفاده کنید.

۲- چهار بار از دستور Console.Write استفاده کنید.

۵- یک برنامه کاربردی کنسول بنویسید که شکل زیر را با استفاده از کاراکتر ستاره '*' رسم نماید.

```
*********
*       *
*       *
*       *
*       *
*       *
*       *
*       *
*********
```

۶- خروجی دستور زیر چیست ؟

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

۷- خروجی دستورات زیر چیست؟

```
Console.Write("*");
Console.Write("***");
Console.WriteLine("*****");
Console.Write("****");
Console.WriteLine("**");
```

۸- یک برنامه کاربردی کنسول بنویسید که دو عدد صحیح از ورودی دریافت کند و تعیین کند که آیا عدد دوم مقسوم علیه ای از عدد اول است یا نه؟ مثلا برای دو عدد ۱۵ و ۳ این شرایط برقرار است. (راهنمایی از عملگر باقیمانده (%) استفاده کنید.)

۹- خروجی قطعه کدهای زیر را تعیین کنید.

a)
```
sum = 0;
for ( count = 1; count <= 99; count += 2 )
   sum += count;
```

c)
```
x = 1;

while ( x <= 20 )
{
   Console.Write( x );

   if ( x % 5 == 0 )
      Console.WriteLine();
   else
      Console.Write( '\t' );

   ++x;
}
```

```
d) for ( x = 1; x <= 20; x++ )
   {
      Console.Write( x );

      if ( x % 5 == 0 )
         Console.WriteLine();
      else
         Console.Write( '\t' );
   }
```

۱۰-با استفاده از برنامه کاربردی کنسول، هر کدام از الگوهای زیر را (به طور جداگانه) با استفاده از حلقه ها ایجاد کنید.

```
*           **********  **********          *       *
**          *********    *********         **      ***
***         ********      ********        ***     *****
****        *******        *******      ****    *******
*****       ******          ******    *****   *********
******      *****            *****  ******    *******
*******     ****              **** *******     *****
********    ***                ***********      ***
*********   **                  *********       *
**********  *                    *******
```

۱۱-برنامه کاربردی کنسول ایجاد کنید که برای هر عدد ورودی، اعداد اول ۱ تا آن عدد را در خروجی چاپ کند.

۱۲-برنامه کاربردی کنسولی ایجاد کنید که خروجی زیر را تولید کند.

A

A B

A B C

.....

.....

A B C ..... Z

۱۳-برنامه‌ی کاربردی کنسول ایجاد کنید که برای دو عدد ورودی، اعداد کامل بین دو عدد را در خروجی چاپ کند.

۱۴-برنامه‌ی کاربردی کنسول ایجاد کنید که برای تعدادی عدد از ورودی، میانگین و ماکزیمم اعداد را مشخص کند، برنامه زمانی به دریافت اعداد خاتمه دهد که کاربر عدد کوچکتر از صفر وارد کند.

۱۵-برنامه‌ی کاربردی کنسول ایجاد کنید که اعداد جدول ضرب ۱۰×۱۰ را در خروجی چاپ کند.

۱۶-برنامه‌ی کاربردی کنسول ایجاد کنید که کاراکتری را به عنوان ورودی دریافت کند و کاراکتر را به همراه کد اسکی آن در خروجی چاپ کند.

۱۷-برنامه ای بنویسید که متد مرتب سازی **bubble** را پیاده سازی کند، سپس یک آرایه ۱۰ عنصری از ورودی دریافت کند و آن را به کمک متد، مرتب کند و در خروجی چاپ کند.

۱۸-متد جستجوی دودویی را پیاده سازی کنید، سپس یک آرایه را مقدار دهی کنید و به کمک متد، عددی را در آن مورد جستجو قرار دهید.

۱۹- برنامه ای بنویسید که برای ۱۰ دانشجو، نمرات درسهای آنها را در یک آرایه دوبعدی ذخیره کند و معدل هر کدام را محاسبه و در خروجی چاپ نماید. (تعداد دروس دانشجویان برابر نیست)