بسمه تعالی

مبانی کامپیوتر و برنامه سازی

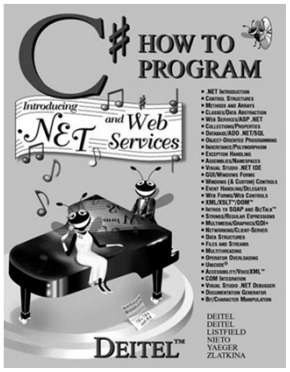
علی چوداری خسروشاهی
Akhosroshahi@Gmail.com
www.Akhosroshahi.ir

دانشگاه آزاد اسلامی

1

مراجع

- C# How to Program
  - PAUL DEITEL
  - HARVEY DEITEL



فصل سوم

مقدمه ای برنامه نویسی C#

3

3.1مقدمه

- Console applications
  - بدون اجزای بصری
  - تنها خروجی متن
  - دو نوع دارد
    - MS-DOS prompt
      - در Windows 95/98/ME استفاده می شود
    - Command prompt
      - در windows 2000/NT/XP n استفاده می شود
- Windows applications
  - از چند نوع خروجی تشکیل شده
  - شامل واسط گرافیکی کاربر (GUIها) می باشد

4

3.2برنامه ساده: چاپ یک خط متن

- توضیحات (Comments)
  - توضیحات میتوانند ایجاد شوند با استفاده از ...//
  - توضیحات چند خطی با استفاده از/* ... */
  - نظرات توسط کامپایلر نادیده گرفته می شود
  - تنها برای خوانندگان استفاده می شود
- فضای نام (Namespaces)
  - بخشهای مرتبط از C# را در یک دسته بندی قرار می دهد
  - استفاده مجدد آسان از کدها را امکانپذیر می سازد
  - بسیاری از فضاهای نام به در کتابخانه چارچوب دات نت پیدا شده است
  - باید به منظور استفاده ارجاع داده شود
- فضای سفید (White Space)
  - شامل فضاها، کاراکترهای خط جدید و زبانه ها

5

3.2برنامه ساده: چاپ یک خط متن

- کلمات کلیدی
  - کلماتی می باشد که نمی توان از آنها به عنوان نام متغییر، نام کلاس و یا کاربردهای دیگر استفاده کرد
  - کاربرد غیر قابل تغییر در زبان دارد
  - به عنوان مثال: **`class`**
  - تمام کلمات کلیدی حروف کوچک می باشند
- کلاسها (Classes)
  - نام کلاس تنها می تواند یک کلمه باشد (بدون فاصله)
  - نام کلاس می تواند با حروف بزرگ در کلمه انگلیسی مشخص شود (به عنوان مثال: MyFirstProgram)
  - نام کلاس یک شناسه (identifier) می باشد
    - می تواند شامل حروف، رقم و کاراکتر زیر خط(_) باشد
    - نمی تواند با رقم شروع شود
    - می توانید با در نماد (@) شروع شود

6

3.2برنامه ساده: چاپ یک خط متن

- بدنه کلاس با ({) شروع می شود
- بدنه کلاس به (}) ختم می شود.
- توابع (Methods)
  - بلوکهای برنامه را می سازد
  - **تابع `Main`**
    - هر برنامه خط فرمان یا ویندوز باید یکی داشته باشد.
    - اجرای هر برنامه از **`Main`** شروع می شود
  - شروع با ({) و پایان با (}) می باشد
- دستورات (Statements)
  - همه چیز داخل (“) به عنوان رشته در نظر گرفته می شود
  - هر دستور باید به (;) ختم شود

7

3.2برنامه ساده: چاپ یک خط متن

- رابط کاربری گرافیکی
- GUI ها برای ساده سازی گرفتن داده از کاربر و نمایش داده برای کاربر استفاده می شود.
  - Message boxes
    - داخل فضای نامی **`System.Windows.Forms`** قرار دارد
    - برای اعلان یا نمایش اطلاعات به کاربر استفاده می شود.

8

```
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   }
```

این یک خط خالی است. این مفهومی برای

این آغاز تعریف کلاس Welcome1است. با کلمه کلیدی class و سپس نام کلاس شروع می شود.

خروجی برنامه

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

3.2برنامه ساده: چاپ یک خط متن

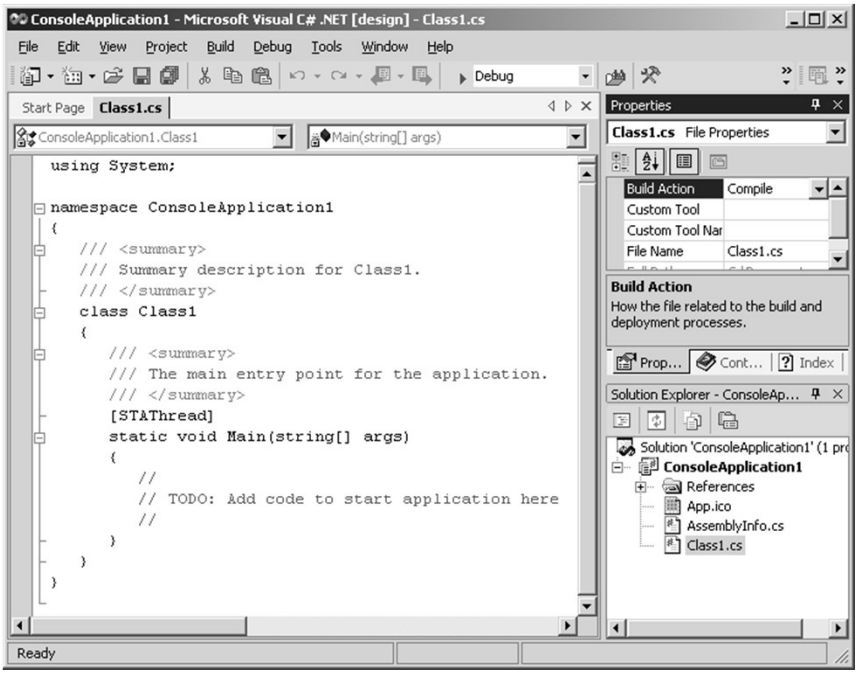


Fig. 3.2 *Visual Studio .NET-generated console application.*

3.2برنامه ساده: چاپ یک خط متن

Fig. 3.3 *Execution of the* **`Welcome1`** *program.*

```
1    // Fig. 3.4: Welcome2.cs
2    // Printing a li
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   }
```

Console.WriteLine جایی استفاده می شود که می خواهید حط پایان یابد. آن موجب می شود که خروجی در یک خط باشد حتی اگر در کد دو خط باشد.

Welcome2.cs

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

خروجی برنامه

13

```
1    // Fig. 3.5: Welcome3.
2    // Printing multiple l
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   }
```

\n برای انتقال مکان نما به خط بعدی استفاده می شود. باعث می شود که خروجی در چندین خط باشد حتی اگر در کد یک خط باشد.

Welcome3.cs

خروجی برنامه

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

3.2برنامه ساده: چاپ یک خط متن

Escape sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line. Any characters output after the carriage return overwrite the previous characters output on that line.
\\	Backslash. Used to print a backslash character.
\"	Double quote. Used to print a double quote (") character.

Fig. 3.6 Some common escape sequences.

15

Welcome4.cs

```
1    // Fig. 3.7: Welcome4.cs
2    // Printing multiple lines in a d
3
4    using System;
5    using System.Windows.Forms
6
7    class Welcome4
8    {
9       static void Main( string[] args )
10      {
11         MessageBox.Show( "Welcome\nto\nC#\nprogramming!" );
12      }
13   }
```

فضای نامی System.Windows.Forms به برنامه نویس اجازه استفاده از کلاس MessageBox را می دهد.

خروجی برنامه

3.2 برنامه ساده: چاپ یک خط متن

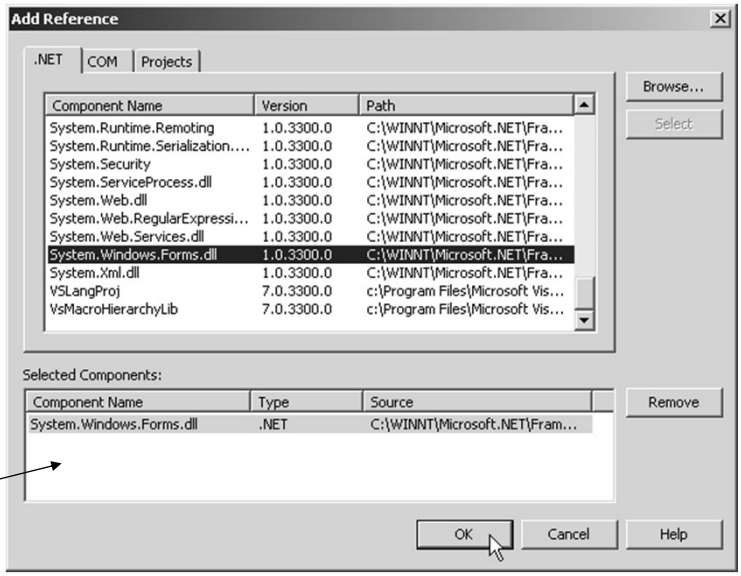


Fig. 3.8 *Adding a reference to an assembly in Visual Studio .NET (part 1).*

3.2 برنامه ساده: چاپ یک خط متن

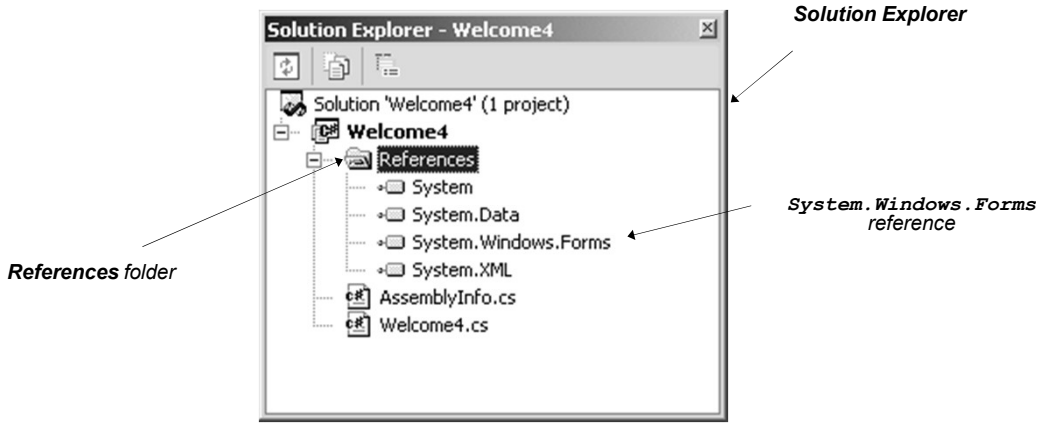


Fig. 3.8 *Adding a reference to an assembly in Visual Studio .NET (part 2).*

17

3.2 برنامه ساده: چاپ یک خط متن

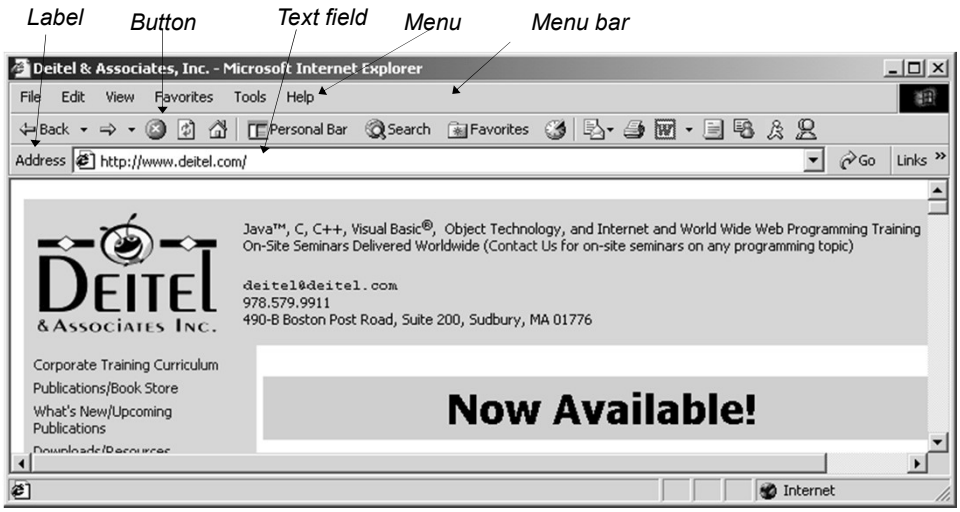


Fig. 3.9 *Internet Explorer's GUI.*

18

3.2 برنامه ساده: چاپ یک خط متن

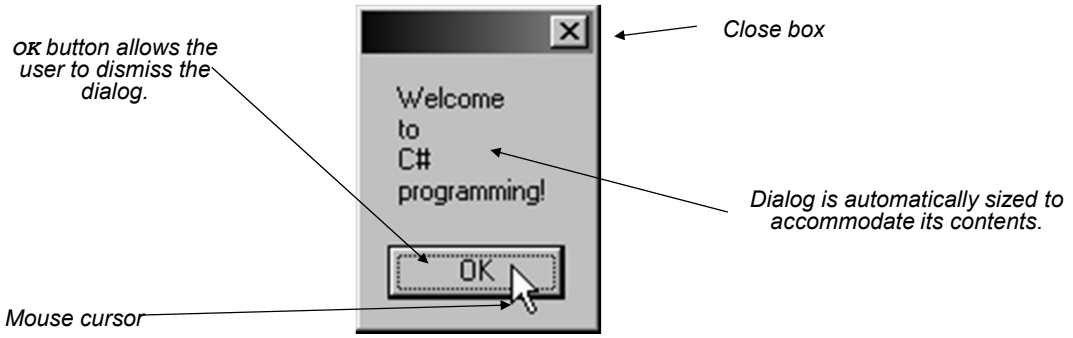


Fig. 3.10 *Dialog displayed by calling* **`MessageBox.Show`**.

3.3یک برنامه ساده دیگر: جمع مقادیر صحیح

- انواع داده اولیه
  - انواع داده ای که C# وجود دارد
    - String, Int, Double, Char, Long
    - ۱۵ نوع داده اولیه (فصل ٤)
  - هر نام نوع داده ای کلمه کلیدی در C# است
  - متغییرهای نوع یکسان می توانند در یک یا چند خط مجزا تعریف شوند
- Console.ReadLine()
  - می تواند برای گرفتن داده از کاربر استفاده شود
- Int32.Parse()
  - برای تبدیل آرگومان به یک مقدار صحیح استفاده می شود

20

Addition.cs

```
1    // Fig. 3.11: Addition.cs
2    // An addition program.
3
4    using System;
5
6    class Addition
7    {
8       static void Main( s
9       {
10         string firstNumb
11                secondNu
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.ReadL
20
21                                   n
22                                   en
23         secondNumber = Console.Read
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
31
```

This is the start of class Addition

Two strin... defined c...

The comment after the declaration is used to briefly ...pose

These are three **int**s that are declared over several lines and only use one semicolon. Each is s...

This line is considered a prompt because it asks the user to input data.

The two numbers are added and stored in the variable sum.

Int32.Parse is used to convert the given string into an integer. It is then stored in a variable.

Console.ReadLine is used to take the users input and place it into a variable.

```
32         // display results
33         Console.WriteLine( "\nThe sum is {0}.", sum );
34
35      } // end method Main
36
37   } // end class Addition
```

Addition.cs

```
Please enter the first integer: 45

Please enter the second integer: 72

The sum is 117.
```

Program Output

Putting a variable out through Console.WriteLine is done by placing the variable after the text while using a marked place to show where the variable should be placed.

3.4 مفاهیم حافظه

- مکان حافظه
  - هر متغیر مکانی از حافظه است
    - شامل نام، نوع، اندازه و مقدار
  - هنگامی ورود یک مقدار جدید مقدار قدیمی از دست می رود
  - متغیرهای مورد استفاده اطلاعات خود را پس از استفاده حفظ می کنند

23

3.4 مفاهيم حافظه

Fig. 3.12 *Memory location showing name and value of variable* ***`number1`****.*

3.5محاسبات

- عملگرهای محاسباتی
  - تمام عملگرها در نماد یکسان استفاده نمی کنند
    - ستاره (*) ضرب است
    - اسلش (/) تقسیم است
    - علامت درصد (٪) عملگر باقیمانده است
    - مثبت (+) و منفی (-) یکسان هستند
  - باید در یک خط مستقیم نوشته شوند
  - توان وجود ندارد
- تقسیم
  - تقسیم می تواند بسته به متغییر استفاده شده متفاوت باشد
    - هنگام تقسیم دو عدد صحیح، نتیجه همیشه به یک عدد صحیح گرد شده پایین می باشد
    - دقت بیشتر استفاده از نتغییری می باشد که اعشار را پشتیبانی کند

25

3.5محاسبات

- ترتیب
  - پرانتز اول انجام می شود
  - تقسیم، ضرب و باقیمانده در مرحله دوم انجام می شود
    - از چپ به راست
  - جمع و تفریق در آخر انجام می شود
    - از چپ به راست

26

3.5 محاسبات

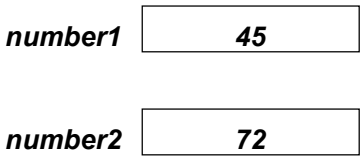


Fig. 3.13 *Memory locations after values for variables* ***`number1`*** *and* ***`number2`*** *have been input.*

3.5 محاسبات

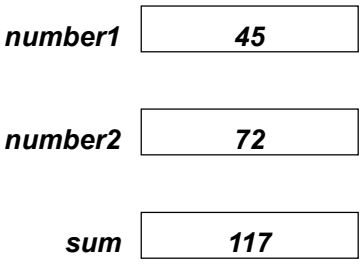


Fig. 3.14 *Memory locations after a calculation.*

3.5 محاسبات

C# operation	Arithmetic operator	Algebraic expression	C# expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y *or* $\frac{x}{y}$	`x / y`
Modulus	%	$r \bmod s$	`r % s`

Fig. 3.15 Arithmetic operators.

3.5 محاسبات

Operator(s)	Operation	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
*, / or %	Multiplication Division Modulus	Evaluated second. If there are several such operators, they are evaluated left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several such operators, they are evaluated left to right.

Fig. 3.16 Precedence of arithmetic operators.

3.5 محاسبات

Step 1.
```
y = 2 * 5 * 5 + 3 * 5 + 7;
    2 * 5 is 10
```
(Leftmost multiplication)

Step 2.
```
y = 10 * 5 + 3 * 5 + 7;
    10 * 5 is 50
```
(Leftmost multiplication)

Step 3.
```
y = 50 + 3 * 5 + 7;
        3 * 5 is 15
```
(Multiplication before addition)

Step 4.
```
y = 50 + 15 + 7;
    50 + 15 is 65
```
(Leftmost addition)

Step 5.
```
y = 65 + 7;
    65 + 7 is 72
```
(Last addition)

Step 6.
```
y = 72;
```
(Last operation—place 72 into y)

Fig. 3.17 Order in which a second-degree polynomial is evaluated.

31

3.6 تصمیم گیری: عملگر های مقایسه ای و رابطه ای

- ساختار **`if`**
  - برای تصمیم گیری بر اساس شرط استفاده می شود.
    - درست: دستورات اجرا می شوند.
    - نادرست: از اجرای دستور صرفنظر می شود.
  - دستور **`if`** نباید به سمیکولون ختم شود.
  - شکل ۳.۱۸ عملگرهای رابطه ای و مقایسه ای را نشان می دهد.
    - نباید عملگرها با فاصله از هم جدا شوند.

32

3.6 تصمیم گیری: عملگر های مقایسه ای و رابطه ای

Standard algebraic equality operator or relational operator	C# equality or relational operator	Example of C# condition	Meaning of C# condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 3.18 Equality and relational operators.

Comparison.cs

```
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                                om user
19                                ter second integer: " );
20
21
22         if ( n
23            Co                          number2 );
24
25         if ( num
26            Console.WriteLine( n
27
28         if ( number1 < number2 )
29            Console.WriteLine( number1 + " < " + number2 );
30
31         if ( number1 > number2 )
32            Console.WriteLine( number1 + " > " +  number2 );
33
```

If number1 is the same as number2 this line is preformed

If number1 does not equal number2 th

If number1 is less than number2 the program wil

If number1 is greater than number2 this line will be preformed

Combining these two methods eliminates temporary string variable.

35

```
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
```

Comparison.cs

Lastly if number1 is greater than or equal to number2 then this code will be executed

Program Output

```
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
```

3.6 تصمیم گیری: عملگر های مقایسه ای و رابطه ای

Operators	Associativity	Type
()	left to right	parentheses
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
=	right to left	assignment

Fig. 3.20 اولویت و شرکت پذیری عملگرها.

فصل چهارم

ساختارهای کنترلی

4.3 شبه کد

- شبه کد (Pseudocode)
  - زبان مصنوعی و غیررسمی
  - به برنامه نویسان برای برنامه ریزی یک الگوریتم کمک می کند
  - مشابه انگلیسی روزمره است
  - یک زبان برنامه نویسی حقیقی نیست
  - می توانید به سادگی با کدهای C# جایگزین کنید

4.4 ساختارهای کنترلی

- Program of control
- برنامه کاری را دستوری را اجرا کرده سپس به سطر بعدی می رود
  - اجرای ترتیبی
- دستورات متفاوتی که ترتیب اجرا را تغییر می دهند
  - ساختارهای انتخابی
    - دستورات **if** و **if/else**
    - دستور **goto**
      - دیگر استفاده نمی شود مگر اینکه شدیداً مورد نیاز باشد.
      - باعث بسیاری از مشکلات خوانایی می شود
  - ساختارهای تکرار
    - حلقه **while** و **do/while** (فصل ۵)
    - حلقه **for** و **foreach** (فصل ۵)

4.4 ساختارهای کنترلی

- فلوچارت
  - جهت ترسیم برنامه استفاده می شود
  - ترتیب رخ دادها را مشخص می کند
    - مستطیل برای انجام عمل استفاده می شود.
    - بیضی برای شروع استفاده می شود
    - دایره ابرای اتصال دادن استفاده می شود
    - لوزی برای تصمیم استفاده می شود
  - ترکیب ساختارهای کنترلی
    - پشته سازی (Stacking)
      - یکی پس از دیگری با قرار دادن یک
    - تودر تو سازی (Nesting)
      - قرار دادن یک ساختار درون ساختار دیگری.

4.4 ساختارهای کنترلی

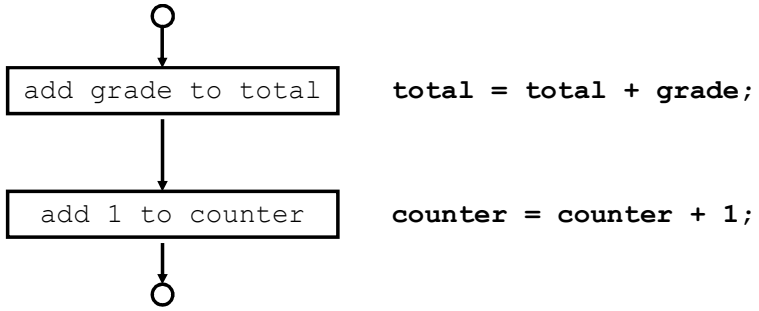


Fig. 4.1 Flowcharting C#'s sequence structure.

4.4 ساختارهای کنترلی

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	get
goto	if	implicit	in	int
interface	internal	is	lock	long
namespace	new	null	object	operator
out	override	params	private	protected
public	readonly	ref	return	sbyte
sealed	set	short	sizeof	stackalloc
static	string	struct	switch	this
throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using
value	virtual	void	volatile	while

Fig. 4.2 کلمات کلیدی C#

4.5 ساختار انتخاب `if`

- ساختار **`if`**
  - موجب می شد برنامه انتخاب کند
  - انتخاب بر اساس شرایط انجام می شود
    - شرط بصورت **`bool`** ارزیابی می شود
      - True: عمل انجام شود
      - False: از انجام عمل صرفنظر شود
  - یک نقطه ورود/خروج
  - دستورات به سمی کولون نیاز دارند.

4.5 ساختار انتخاب `if`

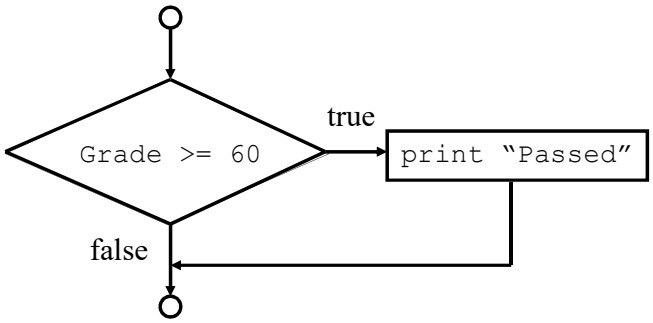


Fig. 4.3 Flowcharting a single-selection **`if`** structure.

4.6 ساختار انتخاب `if/else`

- ساختار **`if/else`**
  - در صورت نادرست بودن شرط جریان متفاوتی می تواند دنبال شود
  - به جای یک عمل دو انتخاب وجود دارد
  - ساختارهای تو در تو می توانند چند حالت را کنترل کند.
  - ساختارهایی که بیش از یک سطر کد داشته باشند نیاز به ({) دارند.
    - می تواند موجب خطا شود
      - Fatal logic error
      - Nonfatal logic error

4.6 ساختار انتخاب if/else

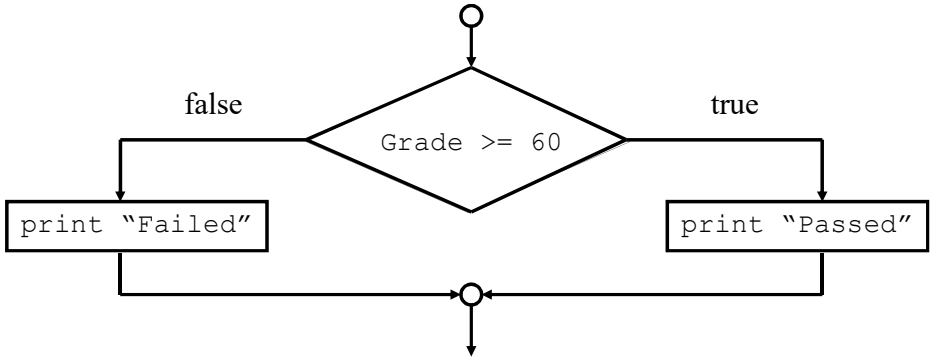


Fig. 4.4 Flowcharting a double-selection **if**/**else** structure.

عملگر شرطی (:?)

- عملگر شرطی (:?)
  - تنها عملگر سه گانه C#
  - شبیه ساختار **`if/else`** می باشد
  - شکل کلی آن بصورت زیر است
    - (*boolean value* ? *if true* : *if false*)

4.7 ساختار تکرار `while`

- ساختار تکرار
  - دستوری که باید تکرار شود
    - دستورات while ادامه می یابند اگر true باشد
    - دستورات پایان می یابد اگر false باشد
  - ممکن است شامل یک سطر یا یک بدنه کد باشد
    - باید شرط پایان یابد
      - حلقه بی نهایت

4.7 ساختار تکرار `while`

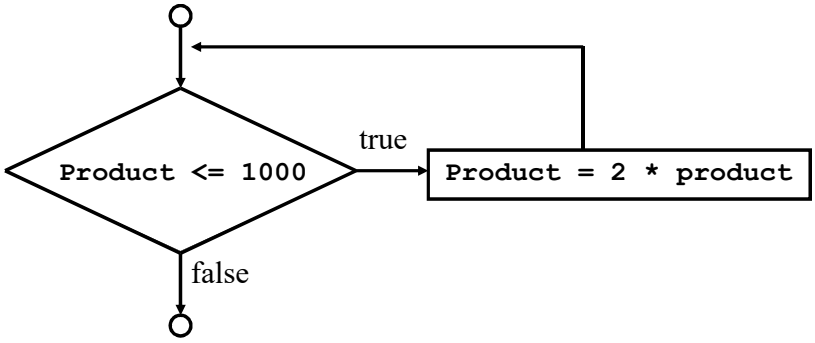


Fig. 4.5 Flowcharting the **`while`** repetition structure.

4.8 فرموله کردن الگو ریتم: بررسی موردی ۱ (تکرار کنترل شده با شمارنده)

- تکرار کنترل شده با شمارنده
  - Used to enter data one at a time
    - Constant amount
  - شمارنده برای مشخص کردن زمان شکستن حلقه استفاده می شود
  - زمانی که شمارنده به یک مقدار مشخص می رسد، دستورات پایان می یابند.

4.8 فرموله کردن الگو ریتم: مطالعه موردی ۱ (تکرار کنترل شده با شمارنده)

Set total to zero
Set grade counter to one

While grade counter is less than or equal to ten
    Input the next grade
    Add the grade into the total
    Add one to the grade counter

Set the class average to the total divided by ten
Print the class average

Fig. 4.6 Pseudocode algorithm that uses counter-controlled repetition to solve the class-average problem.

Average1.cs

```
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,   // numbe                    d
12             gradeValue,     // grade
13             average;        // average of all grades
14
15         // initialization phase
16         total = 0;          // clear total
17         gradeCounter = 1;   // prepare to l
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( Conso
27
28            // add gradeValue to total
29            total = total + gradeValue;
30
31            // add 1 to gradeCounter
32            gradeCounter = gradeCounter + 1;
33         }
```

Initialize total to 0

Initialize gradeCounter to 1

The **`while`** loop will loop through 10 times to get the grades of the 10 students

Prompt the user to enter a grade

Accumulate the total of the 10 grades

Add 1 to the counter so the loop will eventually end

```
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
```

Average1.cs

Divide the total by ten to get the average of the ten grades

Display the results

Program Output

```
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
```

4.9 تدوین الگوریتم ها بالا به پایین، پالایش گام به گام : مطالعه موردی ۲ (نگهبان کنترل تکرار)

- تکرار کنترل شده با نگهبان
  - میزان زمان دلخواهی ادامه پیدا می کند.
  - مقدار نگهبان
    - موجب می شود حلقه شکسته شود
    - از برخورد اجتناب شود
    - When flag value = user entered value
- ایجاد شبه کد
  - با یک وظیفه شروع می شود
  - انرا به چندین وظیفه تقسیم کنید
  - شکستن وظیفه را تا جایی ادامه دهید که وظیفه ایجاد شده ساده باشد.
- Casting
  - اجازه می دهد تا یک متغیر به طور موقت به عنوان یکی دیگر مورد استفاده قرار گیرد

4.9 تدوین الگوریتم ها بالا به پایین، پالایش گام به گام : مطالعه موردی ۲ (نگهبان کنترل تکرار)

Initialize total to zero
Initialize counter to zero

Input the first grade (possibly the sentinel)

While the user has not as yet entered the sentinel
    Add this grade into the running total
    Add one to the grade counter
    Input the next grade (possibly the sentinel)

If the counter is not equal to zero
    Set the average to the total divided by the counter
    Print the average
Else
    Print "No grades were entered"

Fig. 4.8 Pseudocode algorithm that uses sentinel-controlled repetition to solve the class-average problem.

Average2.cs

```
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 ent
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
```

The variable average is set to a double so that it can be more exact and have an answer with decimals

Variables gradeCounter and total are set to zero at the beginning

Get a value from the user and store it in gradeValue

```
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 u
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          else
50          {
51             Console.WriteLine( "\nNo grades were entered" );
52          }
53
54       } // end method Main
55
56    } // end class Average2
```

Have the program loop as long as

Accumulate the total of the grades

Add 1 to the counter in order to know the student count

Prompt the user for another grade, this time it is in the loop so it can happen repeatedly

Divide the total by the number of times the program looped to find the average

Display the average

Inform user if no grades were entered

```
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
```

Average2.cs
Program Output

4.10 الگوریتم تدوین با بالا به پایین، گام به گام پالایش: مطالعه موردی ۳ (سازه های کنترل تو در تو)

- Nesting
  - درج یک ساختار کنترل در داخل یکی دیگر
    - چندین حلقه
    - حلقه ها با دستور **if**

4.10 الگوریتم تدوین با بالا به پایین، گام به گام پالایش: مطالعه موردی ۳ (سازه های کنترل تو در تو)

```
Initialize passes to zero
Initialize failures to zero
Initialize student to one

While student counter is less than or equal to ten
        Input the next exam result

        If the student passed
                Add one to passes
        Else
                Add one to failures

        Add one to student counter

Print the number of passes
Print the number of failures

If more than eight students passed
        Print "Raise tuition"
```

Fig. 4.10 Pseudocode for examination-results problem.

Analysis.cs

```
1    // Fig. 4.11: Analysis.cs
2    // Analysis of Examination Results.
3
4    using System;
5
6    class Analysis
7    {
8       static void Main( string[] args )
9       {
10         int passes = 0,             // number of
11             failures = 0,           // number of
12             student = 1,            // student counter
13             result;                 // one exam result
14
15         // process 10 students; counter-controlled loop
16         while ( student <= 10 )
17         {
18            Console.Write( "Enter result (1=pass, 2=fail): " );
19            result = Int32.Parse( Console.ReadLine() );
20
21            if ( result == 1 )
22               passes = passes + 1;
23
24            else
25               failures = failures + 1;
26
27            student = student + 1;
28         }
29
```

Initialize both passes and failures to 0
Set the student count to 1

A while loop that will loop 10 times

A nested if statement that determines which counter should be added to

If the user enters 1 add one to passes

If the user enters 2 then add one to failures

Keep track of the total number of students

```
30         // termination phase
31         Console.WriteLine();
32         Console.WriteLine( "Passed: " + passes );
33         Console.WriteLine( "Failed: " + failures );
34
35         if ( passes > 8 )
36            Console.WriteLine( "Raise Tuition\n" );
37
38      } // end of method Main
39
40   } // end of class Analysis
```

Analysis.cs

Display the results to the user

If the total number of passes was greater than 8 then also tell the user to raise the tuition

```
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 2
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1

Passed: 9
Failed: 1
Raise Tuition
```

Program Output

```
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 2
Enter result (1=pass, 2=fail): 2
Enter result (1=pass, 2=fail): 2
Enter result (1=pass, 2=fail): 2
Enter result (1=pass, 2=fail): 2
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1

Passed: 5
Failed: 5
```

Analysis.cs
Program Output

4.11 عملگرهای انتساب

- عملگرهای انتساب
  - می توان کدها را کاهش داد
    - x += 2 معادل x = x + 2 می باشد.
  - می تواند با تمام عملگر های ریاضی انجام شود
    - ++, -=, *=, /=, and %=

4.11 عملگرهای انتساب

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`

Fig. 4.12 Arithmetic assignment operators.

4.12 عملگرهای افزایشی و کاهشی

- عملگر افزایشی
  - برای افزودن یک واحد به متغییر استفاده می شود.
  - x++
  - همانند x = x + 1
- عملگر کاهشی
  - برای کاستن یک واحد از متغییر استفاده می شود.
  - y--
- پیش افزایش (Pre-increment) در مقابل پس افزاش (post-increment)
  - x++ یا x--
    - اول عمل انجام می شود سپس یک واحد افزوده یا کاسته می شود.
  - ++x یا --x
    - اول یک واحد به مقدار افزوده شده یا یک مقدار کاسته می شود سپس عمل انجام می شود.

4.12 عملگرهای افزایشی و کاهشی

Operator	Called	Sample expression	Explanation
`++`	preincrement	`++a`	Increment `a` by 1, then use the new value of `a` in the expression in which `a` resides.
`++`	postincrement	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by 1.
`--`	predecrement	`--b`	Decrement `b` by 1, then use the new value of `b` in the expression in which `b` resides.
`--`	postdecrement	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by 1.

Fig. 4.13 The increment and decrement operators.

Increment.cs

```
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 t
15         Console.WriteLine( c );  // print 6
16
17         Console.WriteLine();     // skip a lin
18
19         c = 5;
20         Console.WriteLine( c );  // print
21         Console.WriteLine( ++c ); // preincr
22         Console.WriteLine( c );  // print 6
23
24      } // end of method Main
25
26   } // end of class Increment
```

Declare variable c

Set c equal to 5

Display c (5) then add 1

Display c (6)

c is set to 5

Display c (5)

Add 1 then display c (6)

Display c (6)

Program Output

```
5
5
6

5
6
6
```

4.12 عملگرهای افزایشی و کاهشی

Operators	Associativity	Type
`()` `++ --`	left to right right to left	parentheses unary postfix
`++ -- + - (type)`	right to left	unary prefix
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

Fig. 4.15 Precedence and associativity of the operators discussed so far in this book.

4.13 مقدمه ای بر برنامه نویسی ویندوز

- ارث بری (Inheritance)
  - کلاس پایه
    - یک کلاس که از آن کلاس دیگری ارث می برد
  - کلاس مشتق
    - کلاسی که از کلاس دیگر ارث بری کرده است
  - کلاسایی پایه های یک کلاس را ارث می برند.
    - Attributes (data)
    - Behaviors (methods)
  - از دوباره نویسی کدها جلوگیری می کند.

4.13 مقدمه ای بر برنامه نویسی ویندوز

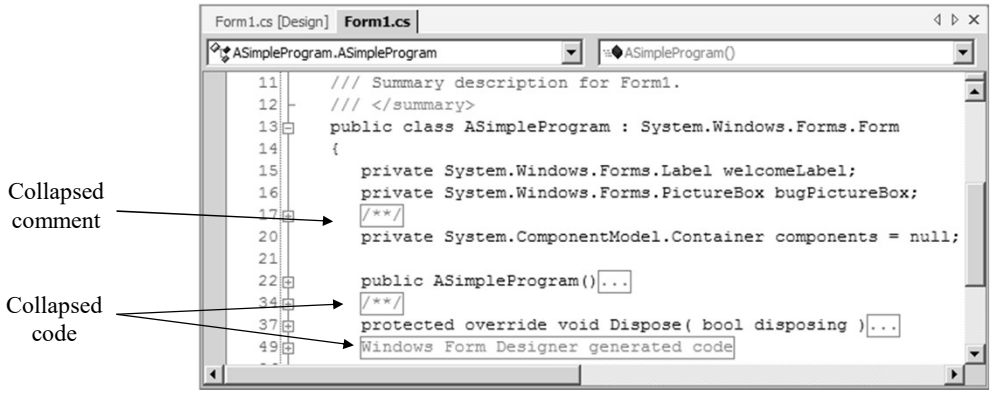
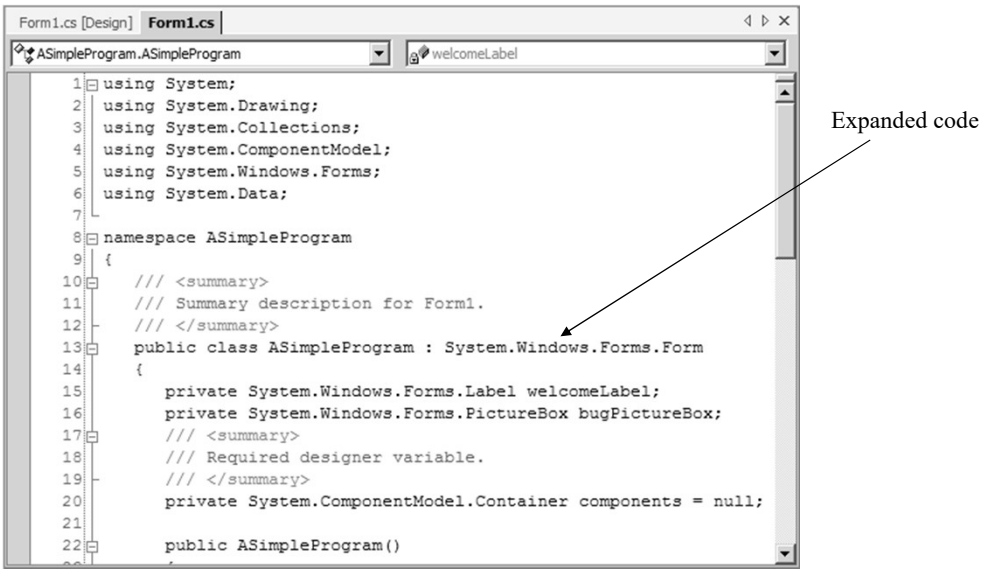


Fig. 4.16 IDE showing program code for Fig. 2.15.

4.13 مقدمه ای بر برنامه نویسی ویندوز




```
1  using System;
2  using System.Drawing;
3  using System.Collections;
4  using System.ComponentModel;
5  using System.Windows.Forms;
6  using System.Data;
7
8  namespace ASimpleProgram
9  {
10    /// <summary>
11    /// Summary description for Form1.
12    /// </summary>
13    public class ASimpleProgram : System.Windows.Forms.Form
14    {
15       private System.Windows.Forms.Label welcomeLabel;
16       private System.Windows.Forms.PictureBox bugPictureBox;
17       /// <summary>
18       /// Required designer variable.
19       /// </summary>
20       private System.ComponentModel.Container components = null;
21
22       public ASimpleProgram()
```

Fig. 4.17 Windows Form Designer generated code when expanded.

4.13 مقدمه ای بر برنامه نویسی ویندوز

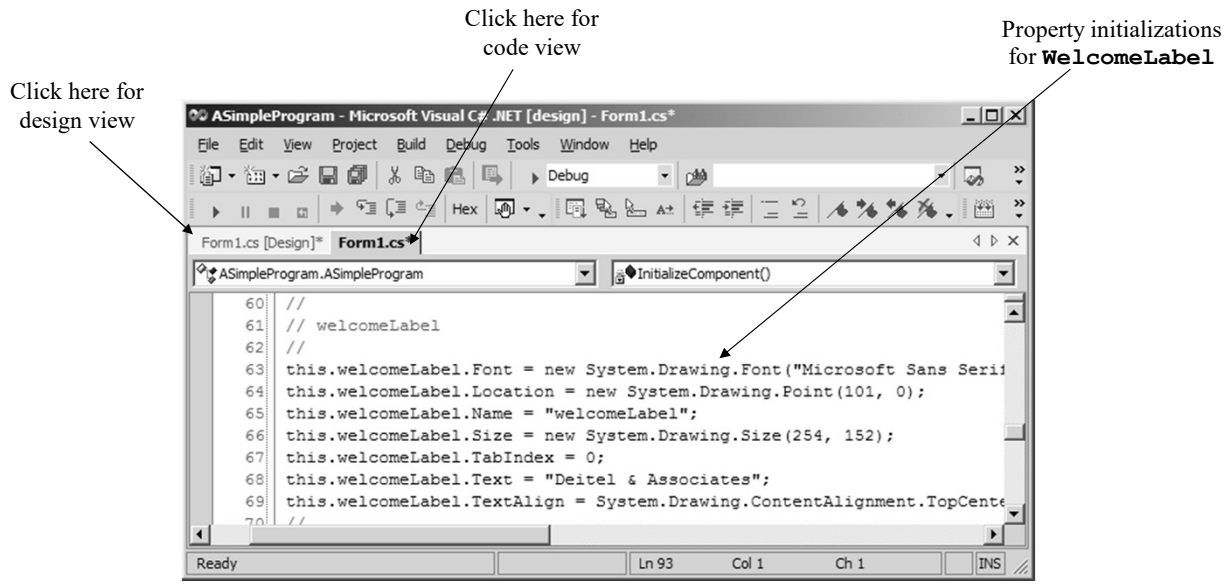


Fig. 4.18 Code generated by the IDE for **`welcomeLabel`**.

4.13 مقدمه ای بر برنامه نویسی ویندوز

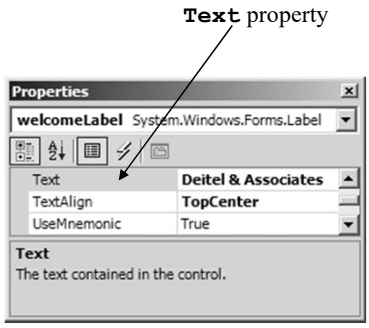


Fig. 4.19 Using the **`Properties`** window to set a property value.

4.13 مقدمه ای بر برنامه نویسی ویندوز

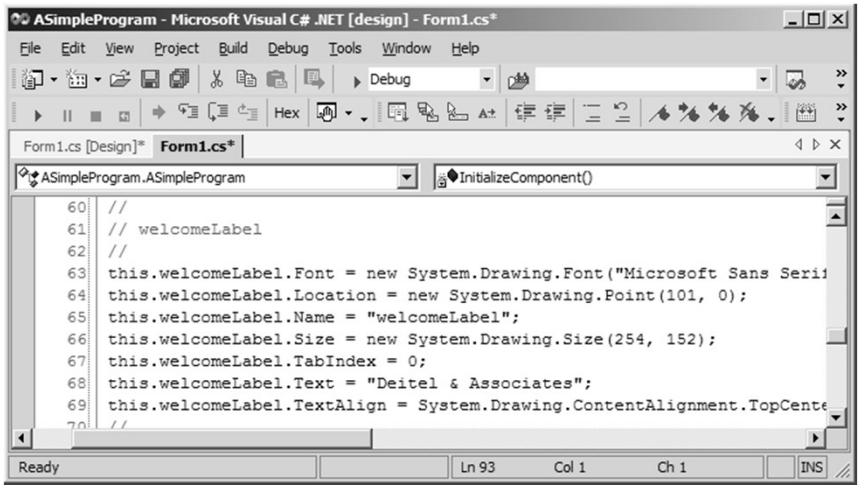


Fig. 4.20 Windows Form Designer generated code reflecting new property values.

4.13 مقدمه ای بر برنامه نویسی ویندوز

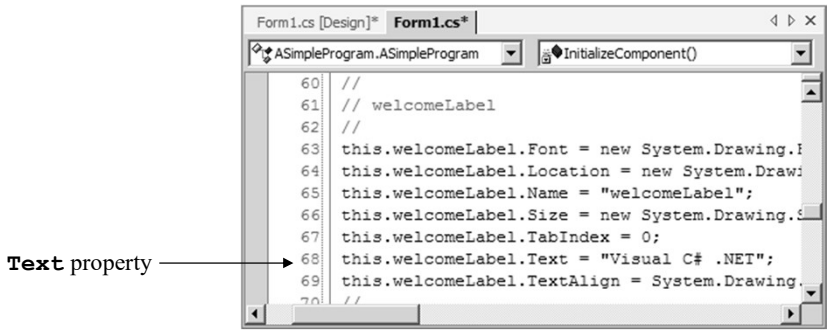


Fig. 4.21 Changing a property in the code view editor.

4.13 مقدمه ای بر برنامه نویسی ویندوز

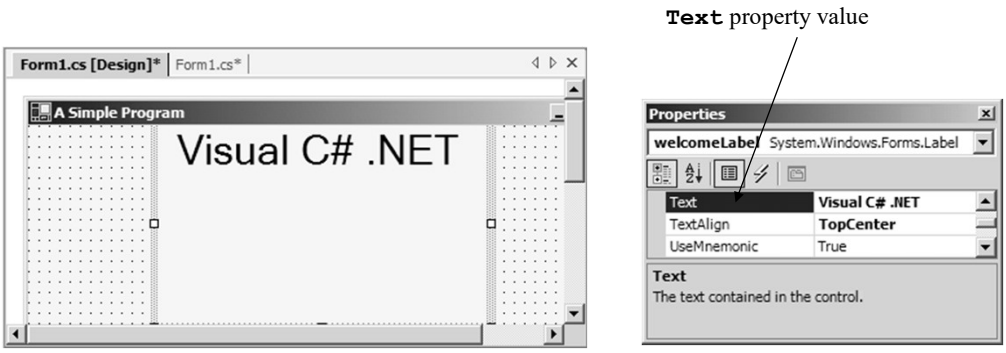


Fig. 4.22 New `Text` property value reflected in design mode.

4.13 مقدمه ای بر برنامه نویسی ویندوز

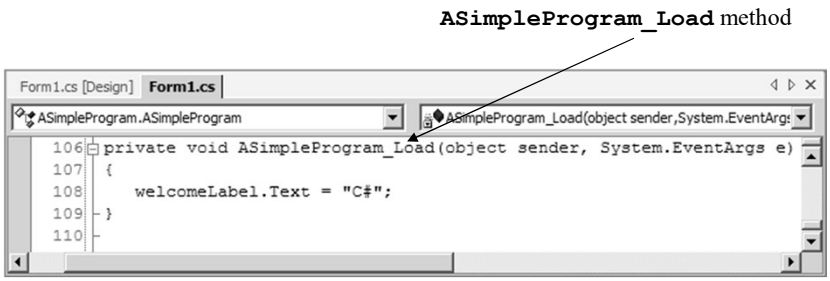


Fig. 4.23 Method **`ASimpleProgram_Load`**.

4.13 مقدمه ای بر برنامه نویسی ویندوز

Fig. 4.24 Changing a property value at runtime.

فصل پنجم

ساختارهای کنترلی : بخش دوم

فصل پنجم ساختارهای کنترلی : بخش دوم

Outline

5.2 تکرار کنترل شده با شمارنده اصلی

- تکرار کنترل شده با شمارنده
  - متغیر را کنترل می کند
    - متغیر برای تعیین اینکه آیا حلقه باید تکرار شود استفاده می شود.
  - متغیر کنترل کننده مقدار دهی اولیه شود.
  - متغییر را کاهش/افزایش دهید
  - شرط
    - کی حلقه ادامه پیدا کند

WhileCounter.cs

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

This is where the counter variable is initialized. It is set to 1.

The loop will continue until counter is greater than five (it will stop once it gets to six)

The counter is incremented and 1 is added to it

Program Output

```
1
2
3
4
5
```

5.3 ساختار تکرار for

- ساختار تکرار for
  - Syntax: **for** (Expression1; Expression2; Expression3)
    - Expression1 = متغیرهای کنترلی را نامگذاری می کند
      - می تواند شامل چندین متغیر باشد
    - Expression2 = شرط ادامه حلقه
    - Expression3 = افزایش/کاهش
      - اگر Expression1 چندید متغیر داشته باش، از این رو Expression3 نیز چندین متغیر خواهد داشت
      - counter++ و ++counter تفاوتی ندارند
  - میدان متغیر
    - Expression1 می تواند در بدنه حلقه **for استفاده شود.**
    - زمانی که حلقه پایان می یابد متغیر از بین می رود.

5.3 `for` Repetition Structure

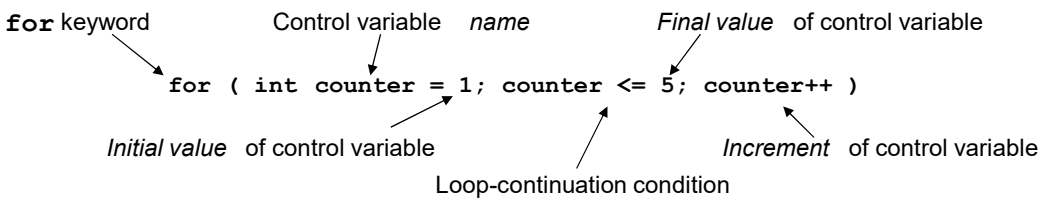


Fig. 5.3 Components of a typical `for` header.

5.3 `for` Repetition Structure

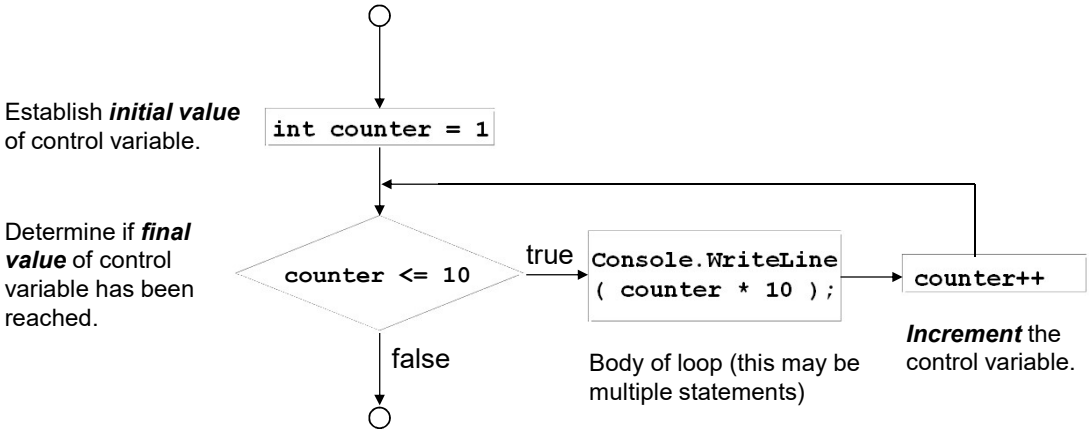


Fig. 5.4 Flowcharting a typical `for` repetition structure.

87

```
1    // Fig. 5.2: ForCounter.cs
2    // Counter
3
4    using Syst
5
6    class ForCounter
7    {
8       static void Main(
9       {
10         // initializat                    nting
11         // are all included in the for structure
12         for ( int counter = 1; counter <= 5; counter++ )
13            Console.WriteLine( counter );
14      }
15   }
```

for structure.

This is where the counter variable is initialized. It is set to 1

The counter is incremented (1 is added to it)

The loop w
greater than five (it will stop once it gets to six)

ForCounter.cs

```
1
2
3
4
5
```

Program Output

5.4 مثال استفاده از ساختار `for`

- افزایش/کاهش
  - هنگام افزودن
    - در اکثر حالات > یا => استفاده می شود
  - هنگام کاستن
    - در اکثر حالات < یا =< استفاده می شود
- Message boxes
  - Buttons
    - **`OK`**
    - **`OK Cancel`**
    - **`Yes No`**
    - **`Abort Retry Ignore`**
    - **`Yes No Cancel`**
    - **`Retry Cancel`**

5.4 مثال استفاده از ساختار `for`

- Massages boxes
  - Icons
    - **`Exclamation`**
    - **`Question`**
    - **`Error`**
    - **`Information`**
- Formatting
  - (*variable* : *format*)
    - جدول ۵.۹ برخی کدهای قالب بندی را نشان می دهد

Sum.cs

```
1    // Fig. 5.5: Sum.cs
2    // Summation with the for structure.
3
4    using System;
5    using System.Windows.Forms;
6
7    class Sum
8    {
9       static void Main( string[] args )
10      {
11         int sum = 0;
12
13         for ( int number = 2;
14            sum += number;
15
16         MessageBox.Show( "The sum is " + sum,
17            "Sum Even Integers from 2 to 100",
18            MessageBoxButtons.OK,
19            MessageBoxIcon.Information );
20
21      } // end method Main
22
23   } // end class Sum
```

The counter. It is initialized to 2

Once the number is greater than 100 the loop breaks

Increments number by 2 every time the loop starts over

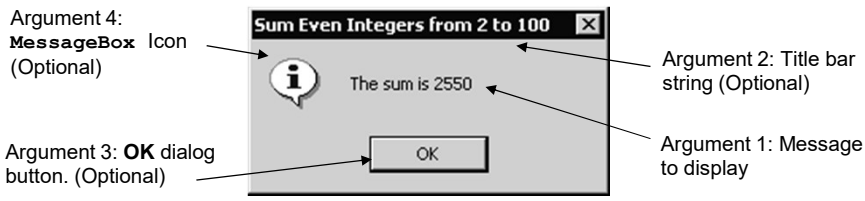
The caption of the message box

The title of the message box

Displays a message box with an **OK** button

Has the message box contain an information icon

Program Output



5.4 مثال استفاده از ساختار `for`

MessageBox Icons	Icon	Description
`MessageBoxIcon.Exclamation`		Displays a dialog with an exclamation point. Typically used to caution the user against potential problems.
`MessageBoxIcon.Information`		Displays a dialog with an informational message to the user.
`MessageBoxIcon.Question`		Displays a dialog with a question mark. Typically used to ask the user a question.
`MessageBoxIcon.Error`		Displays a dialog with an **`x`** in a red circle. Helps alert user of errors or important messages.

Fig. 5.6 Icons for message dialogs.

5.4 مثال استفاده از ساختار `for`

MessageBox Buttons	Description
`MessageBoxButton.``OK`	Specifies that the dialog should include an **OK** button.
`MessageBoxButton.``OKCancel`	Specifies that the dialog should include **OK** and **Cancel** buttons. Warns the user about some condition and allows the user to either continue or cancel an operation.
`MessageBoxButton.``YesNo`	Specifies that the dialog should contain **Yes** and **No** buttons. Used to ask the user a question.
`MessageBoxButton.``YesNoCancel`	Specifies that the dialog should contain **Yes**, **No** and **Cancel** buttons. Typically used to ask the user a question but still allows the user to cancel the operation.
`MessageBoxButton.``RetryCancel`	Specifies that the dialog should contain **Retry** and **Cancel** buttons. Typically used to inform a user about a failed operation and allow the user to retry or cancel the operation.
`MessageBoxButton.``AbortRetryIgnore`	Specifies that the dialog should contain **Abort**, **Retry** and **Ignore** buttons. Typically used to inform the user that one of a series of operations has failed and allow the user to abort the series of operations, retry the failed operation or ignore the failed operation and continue.

Fig. 5.7 Buttons for message dialogs.

Interest.cs

```
1    // Fig. 5.8: Interest.cs
2    // Calculating compound interest.
3
4    using System;
5    using System.Windows.Forms;
6
7    class Interest
8    {
9       static void Main( string[] a
10      {
11         decimal amount, principal
12         double rate = .05;
13         string output;
14
15         output = "Year\tAmount on deposit\n";
16
17         for ( int yea
18         {
19            amount = principal *
20                decimal ) Math.Pow(
21
22            output += year + "\t" +
23               String.Format( "{0:C}"
24         }
25
26         MessageBox.Show( output, "Compound Interest",
27            MessageBoxButtons.OK, MessageBoxIcon.Information );
28
29      } // end method Main
30
31   } // end class Interest
```

Loops through 10 times starting at 1 and ending at 10, adding 1 to the counter (year) each time

Formats amount to have a currency formatting ($0.00)

Insert a Tab

Creates a message box that displays the output with a title of "Compound Interest" has an **OK** button and an information icon

Interest.cs
Program Output

5.4 مثال استفاده از ساختار for

Format Code	Description
C or **c**	Formats the string as currency. Precedes the number with an appropriate currency symbol (**$** in the US). Separates digits with an appropriate separator character (comma in the US) and sets the number of decimal places to two by default.
D or **d**	Formats the string as a decimal. Displays number as an integer.
N or **n**	Formats the string with commas and two decimal places.
E or **e**	Formats the number using scientific notation with a default of six decimal places.
F or **f**	Formats the string with a fixed number of decimal places (two by default).
G or **g**	General. Either **E** or **F**.
X or **x**	Formats the string as hexadecimal.

Fig. 5.9 `string` formatting codes.

5.5 ساختار انتخاب چندگانه `switch`

- دستور **`switch`**
  - عبارت ثابت
    - String
    - Integral
  - حالت ها
    - case ‘*x*’ :
    - Use of constant variable cases
    - case های خالی
    - case پیش فرض
- دستور **`break`**
  - از دستور **`switch`** خارج می شود.

SwitchTest.cs

```
1    // Fig. 5.10: SwitchTest.cs
2    // Counting letter grades.
3
4    using System;
5
6    class SwitchTest
7    {
8       static void Main( string[] args )
9       {
10
11
12
13
14             dCount = 0,  // number of Ds
15             fCount = 0;  // number of Fs
16
17          for ( int i = 1; i <= 10; i++ )
18          {
19             Console.Write( "Enter a letter grad
20
21
22             switch ( grade )
23             {
24                case 'A':   // grade is uppercas
25                case 'a':   // or lowercase a
26                   ++aCount;
27                   break;
28
29                case 'B':   // grade is uppercas
30                case 'b':   // or lowercase b
31                   ++bCount;
32                   break;
33
```

A **for** loop that initializes i to 1, loops 10 times and increments i by one each time

Each of these variables acts as a counter so they are initialized to zero

Prompt the user for a grade and store it into the grade variable

The start of the **switch** statement. The grade variable is used as the data to be tested for

Both cases add one to aCount

case 'A' is empty so it is the same as **case** 'a'

The break statement is used to exit the switch statement and not perform the rest of the operations

Both case 'B' and case 'b' add one to the bCount variable

SwitchTest.cs

```
34                case 'C':   // g
35                case 'c':   // o
36                   ++cCount;
37                   break;
38
39                case 'D':   // grade is uppercase D
40                case 'd':   // or lower
41                   ++dCount;
42                   break;
43
44                case 'F':   // grade is uppercase F
45                case 'f':   // or lowerca
46                   ++fCount;
47                   break;
48
49                default:    // processes all other char
50                   Console.WriteLine(
51                      "Incorrect letter grade entered."
52                      "\nGrade not added to totals." );
53                   break;
54
55             } // end switch
56
57          } // end for
58
59          Console.WriteLine(
60             "\nTotals for each letter grade are:\nA: {0}" +
61             "\nB: {1}\nC: {2}\nD: {3}\nF: {4}", aCount, bCount,
62             cCount, dCount, fCount );
63
64       } // end method Main
65
66    } // end class SwitchTest
```

Both cases add 1 to cCount

If grade equals D or d add one to dCount

Add one to fCount if grade equals F or f

If non of the cases are equal to the value of grade then the **`default`** case is executed

Display the results

```
Enter a letter grade: a
Enter a letter grade: A
Enter a letter grade: c
Enter a letter grade: F
Enter a letter grade: z
Incorrect letter grade entered.
Grade not added to totals.
Enter a letter grade: D
Enter a letter grade: d
Enter a letter grade: B
Enter a letter grade: a
Enter a letter grade: C

Totals for each letter grade are:
A: 3
B: 1
C: 2
D: 2
F: 1
```

SwitchTest.cs
Program Output

5.5 `switch` Multiple-Selection Structure

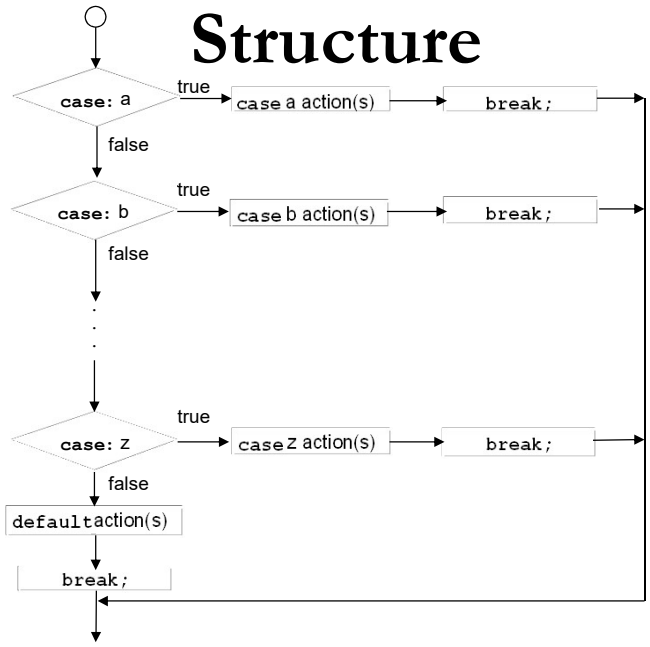


Fig. 5.11 Flowcharting the **`switch`** multiple-selection structure.

5.6 ساختار تکرار do/while

- حلقه های **while** در مقابل خلقه های **do/while**
  - استفاده از یک حلقه **while**
    - شرط تست می شود
    - عمل انجام می شود
    - حلقه می تواند به طور کلی انجام نشود
  - استفاده از یک حلقه **do/while**
    - عمل انجام می شود
    - سپس شرط حلقه تست می شود
    - حلقه حد اقل یک بار اجرا خواهد شد.
    - همیشه از براکت ({) برای جلوگیری از سردرگمی استفاده می شود

DoWhileLoop.cs

```
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
```

The counter is initialized to one

These actions are performed at least one

The incrementing task

Continue looping as long as counter is less than 6

Program Output

```
1
2
3
4
5
```

5.6 ساختار تکرار do/while

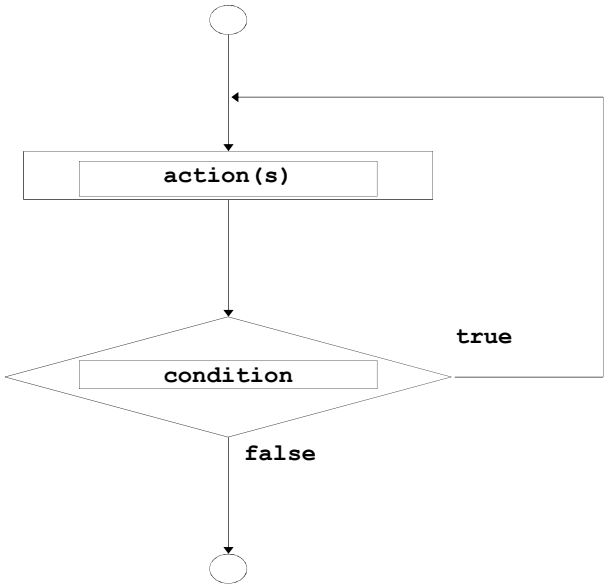


Fig. 5.13 Flowcharting the **do**/**while** repetition structure.

5.7 دستور **break** و **continue**

- استفاده
  - برای تغییر دادن جریان کنترل استفاده می شود
  - دستور **break**
    - برای خروج زود تر از موعد از حلقه استفاده می شود.
  - دستور **continue**
    - برای پرش از دستورات باقی مانده و شروع حلقه از اولین دستور استفاده می شود
  - برنامه را می توان بدون استفاده از آنها به اتمام برد.

BreakTest.cs

```
1    // Fig. 5.14: BreakTest.cs
2    // Using the break statement in a for structure.
3
4    using System;
5    using System.Windows.Forms;
6
7    class BreakTest
8    {
9       static void
10      {
11         string o
12         int count;
13
14         for ( count = 1; count <= 10; count++ )
15         {
16            if ( count == 5 )
17               break;
18                                    // if count == 5
19
20                                 " ";
21
22         } // end for loop
23
24         output += "\nBroke out of loop at count = " + count;
25
26         MessageBox.Show( output, "Demonstrating the break statement",
27            MessageBoxButtons.OK, MessageBoxIcon.Information );
28
29      } // end method Main
30
31   } // end class BreakTest
```

A loop that starts at one, goes to ten, and increments by one

If count = 5 then break out of the loop

Display the last value that the counter was at before it broke

Displays a message box the displays the output, has a title of “demonstrating the break statement,” uses an **OK** button, and displays an information icon

BreakTest.cs
Program Output

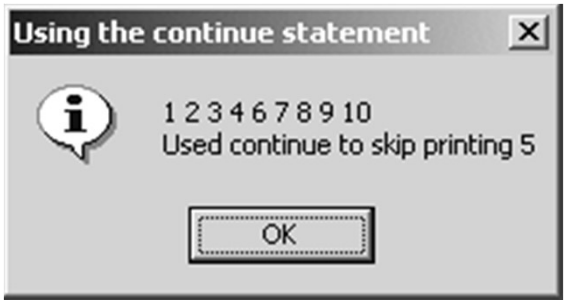
ContinueTest.cs

```
1    // Fig. 5.15: ContinueTest.cs
2    // Using the continue statement in a for structure.
3
4    using System;
5    using System.Windows.Forms;
6
7    class ContinueTest
8    {
9       static void Main( stri
10      {
11         string output = "";
12
13         for ( int count = 1; count <= 10; count++ )
14         {
15            if ( count == 5 )
16               continue;
17
18
19            output += count
20         }
21
22         output += "\nUsed continue to skip printing 5";
23
24         MessageBox.Show( output, "Using the continue statement",
25            MessageBoxButtons.OK, MessageBoxIcon.Information );
26
27      } // end method Main
28
29   } // end class ContinueTest
```

A loop that starts at 1, goes to 10, and

If count = 5 then continue looping causing the program to skip the rest of the loop

Create a message box that displays the output, has the title “using the continue statement,” uses an **OK** button, and displays an information icon.



ContinueTest.cs
Program Output

5.8 عملگرهای منطقی و شرطی

- عملگرها
  - Logical AND (&)
  - Conditional AND (&&)
  - Logical OR (|)
  - Conditional OR (||)
  - Logical XOR (^)
  - Logical NOT (!)
- برای افزودن چندین شرط به دستور استفاده می شود.

5.8 عملگرهای منطقی و شرطی

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

Fig. 5.16 Truth table for the && (logical AND) operator.

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

Fig. 5.17 Truth table for the || (logical OR) operator.

5.8 عملگرهای منطقی و شرطی

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

Fig. 5.18 Truth table for the logical exclusive OR (^) operator.

expression	!expression
`false`	`true`
`True`	`false`

Fig. 5.19 Truth table for operator ! (logical NOT).

LogicalOperators.cs

```
1    // Fig. 5.20: LogicalOperators.cs
2    // Demonstrating the logical operators.
3    using System;
4
5    class LogicalOperators
6    {
7       // main entry point for application
8       static void Main( string[] args )
9       {
10         // testing the conditional AND operator (&&)
11         Console.WriteLine( "Conditional AND (&&)" +
12            "\nfalse && false: " + ( false && false ) +
13            "\nfalse && true:  " + ( false && true ) +
14            "\ntrue && false:  " + ( true && false ) +
15            "\ntrue && true:   " + ( true && true ) );
16
17         // testing the conditional OR operator (||)
18         Console.WriteLine( "\n\nConditional OR (||)" +
19            "\nfalse || false: " + ( false || false ) +
20            "\nfalse || true:  " + ( false || true ) +
21            "\ntrue || false:  " + ( true || false ) +
22            "\ntrue || true:   " + ( true || true ) );
23
24         // testing the logical AND operator (&)
25         Console.WriteLine( "\n\nLogical AND (&)" +
26            "\nfalse & false: " + ( false & false ) +
27            "\nfalse & true:  " + ( false & true ) +
28            "\ntrue & false:  " + ( true & false ) +
29            "\ntrue & true:   " + ( true & true ) );
30
```

Outputs a truth table for the conditional AND operator (&&)

Only true if both inputs are true

Outputs a truth table for the conditional OR operator (||)

Only false if both inputs are false

Outputs a truth table for the logical AND operator (&)

The result is only true if both are true

```
31          // testing the logical OR operator (|)
32          Console.WriteLine( "\n\nLogical OR (|)" +
33             "\nfalse | false: " + ( false | false ) +
34             "\nfalse | true:  " + ( false | true ) +
35             "\ntrue | false:  " + ( true | false ) +
36             "\ntrue | true:   " + ( true | true ) );
37
38          // testing the logical exclusive OR operator (^)
39          Console.WriteLine( "\n\nLogical exclusive OR (^)"
40             "\nfalse ^ false: " + ( false ^ false ) +
41             "\nfalse ^ true:  " + ( false ^ true ) +
42             "\ntrue ^ false:  " + ( true ^ false ) +
43             "\ntrue ^ true:   " + ( true ^ true ) );
44
45          // testing the logical NOT operator (!)
46          Console.WriteLine( "\n\nLogical NOT (!)" +
47             "\n!false: " + ( !false ) +
48             "\n!true:  " + ( !true ) );
49       }
50    }
```

Outputs a truth table for the logical OR operator (||)

LogicalOperators.cs

If one is true the result is true

Outputs a truth table for the logical exclusive OR operator (||)

conditionals are the same

Outputs a truth table for the logical NOT operator (!)

Returns the opposite as the input

Program Output

```
Conditional AND (&&)
false && false: False
false && true:  False
true && false:  False
true && true:   True


Conditional OR (||)
false || false: False
false || true:  True
true || false:  True
true || true:   True
```

```
Logical AND (&)
false & false: False
false & true:  False
true & false:  False
true & true:   True


Logical OR (|)
false | false: False
false | true:  True
true | false:  True
true | true:   True


Logical exclusive OR (^)
false ^ false: False
false ^ true:  True
true ^ false:  True
true ^ true:   False


Logical NOT (!)
!false: True
!true:  False
```

LogicalOperators.cs Program Output

5.9 خلاصه ای از برنامه نویسی ساخت یافته

- ساختارهای کنترلی
  - فقط یک ورودی
  - فقط یک خروجی
  - ساختن بلوکهای برنامه نویسی
  - تو در تویی را اجزه می دهد
- باعث می شود کدها آراسته تر و آسان تر دنبال شود
- ساختارها با همدیگر تداخل ندارند
  - کلمه کلیدی **goto**

5.9 خلاصه ای از برنامه نویسی ساخت یافته

- ۳ شکل از کنترلهای لازم
  - راه های بسیاری برای اجرای این کنترل
  - Sequential (only 1 way)
    - برنامه نویسی رو به جلو
  - Selection (3 ways)
    - **`if`** selection (one choice)
    - **`if/else`** selection (two choices)
    - **`switch`** statement (multiple choices)
  - Repetition (4 ways)
    - **`while`** structure
    - **`do/while`** structure
    - **`for`** structure
    - **`foreach`** structure (chapter 7)

5.9 خلاصه ای از برنامه نویسی ساخت یافته

Operators	Associativity	Type
`()` `++ --`	left to right right to left	parentheses unary postfix
`++ -- + - ! (type)`	right to left	unary prefix
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`&`	left to right	logical AND
`^`	left to right	logical exclusive OR
`|`	left to right	logical inclusive OR
`&&`	left to right	conditional AND
`||`	left to right	conditional OR
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

Fig. 5.21 Precedence and associativity of the operators discussed so far.

5.9 خلاصه ای از برنامه نویسی ساخت یافته

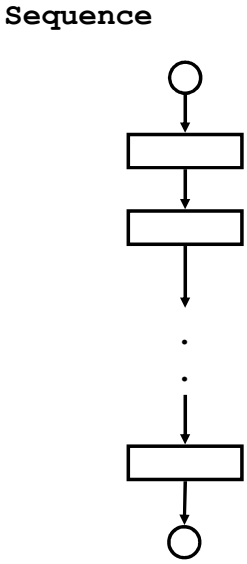


Fig. 5.22 C#'s single-entry/single-exit sequence, selection and repetition structures. (part 1)

5.9 خلاصه ای از برنامه نویسی ساخت یافته

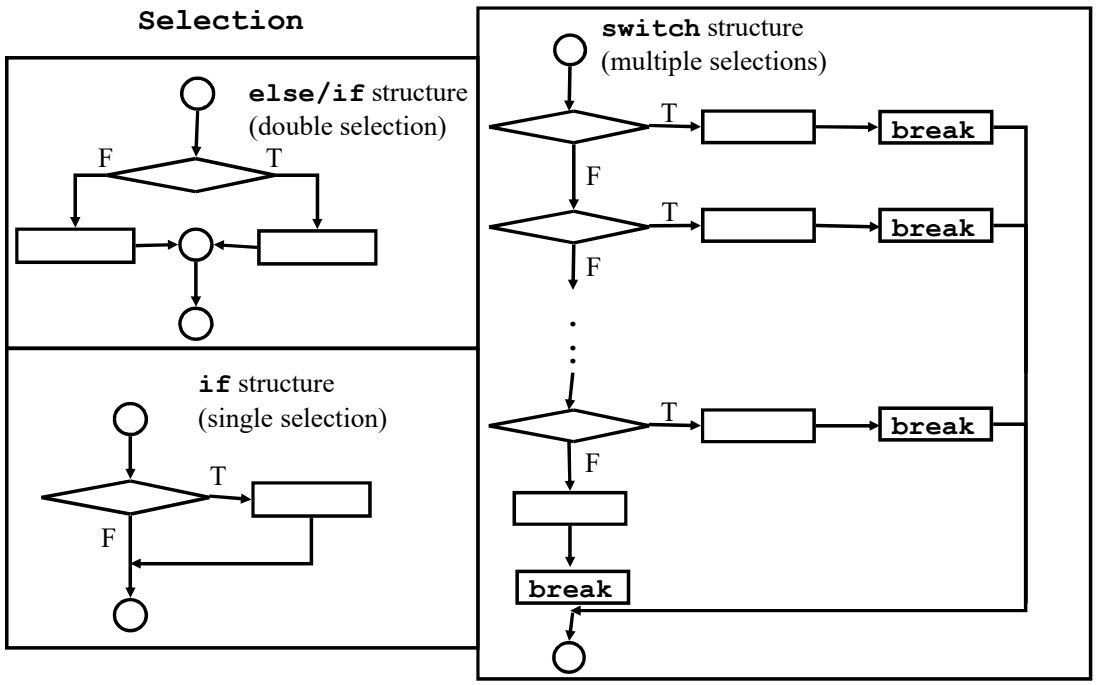


Fig. 5.22 C#'s single-entry/single-exit sequence, selection and repetition structures. (part 2)

5.9 خلاصه ای از برنامه نویسی ساخت یافته

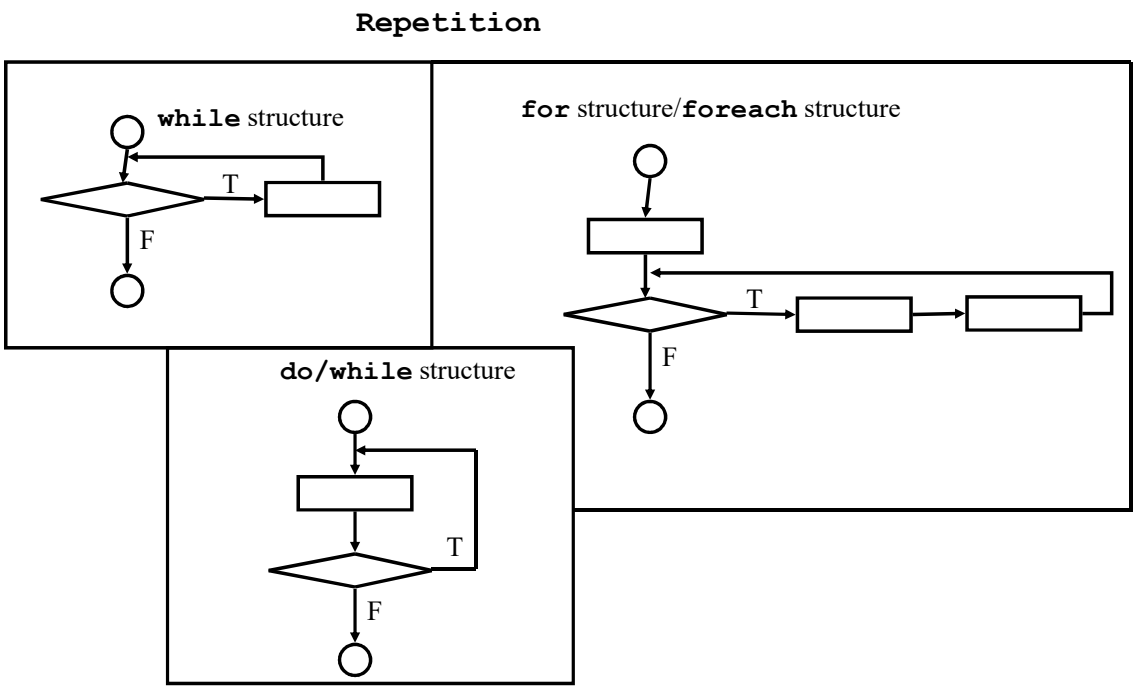


Fig. 5.22 C#'s single-entry/single-exit sequence, selection and repetition structures. (part 3)

5.9 خلاصه ای از برنامه نویسی ساخت یافته

Rules for Forming Structured Programs	
1)	Begin with the "simplest flowchart" (Fig. 5.24).
2)	Any rectangle (action) can be replaced by two rectangles (actions) in sequence.
3)	Any rectangle (action) can be replaced by any control structure (sequence, **`if`**, **`if/else`**, **`switch`**, **`while`**, **`do/while`**, **`for`** or **`foreach`**, as we will see in Chapter 8, Object-Oriented Programming).
4)	Rules 2 and 3 may be applied as often as you like and in any order.

Fig. 5.23 Rules for forming structured programs.

5.9 خلاصه ای از برنامه نویسی ساخت یافته

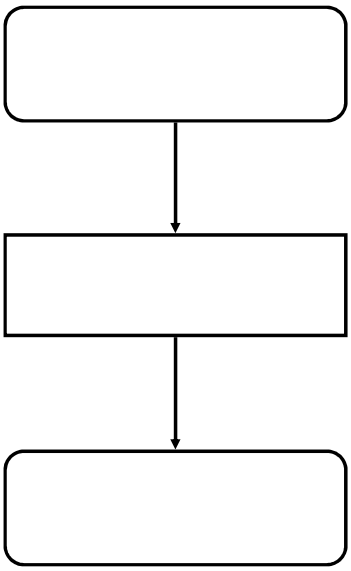


Fig. 5.24 Simplest flowchart.

5.9 خلاصه ای از برنامه نویسی ساخت یافته

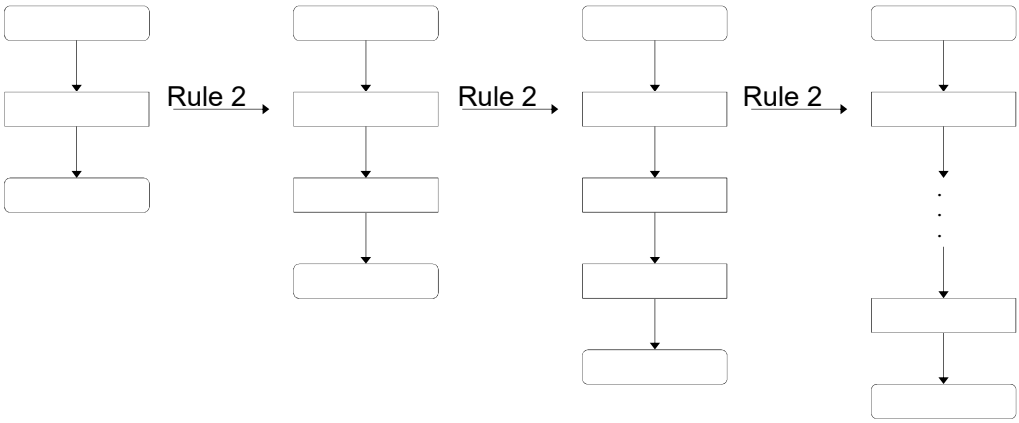


Fig. 5.25 Repeatedly applying rule 2 of Fig. 5.23 to the simplest flowchart.

5.9 خلاصه ای از برنامه نویسی ساخت یافته

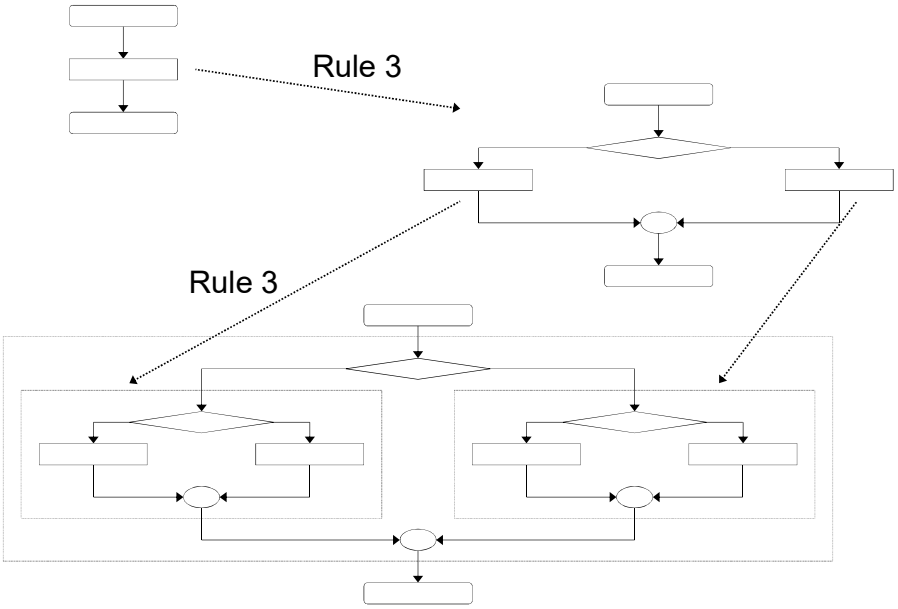


Fig. 5.26 Applying rule 3 of Fig. 5.23 to the simplest flowchart.

5.9 خلاصه ای از برنامه نویسی ساخت یافته

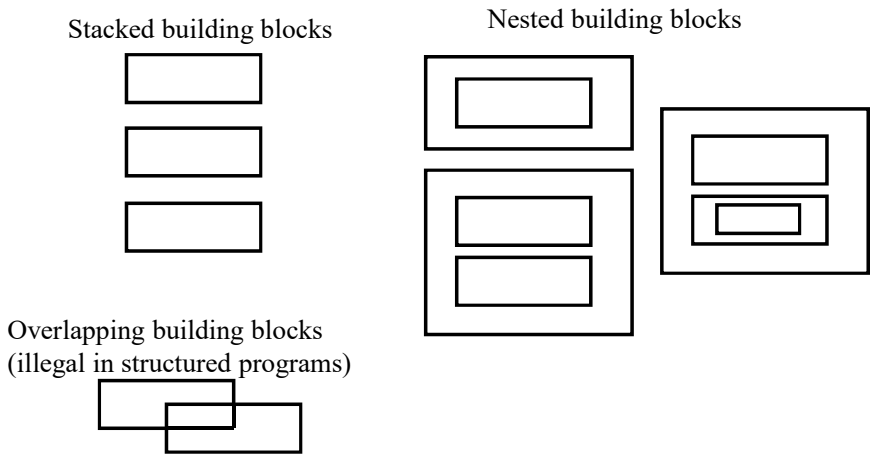


Fig. 5.27 Stacked, nested and overlapped building blocks.

5.9 خلاصه ای از برنامه نویسی ساخت یافته

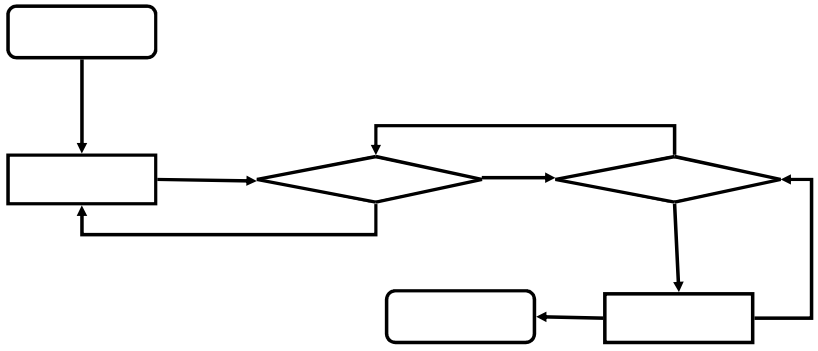


Fig. 5.28 Unstructured flowchart.

انواع داده ای در C#

Short Name	.NET Class	Type	Width	Range (bits)
byte	Byte	Unsigned integer	8	0 to 255
sbyte	SByte	Signed integer	8	-128 to 127
int	Int32	Signed integer	32	-2,147,483,648 to 2,147,483,647
uint	UInt32	Unsigned integer	32	0 to 4294967295
short	Int16	Signed integer	16	-32,768 to 32,767
ushort	UInt16	Unsigned integer	16	0 to 65535
long	Int64	Signed integer	64	-9223372036854775808 to 9223372036854775807
ulong	UInt64	Unsigned integer	64	0 to 18446744073709551615
float	Single	Single-precision floating point type	32	-3.402823e38 to 3.402823e38
double	Double	Double-precision floating point type	64	-1.79769313486232e308 to 1.79769313486232e308
char	Char	A single Unicode character	16	Unicode symbols used in text
bool	Boolean	Logical Boolean type	8	True or false
object	Object	Base type of all other types		
string	String	A sequence of characters		
decimal	Decimal	Precise fractional or integral type that can represent decimal numbers with 29 significant digits	128	±1.0 × 10e-28 to ±7.9 × 10e28

127

انواع داده ای در C#

- نوع داده :byteدر این نوع داده می توان از بازه ۰ تا ۲۵۵ را ذخیره کرد.
- نوع داده :sbyteدر این نوع داده می توان از بازه ۱۲۸- تا ۱۲۷ را ذخیره کرد.
- نوع داده :shortدر این نوع داده می توان از بازه ۳۲،۷۶۸- تا ۳۲،۷۶۷ را ذخیره کرد.
- نوع داده :ushortدر این نوع داده می توان از بازه ۰ تا ۶۵،۵۳۵ را ذخیره کرد.
- نوع داده :intدر این نوع داده می توان از بازه ۲،۱۴۷،۴۸۳،۶۴۸- تا ۲،۱۴۷،۴۸۳،۶۴۷ را ذخیره کرد.
- نوع داده :uintدر این نوع داده می توان از بازه ۰ تا ۴،۲۹۴،۹۶۷،۲۹۵ را ذخیره کرد.
- نوع داده :longدر این نوع داده می توان از بازه ۹،۲۲۳،۳۷۲،۰۳۶،۸۵۴،۷۷۵،۸۰۸- تا ۹،۲۲۳،۳۷۲،۰۳۶،۸۵۴،۷۷۵،۸۰۷ را ذخیره کرد.
- نوع داده :ulongدر این نوع داده می توان از بازه ۰ تا ۱۸،۴۴۶،۷۴۴،۰۷۳،۷۰۹،۵۵۱،۶۱۵ را ذخیره کرد.
- نوع داده :floatدر این نوع داده می توان از بازه ۳.۴۰۲۸۲۳ -e38تا ۳.۴۰۲۸۲۳ e38را ذخیره کرد.
- نوع داده :doubleدر این نوع داده می توان از بازه ۱.۷۹۷۶۹۳۱۳۴۸۶۲۳۲ -e308تا ۱.۷۹۷۶۹۳۱۳۴۸۶۲۳۲ e308را ذخیره کرد.
- نوع داده :decimalدر این نوع داده می توان از بازه ۷۹۲۲۸۱۶۲۵۱۴۲۶۴۳۳۷۵۹۳۵۴۳۹۵۰۳۳۵- تا ۷۹۲۲۸۱۶۲۵۱۴۲۶۴۳۳۷۵۹۳۵۴۳۹۵۰۳۳۵ را ذخیره کرد.

128

فصل ششم

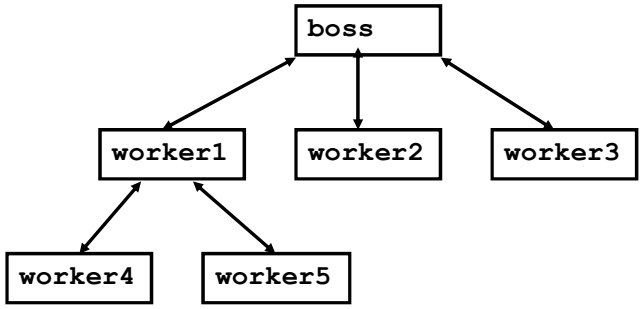
Methods

6.2 برنامه نویسی Method در C#

- ماژول ها
  - Class
  - Method
  - استفاده از کلاسها و روشها را بدون آگاهی از نحوه کار آنها را قادر می کند، چیزی که آنها باید انجام دهید.
- The .NET Framework Class Library (FCL)
  - به قابلیت استفاده مجدد کمک می کند.
  - **`Console`**
  - **`MessageBox`**

130

6.2 برنامه نویسی Method در C#



boss رئیس

worker کارگر

Fig. 6.1 Hierarchical boss method/worker method relationship.

131

6.3 متدهای کلاس Math

- کلاس **Math**
  - به کاربر برای انجام محاسبات ریاضی رایج اجازه می دهد
  - استفاده از متدها
    - *ClassName.MethodName*(*argument1*, *arument2*, …)
    - لیست متد ها در شکل ٦.٢ آمده
  - ثابتها
    - **Math.PI** = 3.1415926535…
    - **Math.E** = 2.7182818285…

6.3 `Math` Class Methods

Method	Description	Example
`Abs( x )`	absolute value of *x*	`Abs( 23.7 )` is 23.7 `Abs( 0 )` is 0 `Abs( -23.7 )` is 23.7
`Ceiling( x )`	rounds *x* to the smallest integer not less than *x*	`Ceiling( 9.2 )` is 10.0 `Ceiling( -9.8 )` is -9.0
`Cos( x )`	trigonometric cosine of *x* (*x* in radians)	`Cos( 0.0 )` is 1.0
`Exp( x )`	exponential method e^x	`Exp( 1.0 )` is approximately 2.7182818284590451 `Exp( 2.0 )` is approximately 7.3890560989306504
`Floor( x )`	rounds *x* to the largest integer not greater than *x*	`Floor( 9.2 )` is 9.0 `Floor( -9.8 )` is -10.0

`Log( x )`	natural logarithm of *x* (base *e*)	`Log( 2.7182818284590451 )`
		is approximately 1.0
		`Log( 7.3890560989306504 )`
		is approximately 2.0

`Max( x, y )`	larger value of *x* and *y* (also has versions for `float`, `int` and `long` values)	`Max( 2.3, 12.7 )` is 12.7 `Max( -2.3, -12.7 )` is -2.3
`Min( x, y )`	smaller value of *x* and *y* (also has versions for `float`, `int` and `long` values)	`Min( 2.3, 12.7 )` is 2.3 `Min( -2.3, -12.7 )` is -12.7
`Pow( x, y )`	*x* raised to power *y* (x^y)	`Pow( 2.0, 7.0 )` is 128.0 `Pow( 9.0, .5 )` is 3.0
`Sin( x )`	trigonometric sine of *x* (*x* in radians)	`Sin( 0.0 )` is 0.0
`Sqrt( x )`	square root of *x*	`Sqrt( 900.0 )` is 30.0 `Sqrt( 9.0 )` is 3.0
`Tan( x )`	trigonometric tangent of *x* (*x* in radians)	`Tan( 0.0 )` is 0.0

Fig. 6.2 Commonly used `Math` class methods.

6.4 متدها

- متغیرها
  - داخل متد تعریف شود = متغیر محلی
  - خارج از متد تعریف شود = متغیر سراسری
  - فقط متدهایی که آنها را تعریف می کنند می دانند وجود دارد
    - ارسال پارامتر برای ارتباط با سایر متدها
- دلایل استفاده
  - مقسیم و غلبه (Divide and conquer)
  - قابلیت استفاده مجدد
    - استفاده از کلاس ها و روش به عنوان بلوک های ساختمان برای آنهایی که جدید هستند
  - کاهش تکرار
    - روشها می توانند از هر نقطه برنامه فراخوانی شوند.

134

6.5 تعریف متد

- نوشتن یک متد دلخواه
  - هدر
    - ***ReturnType Properties*** *Name*(*Param1*, *Param2*, …)
  - بدنه
    - شامل کدهایی می باشد که باید متد انجام دهد
    - شامل مقدار بازگشتی خواهد بود اگر لازم باشد
- برای استفاده از فراخوانی در جاهای دیگر برنامه
  - ارسال پارامترها در صورت نیاز
- همه متد ها باید در داخل یک کلاس تعریف شوند

135

Subtract.cs

```
1    // Fig. 6.3: SquareInt.cs
2    // A programmer-defined Square method.
3
4    using System;                  // includes basic data types
5    using System.Drawing;          // for graphics capabilities
6    using System.Collections;
7    using System.ComponentMode
8    using System.Windows.Forms
9    using System.Data;             // for reading outside data
10
11   // form used to display results of squaring 10 numbers
12   public class SquareIntegers : System.Windows.Forms.Form
13   {
14      private System.ComponentModel.Container components = null;
15
16      // label containing results
17      private System.Windows.Forms.Label outputLabel;
18
19      public SquareIntegers()
20      {
21         // Required for Windows Form Designer support
22         InitializeComponent();
23
24         int result; // store result of call to method Square
25
```

Start of class SquareInteger. It implements **`System.Windows.Forms.Form`**

Start of the SquareIntegers method

This is the method's variables. They can only be used within the method.

```
26          // loop 10 times
27          for ( int counter = 1; counter <= 10; counter++ )
28          {
29             // calculate square of counter and store in result
30             result = Square( counter );
31
32             // append result to output string
33             outputLabel.Text += "The square of " + counter +
34                " is " + result + "\n";
35          }
36
37       } // end SquareIntegers
38
39       // Clean up any resources being used.
40       protected override void Dispose( bool disposing )
41       {
42          // Visual Studio .NET-generated code for method Dispose
43       }
44
45       // Required method for Designer support
46       private void InitializeComponent()
47       {
48          // Visual Studio .NET generated code
49          // for method InitializeComponent
50       }
51
```

The main body of the SquareIntegers method

A call to the Square method. The counter variable is passed to it for use. The return value is stored in result

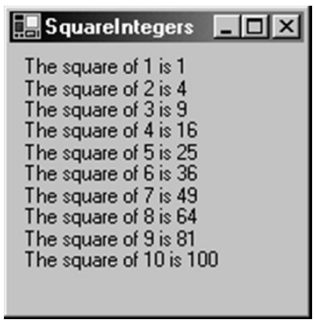
Subtract.cs

```
52      // The main entry point for the application.
53      [STAThread]
54      static void Main()
55      {
56         Application.Run( new SquareIntegers() );
57      }
58
59      // Square method definition
60      int Square( int y )
61      {
62         return y * y; // return square of y
63
64      } // end method Square
65
66   } // end of class SquareIntegers
```

Subtract.cs

The Square method. Receives one integer and returns an integer

The method returns the passed variable multiplied by itself



Program Output

MaximumValue.cs

```
1    // Fig. 6.4: MaximumValue.cs
2    // Finding the maximum of three doubles.
3
4    using System;
5
6    class MaximumValue
7    {
8       // main entry point for application
9       static void Main( string[] args )
10      {
11         // obtain user input and convert to double
12         Console.Write( "Enter first floating-point value: " );
13         double number1 = Double.Parse( Console.ReadLine() );
14
15         Console.Write( "Enter second floating-point value: " );
16         double number2 = Double.Parse( Console.ReadLine() );
17
18         Console.Write( "Enter third floating-point value: " );
19         double number3 = Double.Parse( Console.ReadLine() );
20
21         // call method Maximum to determine largest value
22         double max = Maximum( number1, number2, number3 );
23
24         // display maximum value
25         Console.WriteLine("\nmaximum is: " + max );
26
27      } // end method Main
```

The program gets three values from the user

The three values are then passed to the Maximum method for use

```
28
29      // Maximum method uses method Math.Max to help determine
30      // the maximum value
31      static double Maximum( double x, double y, double z )
32      {
33         return Math.Max( x, Math.Max( y, z ) );
34
35      } // end method Maximum
36
37   } // end class MaximumValue
```

MaximumValue.cs

The Maximum method receives 3 variables and returns the largest one

The use of Math.Max uses the Max method in class Math. The dot operator is used to call it.

```
Enter first floating-point value: 37.3
Enter second floating-point value: 99.32
Enter third floating-point value: 27.1928

maximum is: 99.32
```

Program Output

6.6 توسعه آرگومان

- تبدیل نوع ضمنی
  - شئ به طور ضمنی به نوع مورد نظر در صورت نیاز تبدیل می شود.
  - فقط زمانی انجام می شود که کامپایلر بداند هیچ داده ای از دست نخواهد رفت.
- تبدیل نوع صریح
  - شئ به صورت دستی تبدیل می شود
  - لازم است حتی اگر اطلاعات از دست برود
  - گسترش
    - Make an object that of a derived class and more complex
  - تنگ شدن
    - Make an object that of a base class and cause some data loss

141

6.6 توسعه آرگومان

Type	Can be Converted to Type(s)
bool	object
byte	decimal, double, float, int, uint, long, ulong, object, short or ushort
sbyte	decimal, double, float, int, long, object or short
char	decimal, double, float, int, uint, long, ulong, object or ushort
decimal	object
double	object
float	double or object
int	decimal, double, float, long or object
uint	decimal, double, float, long, ulong, or object
long	decimal, double, float or object
ulong	decimal, double, float or object
object	None
short	decimal, double, float, int, long or object
ushort	decimal, double, float, int, uint, long, ulong or object
string	object

Fig. 6.5 Allowed implicit conversions.

6.7 فضای نامی C#

- Namespace
  - یک گروه از کلاسها و توابع آنها
  - FCL یک گروه از فضاهای نامی می باشد
  - فضاهای نامی در یک فایل dll. که اسمبلی نامیده می شود قرار می گیرد.
  - یک لیست از فضاهای نامی FLC در شکل 6.6 آمده است
  - با کلمه کلیدی **`using`** می توان در برنامه وارد کرد.

143

6.7 C# Namespaces

Namespace	Description
`System`	Contains essential classes and data types (such as **`int`**, **`double`**, **`char`**, etc.). Implicitly referenced by all C# programs.
`System.Data`	Contains classes that form ADO .NET, used for database access and manipulation.
`System.Drawing`	Contains classes used for drawing and graphics.
`System.IO`	Contains classes for the input and output of data, such as with files.
`System.Threading`	Contains classes for multithreading, used to run multiple parts of a program simultaneously.
`System.Windows.Forms`	Contains classes used to create graphical user interfaces.
`System.Xml`	Contains classes used to process XML data.

Fig. 6.6 Namespaces in the Framework Class Library.

6.8 نوع های مقدار و نوعهای ارجاع

- نوع های مقدار
  - شامل داده از نوع مشخصی می باشند
  - توسط برنامه نویس ایجاد می شود
    - **`struct`** ها
    - **`enum`**erations (Chapter 8)
- نوعهای ارجاع
  - شامل آدرس نقطه ای در حافظه است که شامل داده می باشد
  - توسط برنامه نویس ایجاد می شود
    - Classes (Chapter 8)
    - Interfaces (Chapter 8)
    - Delegates (Chapter 9)
- All values are 32bit allowing cross-platform use

145

6.9 ارسال آرگومان: فراخوانی با مقدار در مقابل فراخوانی با ارجاع

- ارسال با مقدار
  - یک کپی از شئ به تابع ارسال می شود
  - هنگام برگشت، همیشه با مقدار برگشت داده می شود.
  - Set by value by default
- ارسال با ارجاع
  - به تابع نقطه ارجاع واقعی ارسال می شود
    - موجب می شود متغیر از طریق برنامه تغییر یابد.
  - هنگام برگشت، همیشه با ارجاع برگشت داده می شود.
  - کلمه کلیدی **`ref`** مشخص کننده با ارجاع می باشد
  - کلمه کلیدی **out** به معنی می باشد که تابع فراخوانی شده آنرا مقدار دهی اولیه خواهد کرد.

146

RefOutTest.cs

```
1    // Fig. 6.8: RefOutTest.cs
2    // Demonstrating ref and out parameters.
3
4    using System;
5    using System.Windows.Forms;
6
7    class RefOutTest
8    {
9       // x is passed as a ref int (original va
10      static void SquareRef( ref int x )
11      {
12         x = x * x;
13      }
14
15      // original
16      static void
17      {
18         x = 6;
19         x = x * x;
20      }
21
22      // x is passed by value (original value not changed)
23      static void Square( int x )
24      {
25         x = x * x;
26      }
27
28      static void Main( string[] args )
29      {
30         // create a new integer value, set it to 5
31         int y = 5;
32         int z;   // declare z, but do not initialize it
33
```

When passing a value by reference the value will be altered in the rest of the program as well

Since the methods are **`void`** they do not need a return value.

Since x is passed as **`out`** the variable can then be initialed in the method

Since not specified, this value is defaulted to being passed by value. The value of x will not be changed elsewhere in the program because a duplicate of the variable is created.

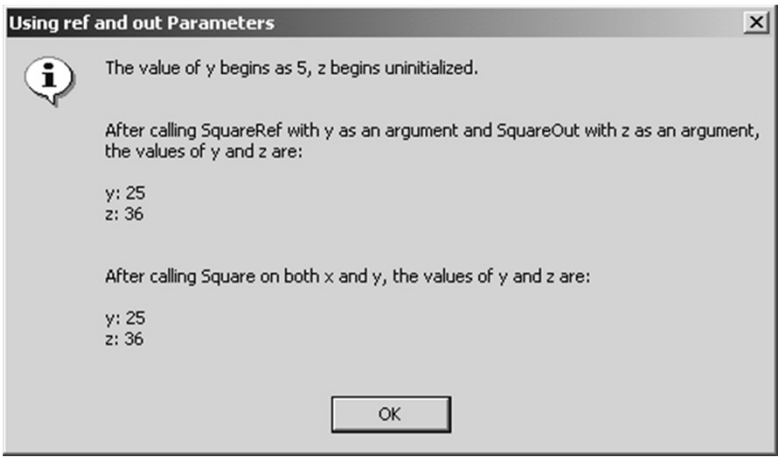
RefOutTest.cs

```
34          // display original values of y and z
35          string output1 = "The value of y begins as "
36             + y + ", z begins uninitialized.\n\n\n";
37
38          // values of y and z are passed by valu
39          RefOutTest.SquareRef( ref y );
40          RefOutTest.SquareOut( out z );
41
42          // display values of y and z after modified by methods
43          // SquareRef and SquareOut
44          string output2 = "After calling SquareRef with y as an " +
45             "argument and SquareOut with z as an argument,\n" +
46             "the values of y and z are:\n\n" +
47             "y: " + y + "\nz: " + z + "\n\n\n";
48
49          // values of y and z are passed by value
50          RefOutTest.Square( y );
51          RefOutTest.Square( z );
52
53          // values of y and z will be same as before because Square
54          // did not modify variables directly
55          string output3 = "After calling Square on both x and y, " +
56             "the values of y and z are:\n\n" +
57             "y: " + y + "\nz: " + z + "\n\n";
58
59          MessageBox.Show( output1 + output2 + output3,
60             "Using ref and out Parameters", MessageBoxButtons.OK,
61             MessageBoxIcon.Information );
62
63       } // end method Main
64
65    } // end class RefOutTest
```

The calling of the SquareRef and SquareOut methods

The calling of the SquareRef and SquareOut methods by passing the variables by value

RefOutTest.cs
Program Output



6.10 تولید اعداد تصادفی

- Class **Random**
  - Within namespace **System**
  - Truly random
    - The numbers are generated using an equations with a seed
      - The seed is usually the exact time of day
- **randomObject.Next**()
  - Returns a number from 0 to **Int32.MaxValue**
    - Int32.MaxValue = 2,147,483,647
- **randomObject.Next**(*x*)
  - Returns a value from 0 up to but not including *x*
- **randomObject.Next**(*x*, *y*)
  - Returns a number between *x* and up to but not including *y*

150

RandomInt.cs

```
1    // Fig. 6.9: RandomInt.cs
2    // Random integers.
3
4    using System;
5    using System.Windows.Forms;
6
7    // calculates and displays 20 random integers
8    class RandomInt
9    {
10      // main entry poi
11      static void Main(
12      {
13         int value;
14         string output = "";
15
16         Random randomInteger = new Random();
17
18         // loop 20 times
19         for ( int i = 1; i <= 20; i++ )
20         {
21            // pick random integer between 1 and 6
22            value = randomInteger.Next( 1, 7 );
23            output += value + " "; // append value to output
24
25            // if counter divisible by 5, append newline
26            if ( i % 5 == 0 )
27               output += "\n";
28
29         } // end for structure
30
```

Creates a new Random object

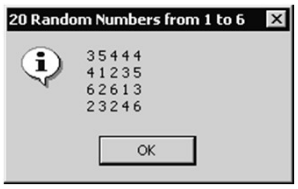
Will set value to a random number from1 up to but not including 7

Format the output to only have 5 numbers per line

```
31          MessageBox.Show( output, "20 Random Numbers from 1 to 6",
32             MessageBoxButtons.OK, MessageBoxIcon.Information );
33
34       } // end Main
35
36    } // end class RandomInt
```

Display the output in a message box

RandomInt.cs



Program Output

RollDie.cs

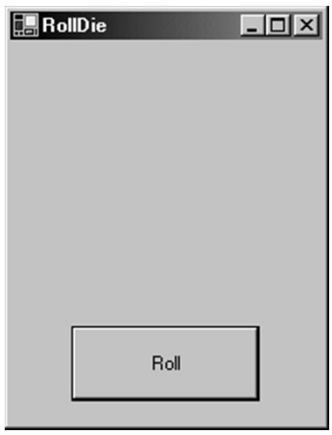
```
1    // Fig. 6.10: RollDie.cs
2    // Rolling 12 dice.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10   using System.IO;   // enables reading data from files
11
12   // form simulates the rolling of 12 dice,
13   // and displays them
14   public class RollDie : System.Windows.Forms.Form
15   {
16      private System.ComponentModel.Container components = null;
17
18      private System.Windows.Forms.Button rollButton;
19
20      private System.Windows.Forms.Label dieLabel2;
21      private System.Windows.Forms.Label dieLabel1;
22      private System.Windows.Forms.Label dieLabel3;
23      private System.Windows.Forms.Label dieLabel4;
24
25      private Random randomNumber = new Random();
26
27      public RollDie()
28      {
29         InitializeComponent();
30      }
31
32      // Visual Studio .NET-generated code
33
```

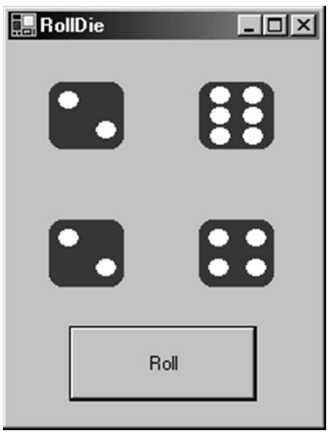
RollDie.cs

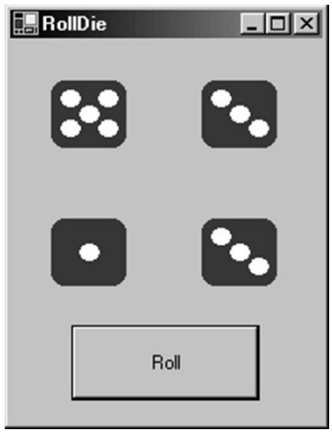
```
34      // method called when rollButton clicked,
35      // passes labels to another method
36      protected void rollButton_Click(
37         object sender, System.EventArgs e )
38      {
39         // pass the labels to a method that will
40         // randomly assign a face to each die
41         DisplayDie( dieLabel1 );
42         DisplayDie( dieLabel2 );
43         DisplayDie( dieLabel3 );
44         DisplayDie( dieLabel4 );
45
46      } // end rollButton_Click
47
48      // determines image to be displayed by current die
49      public void DisplayDie( Label dieLabel )
50      {
51         int face = 1 + randomNumber.Next( 6 );
52
53         // displays image specified by filename
54         dieLabel.Image = Image.FromFile(
55            Directory.GetCurrentDirectory() +
56            "\\images\\die" + face +".gif" );
57      }
58
59      // main entry point for application
60      [STAThread]
61      static void Main()
62      {
63         Application.Run( new RollDie() );
64      }
65
66   } // end class RollDie
```

Pass the labels to be assigned data

Will return a random integer from 0 up to 6







RollDie.cs
Program Output

RollDie2.cs

```
1    // Fig. 6.11: RollDie2.cs
2    // Rolling 12 dice with frequency chart.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10   using System.IO;
11
12   // displays the different dice and frequency information
13   public class RollDie2 : System.Windows.Forms.Form
14   {
15      private System.ComponentModel.Container components = null;
16
17      private System.Windows.Forms.Button rollButton;
18
19      private System.Windows.Forms.RichTextBox displayTextBox;
20
21      private System.Windows.Forms.Label dieLabel1;
22      private System.Windows.Forms.Label dieLabel2;
23      private System.Windows.Forms.Label dieLabel3;
24      private System.Windows.Forms.Label dieLabel4;
25      private System.Windows.Forms.Label dieLabel5;
26      private System.Windows.Forms.Label dieLabel6;
27      private System.Windows.Forms.Label dieLabel7;
28      private System.Windows.Forms.Label dieLabel8;
29      private System.Windows.Forms.Label dieLabel9;
30      private System.Windows.Forms.Label dieLabel10;
31      private System.Windows.Forms.Label dieLabel11;
32      private System.Windows.Forms.Label dieLabel12;
33
34      private Random randomNumber = new Random();
```

RollDie2.cs

```
35
36      private int ones, twos, threes, fours, fives, sixes;
37
38      public RollDie2()
39      {
40         InitializeComponent();
41         ones = twos = threes = fours = fives = sixes = 0;
42      }
43
44      // Visual Studio .NET-generated code
45
46      // simulates roll by calling DisplayDie for
47      // each label and displaying the results
48      protected void rollButton_Click(
49         object sender, System.EventArgs e )
50      {
51         // pass the labels to a method that will
52         // randomly assign a face to each die
53         DisplayDie( dieLabel1 );
54         DisplayDie( dieLabel2 );
55         DisplayDie( dieLabel3 );
56         DisplayDie( dieLabel4 );
57         DisplayDie( dieLabel5 );
58         DisplayDie( dieLabel6 );
59         DisplayDie( dieLabel7 );
60         DisplayDie( dieLabel8 );
61         DisplayDie( dieLabel9 );
62         DisplayDie( dieLabel10 );
63         DisplayDie( dieLabel11 );
64         DisplayDie( dieLabel12 );
65
66         double total = ones + twos + threes + fours + fives + sixes;
67
```

Sets all of the variables to 0

Pass the label to be assigned a random number

```
68          // display the current frequency values
69          displayTextBox.Text = "Face\t\tFrequency\tPercent\n1\t\t" +
70             ones + "\t\t" +
71             String.Format( "{0:F2}", ones / total * 100 ) +
72             "%\n2\t\t" + twos + "\t\t" +
73             String.Format( "{0:F2}", twos / total * 100 ) +
74             "%\n3\t\t" + threes + "\t\t" +
75             String.Format( "{0:F2}", threes / total *
76             "%\n4\t\t" + fours + "\t\t" +
77             String.Format( "{0:F2}", fours / total *
78             "%\n5\t\t" + fives + "\t\t" +
79             String.Format( "{0:F2}", fives / total * 100 ) +
80             "%\n6\t\t" + sixes + "\t\t" +
81             String.Format( "{0:F2}", sixes / total * 100 ) + "%";
82
83       } // end rollButton_Click
84
85       // display the current die, and modify frequency values
86       public void DisplayDie( Label dieLabel )
87       {
88          int face = 1 + randomNumber.Next( 6 );
89
90          dieLabel.Image = Image.FromFile(
91             Directory.GetCurrentDirectory() +
92             "\\images\\die" + face + ".gif" );
93
```

RollDie2.cs

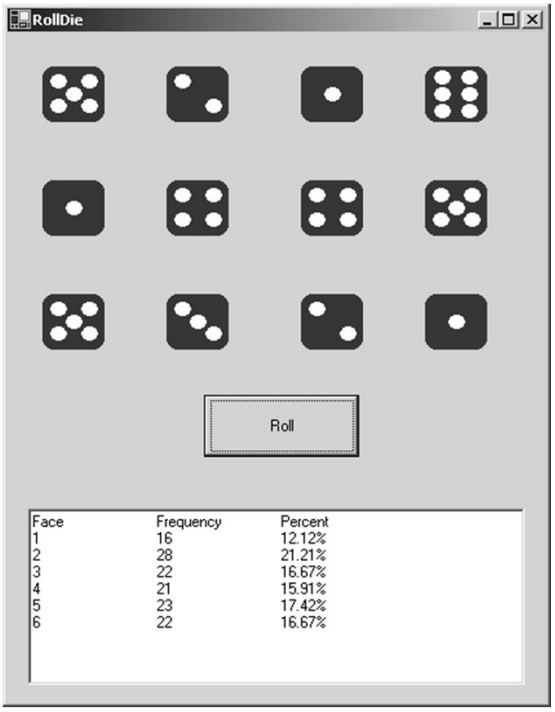
Displays to the user the amount of times each dice number has shown up

Assign a random face to the label based on the number generated

```
94          // add one to frequency of current face
95          switch ( face )
96          {
97             case 1: ones++;
98                break;
99
100            case 2: twos++;
101               break;
102
103            case 3: threes++;
104               break;
105
106            case 4: fours++;
107               break;
108
109            case 5: fives++;
110               break;
111
112            case 6: sixes++;
113               break;
114
115         } // end switch
116
117      } // end DisplayDie
118
119      // The main entry point for the application.
120      [STAThread]
121      static void Main()
122      {
123         Application.Run( new RollDie2() );
124      }
125
126   } // end of class RollDie2
```

A **`switch`** statement is used to keep track of number of each die rolled

RollDie2.cs



RollDie2.cs
Program Output

6.11 مثال: بازی شانس

- GUI controls
  - A **`GroupBox`**
    - Holds other controls
    - Manages/organizes them
  - A **`PictureBox`**
    - Used to display a picture on the form

161

CrapsGame.cs

```
1    // Fig. 6.12: CrapsGame.cs
2    // Craps Game
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10   using System.IO;
11
12   public class CrapsGame : System.Windows.Forms.Form
13   {
14      private System.ComponentModel.Container components = null;
15
16      private System.Windows.Forms.PictureBox imgPointDie2;
17      private System.Windows.Forms.PictureBox imgDie2;
18      private System.Windows.Forms.PictureBox imgDie1;
19
20      private System.Windows.Forms.Label lblStatus;
21
22      private System.Windows.Forms.Button rollButton;
23      private System.Windows.Forms.Button playButton;
24
25      private System.Windows.Forms.PictureBox imgPointDie1;
26
27      private System.Windows.Forms.GroupBox fraPoint;
28
29      // declare other variables
30      int myPoint;
31      int myDie1;
32      int myDie2;
33
```

```
34      public enum DiceNames
35      {
36         SNAKE_EYES = 2,
37         TREY = 3,
38         CRAPS = 7,
39         YO_LEVEN = 11,
40         BOX_CARS = 12,
41      }
42
43      public CrapsGame()
44      {
45         InitializeComponent();
46      }
47
48      // Visual Studio .NET-generated code
49
50      // simulate next roll and result of that roll
51      protected void rollButton_Click(
52         object sender, System.EventArgs e )
53      {
54         int sum;
55         sum = rollDice();
56
57         if ( sum == myPoint )
58         {
59            lblStatus.Text = "You Win!!!";
60            rollButton.Enabled = false;
61            playButton.Enabled = true;
62         }
```

Creates an enumeration of the constant values in craps

When the second rolled sum equals the first rolled sum the player wins

CrapsGame.cs

```
63          else
64             if ( sum == ( int )DiceNames.CRAPS )
65          {
66             lblStatus.Text = "Sorry. You lose.";
67             rollButton.Enabled = false;
68             playButton.Enabled = true;
69          }
70
71       } // end rollButton_Click
72
73       // simulate first roll and result of that roll
74       protected void playButton_Click(
75          object sender, System.EventArgs e )
76       {
77          int sum;
78          myPoint = 0;
79          fraPoint.Text = "Point";
80          lblStatus.Text = "";
81          imgPointDie1.Image = null;
82          imgPointDie2.Image = null;
83
84          sum = rollDice();
85
```

CrapsGame.cs

If the second roll equals CRAPS (7), then the player loses

CrapsGame.cs

```
86          switch ( sum )
87          {
88             case ( int )DiceNames.CRAPS:
89             case ( int )DiceNames.YO_LEVEN:
90                rollButton.Enabled = false;  // disable Roll button
91                lblStatus.Text = "You Win!!!";
92                break;
93             case ( int )DiceNames.SNAKE_EYES:
94             case ( int )DiceNames.TREY:
95             case ( int )DiceNames.BOX_CARS:
96                rollButton.Enabled = false;
97                lblStatus.Text = "Sorry. You lose.";
98                break;
99             default:
100               myPoint = sum;
101               fraPoint.Text = "Point is " + sum;
102               lblStatus.Text = "Roll Again";
103               displayDie( imgPointDie1, myDie1 );
104               displayDie( imgPointDie2, myDie2 );
105               playButton.Enabled = false;
106               rollButton.Enabled = true;
107               break;
108
109         } // end switch
110
111      } // end playButton_Click
112
113      private void displayDie( PictureBox imgDie, int face )
114      {
115         imgDie.Image = Image.FromFile(
116            Directory.GetCurrentDirectory() +
117            "\\images\\die" + face + ".gif" );
118      }
119
```

If on the first roll the players gets a 7 or an 11 they win

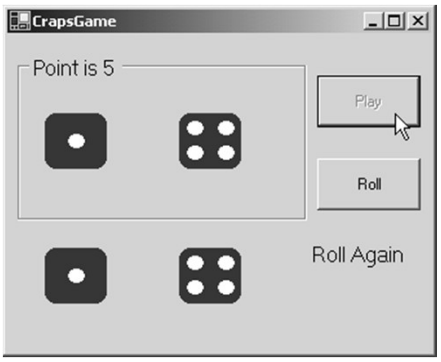
If the first roll is a 2, 3 or 12, the player loses.

Any other number allows the player to roll again

CrapsGame.cs

```
120     // simulates the rolling of two dice
121     private int rollDice()
122     {
123        int die1, die2, dieSum;
124        Random randomNumber = new Random();
125
126        die1 = randomNumber.Next( 1, 7 );
127        die2 = randomNumber.Next( 1, 7 );
128
129        displayDie( imgDie1, die1 );
130        displayDie( imgDie2, die2 );
131
132        myDie1 = die1;
133        myDie2 = die2;
134        dieSum = die1 + die2;
135        return dieSum;
136
137     } // end rollDice
138
139     // main entry point for application
140     [STAThread]
141     static void Main()
142     {
143        Application.Run(new CrapsGame());
144     }
145
146  } // end of class CrapsGame
```

Generates two random numbers, one for each die



CrapsGame.cs
Program Output

6.12 مدت زمان شناسه ها

- مدت زمان (Duration)
  - زمانی که یک شناسه در حافظه باقی می ماند.
- حوزه (Scope)
  - بخشی از برنامه که شی شابل دسترس می باشد.
- متغیرهای محلی
  - زمانی که تعریف می شود ایجاد خواهد شد.
  - هنگام خروج از بلوک از بین می رود.
  - Not initialized
    - Most variables are set to 0
    - All **bool** variables are set to **false**
    - All reference variables are set to **null**

168

6.13 قوانین دامنه

- دامنه (Scope)
  - بخشی از برنامه که در آن یک متغیر قابل دسترس می باشد.
  - Class scope
    - از زمانی که داخل کلاس ایجاد می شود.
    - تا پایان کلاس (})
    - برای تمام متد های آن کلاس سراسری می باشد.
      - تغییر مستقیم
    - تکرار کردن نام باعث می شود تا قبلی تا پایان دامنه پنهان شوند.
  - Block scope
    - از زمان ایجاد
    - تا پایان بلوک (})
    - فقط داخل همان بلوک استفاده می شود
      - باید ارسال شده و بصورت غیر مستقیم تغییر یابد.
    - نمی تواند نام متغیر تکرار شود

169

Scoping.cs

```
1    // Fig. 6.13: Scoping.cs
2    // A Scoping example.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10
11   public class Scoping : System.Windows.Forms.Form
12   {
13      private System.ComponentModel.Container components = null;
14      private System.Windows.Forms.Label outputLabel;
15
16      public int x = 1;
17
18      public Scoping()
19      {
20         InitializeComponent();
21
22         int x = 5;      // variable local
23
24         outputLabel.Text = outputLabel.Text +
25            "local x in method Scoping is " + x;
26
27         MethodA();    // MethodA has automatic local x;
28         MethodB();    // MethodB uses instance variable x
29         MethodA();    // MethodA creates new automatic local x
30         MethodB();    // instance variable x retains its value
31
32         outputLabel.Text = outputLabel.Text +
33            "\n\nlocal x in method Scoping is " + x;
34      }
```

This variable has class scope and can be used by any method in the class

This variable is local only to Scoping. It hides the value of the global variable

Will output the value of 5

Remains 5 despite changes to global version of x

Scoping.cs

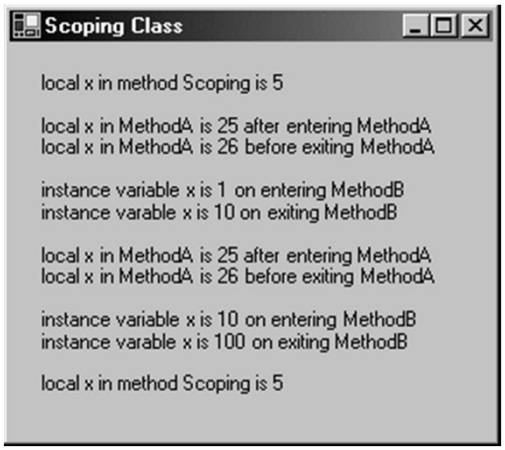
```
35
36      // Visual Studio .NET-generated code
37
38      public void MethodA()
39      {
40         int x = 25;     // initialized each tim
41
42         outputLabel.Text = outputLabel.Text +
43            "\n\nlocal x in MethodA is " + x +
44            " after entering MethodA";
45         ++x;
46         outputLabel.Text = outputLabel.Text +
47            "\nlocal x in MethodA is " + x +
48            " before exiting MethodA";
49      }
50
51      public void MethodB()
52      {
53         outputLabel.Text = outputLabel.Text +
54            "\n\ninstance variable x is " + x +
55            " on entering MethodB";
56         x *= 10;
57         outputLabel.Text = outputLabel.Text +
58            "\ninstance varable x is " + x +
59            " on exiting MethodB";
60      }
61
62      // main entry point for application
63      [STAThread]
64      static void Main()
65      {
66         Application.Run( new Scoping() );
67      }
68
69   } // end of class Scoping
```

Uses a new x variable that hides the value of the global x

Uses the global version of x (1)

Will permanently change the value of x globally

Scoping.cs
Program Output

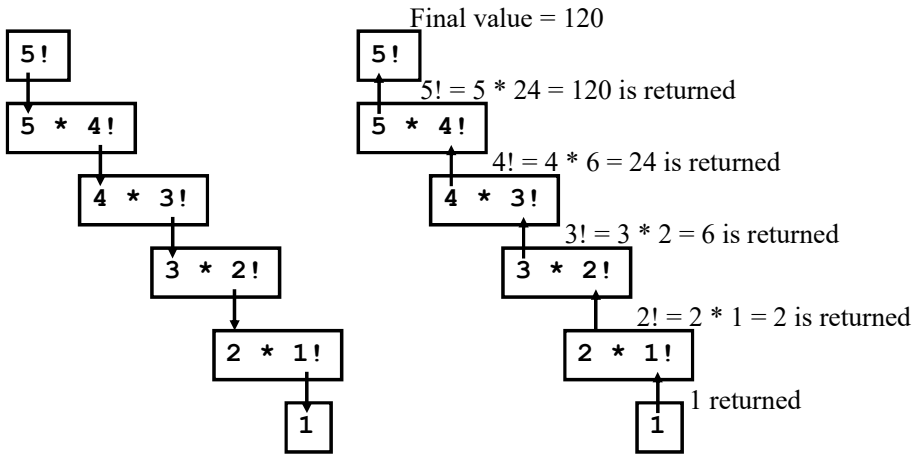


6.14 بازگشت

- متدهای بازگشتی
  - متد هایی که می توانند خودشان را فراخوانی نمایند
    - مستقیم
    - غیر مستقیم
      - متدی که آنرا فراخوانی کرده است فراخوانی می کند
  - به طور مستمر مسئله را به اشکال ساده تر می شکند.
  - باید برای پایان دادن به بازگشت همگرا باشد.
  - هر فراخوانی متد باز می ماند (ناتمام).
    - با پایان هر فراخوانی سپس خودش را پایان می دهد.

173

6.14 Recursion



(a) Procession of recursive calls. (b) Values returned from each recursive call.

Fig. 6.14 Recursive evaluation of 5!.

FactorialTest.cs

```
1    // Fig. 6.15: FactorialTest.cs
2    // Recursive Factorial method.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10
11   public class FactorialTest : System.Windows.Forms.Form
12   {
13      private System.ComponentModel.Container components = null;
14
15      private System.Windows.Forms.Label outputLabel;
16
17      public FactorialTest()
18      {
19         InitializeComponent();
20
21         for ( long i = 0; i <= 10; i++ )
22            outputLabel.Text += i + "! = " +
23               Factorial( i ) + "\n";
24      }
25
```

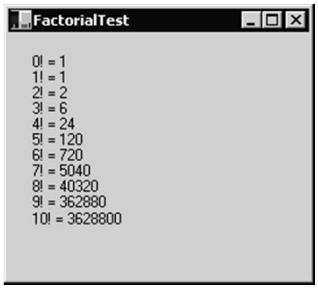
```
26      // Visual Studio .NET-generated code
27
28      public long Factorial( long number )
29      {
30         if ( number <= 1 )      // base case
31            return 1;
32
33         else
34            return number * Factorial( number - 1 );
35      }
36
37      [STAThread]
38      static void Main()
39      {
40         Application.Run( new FactorialTest());
41      }
42
43   } // end of class FactorialTest
```

The Factorial method calls itself (recursion)

The recursion ends when the value is less than or equal to 1

FactorialTest.cs

Program Output



6.15 Example Using Recursion: The Fibonacci Sequence

- Fibonacci Sequence
  - F(0) = 0
  - F(1) = 1
  - F(n) = F(n - 1) + F(n - 2)
  - Recursion is used to evaluate F(n)
- Complexity theory
  - How hard computers need to work to perform algorithms

FibonacciTest.cs

```
1    // Fig. 6.16: FibonacciTest.cs
2    // Recursive fibonacci method.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10
11   public class FibonacciTest : System.Windows.Forms.Form
12   {
13      private System.ComponentModel.Container components = null;
14
15      private System.Windows.Forms.Button calculateButton;
16
17      private System.Windows.Forms.TextBox inputTextBox;
18
19      private System.Windows.Forms.Label displayLabel;
20      private System.Windows.Forms.Label promptLabel;
21
22      public FibonacciTest()
23      {
24         InitializeComponent();
25      }
26
27      // Visual Studio .NET-generated code
28
```

FibonacciTest.cs

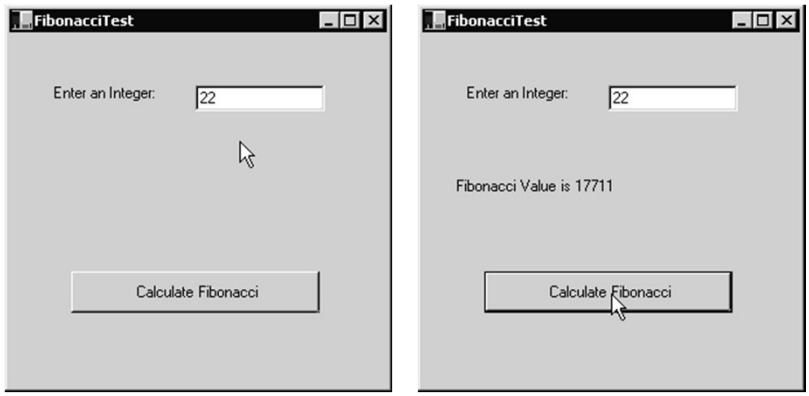
```
29      // call Fibonacci and display results
30      protected void calculateButton_Click(
31         object sender, System.EventArgs e )
32      {
33         string numberString = ( inputTextBox.Text );
34         int number = System.Convert.ToInt32( numberString );
35         int fibonacciNumber = Fibonacci( number );
36         displayLabel.Text = "Fibonacci Value is " + fibonacciNumber;
37      }
38
39      // calculates Fibonacci number
40      public int Fibonacci( int number )
41      {
42         if ( number == 0 || number == 1 )
43            return number;
44         else
45            return Fibonacci( number - 1 ) + Fibonacci( number - 2 );
46      }
47
48      [STATh
49      static
50      {
51         Application.Run( new FibonacciTest() );
52      }
53
54   } // end of class FibonacciTest
```

The number uses the Fibonacci method to get its result

The recursion ends when the number is 0 or 1

Calls itself twice, to get the result of the the two previous numbers

FibonacciTest.cs
Program Output



6.15 Example Using Recursion: The Fibonacci Sequence

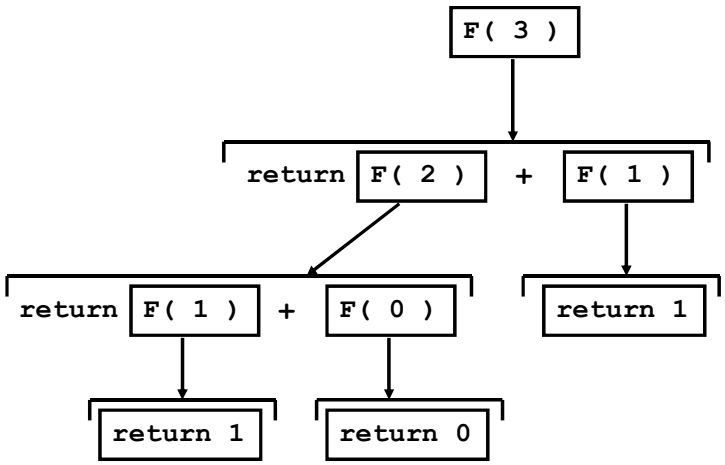


Fig. 6.17 *Set of recursive calls to method* ***`Fibonacci`*** *(abbreviated as* ***`F`****).*

6.16 Recursion vs. Iteration

- Iteration
  - Uses repetition structures
    - **`while`**, **`do/while`**, **`for`**, **`foreach`**
  - Continues until counter fails repetition case
- Recursion
  - Uses selection structures
    - **`if`**, **`if/else`**, **`switch`**
  - Repetition through method calls
  - Continues until a base case is reached
  - Creates a duplicate of the variables
    - Can consume memory and processor speed

6.17 Method Overloading

- Methods with the same name
  - Can have the same name but need different arguments
    - Variables passed must be different
      - Either in type received or order sent
  - Usually perform the same task
    - On different data types

MethodOverload.cs

```
1    // Fig. 6.18: MethodOverload.cs
2    // Using overloaded methods.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10
11   public class MethodOverload : System.Windows.Forms.Form
12   {
13      private System.ComponentModel.Container components = null;
14
15      private System.Windows.Forms.Label outputLabel;
16
17      public MethodOverload()
18      {
19         InitializeComponent();
20
21         // call both versions of Square
22         outputLabel.Text =
23            "The square of integer 7 is " + Square( 7 ) +
24            "\nThe square of double 7.5 is " + Square ( 7.5 );
25      }
26
27      // Visual Studio .NET-generated code
28
```

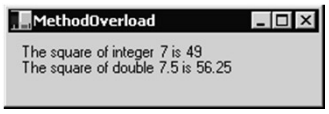
Two versions of the square method are called

```
29      // first version, takes one integer
30      public int Square ( int x )
31      {
32         return x * x;
33      }
34
35      // second version, takes one double
36      public double Square ( double y )
37      {
38         return y * y;
39      }
40
41      [STAThread]
42      static void Main()
43      {
44         Application.Run( new MethodOverload() );
45      }
46
47   } // end of class MethodOverload
```

One method takes an **`int`** as parameters

The other version of the method uses a **`double`** instead of an integer

MethodOverload.cs



Program Output

MethodOverload2.cs

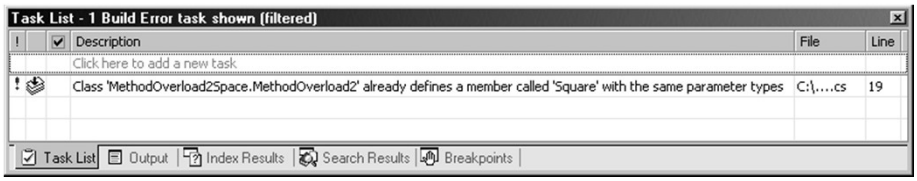
```
1    // Fig. 6.19: MethodOverload2.cs
2    // Overloaded methods with identical signatures and
3    // different return types.
4
5    using System;
6
7    class MethodOverload2
8    {
9       public int Square( double x )
10      {
11         return x * x;
12      }
13
14      // second Square method takes same number,
15      // order and type of arguments, error
16      public double Square( double y )
17      {
18         return y * y;
19      }
20
21      // main entry point for application
22      static void Main()
23      {
24         int squareValue = 2;
25         Square( squareValue );
26      }
27
28   } // end of class MethodOverload2
```

This method returns an integer

This method returns a double number

Since the compiler cannot tell which method to use based on passed values an error is generated

Program Output



فصل هفتم

آرایه ها

187

7.1 مقدمه

- Data structures
  - شامل انواع داده ای با نوع یکسان می باشد
  - ایستا: در هوان سایز خود تا پایان باقی می ماند

7.2 آرایه ها

- یک گروه از مکان های حافظه به هم پیوسته.
  - نام یکسان
  - نوع یکسان
- به عنصر خاصی از آرایه با *شماره موقعیت (اندیس)* می توان دسترسی پیدا کرد.
- می توانید به هر عنصر با دادن نام آرایه به دنبال آن شماره موقعیت (اندیس) از این عنصر داخل براکت مراجعه کنید ([])
- عنصر اول عنصر صفر است
  - First element of array **c** is **c[0]**

189

7.2 آرایه ها

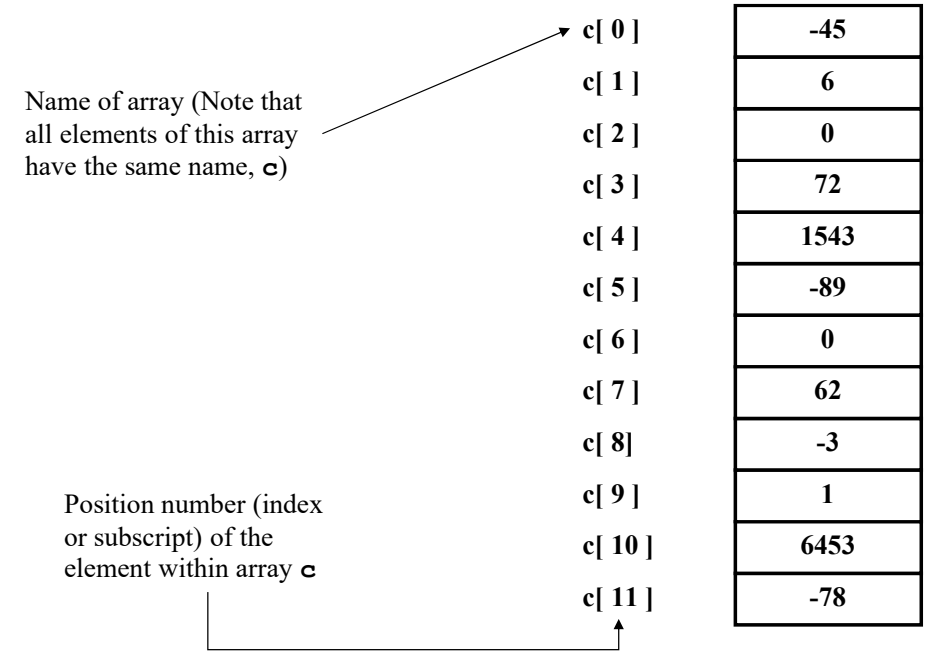


Fig. 7.1 A 12-element array.

7.2 Arrays

Operators	Associativity	Type
`()  []  .  ++  --`	left to right	highest (unary postfix)
`++  --  +  -  !  (type)`	right to left	unary (unary prefix)
`*  /  %`	left to right	multiplicative
`+  -`	left to right	additive
`<  <=  >  >=`	left to right	relational
`==  !=`	left to right	equality
`&`	left to right	boolean logical AND
`^`	left to right	boolean logical exclusive OR
`|`	left to right	boolean logical inclusive OR
`&&`	left to right	logical AND
`||`	left to right	logical OR
`?:`	right to left	conditional
`=  +=  -=  *=  /=  %=`	right to left	assignment

Fig. 7.2 Precedence and associativity of the operators discussed so far.

191

7.3 اعلان و اختصاص آرایه

- برنامه نویس نوع عناصر آرایه مشخص می کند.
- عملگر new برای تخصیص پویای تعداد عناصر به آرایه استفاده می شود.
- اعلان و مقدار دهی آرایه نیاز ندارد در یک دستور یکسان باشد.
- در آرایه هایی از نوع مقدار، هر عنصر شامل یک مقدار از نوع اعلام شده خواهد بود.
- در آرایه هایی از نوع ارجاع، هر عنصر از آرایه یک ارجاع به یک شیء از نوع داده آرایه می باشد.

192

7.3.1 اختصاص یک آرایه و مقدار دهی عناصر آن

- آرایه ها را می توان با استفاده از کلمه **new** اختصاص داد و مشخص کرد چه تعداد عنصر آرایه می تواند نگهداری کند.
- آرایه می تواند با *لیست مقداردهی*، مقدار دهی اولیه شود.
  - فضای اختصاصی به آرایه – تعداد عناصر لیست اختصاصی اندازه آرا یه را مشخص می کند.
  - عناصر آرایه با مقادیر مشخص شده داخل لیست مقدار دهی اولیه خواهند شد.

193

InitArray.cs

```
1    // Fig 7.3: InitArray.cs
2    // Different ways of initializing arrays.
3
4    using System;
5    using System.Windows.Forms;
6
7    class InitArray
8    {
9       // main entry point for application
10      static void Main( string[] args )
11      {
12         string output = "";
13
14         int[] x;          // declare reference to an array
15         x = new int[ 10 ];  // dynamically allocate array and se
16                           // default values
17
18         // initializer list specifies number of elements
19         // and value of each element
20         int[] y = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
21
22         const int ARRAY_SIZE = 10; // named const
23         int[] z;                   // reference t
24
25         // allocate array of ARRAY_SIZE (i.e., 10) elements
26         z = new int[ ARRAY_SIZE ];
27
28         // set the values in the array
29         for ( int i = 0; i < z.Length; i++ )
30            z[ i ] = 2 + 2 * i;
31
32         output += "Subscript\tArray x\tArray y\tArray z\n";
33
```

Allocate x to be of size 10

Declare an integer array x

Declare a constant ARRAY_SIZE

Declare an integer array y and initialize it with values

Initialize the elements in z using a for loop

Declare an integer array z

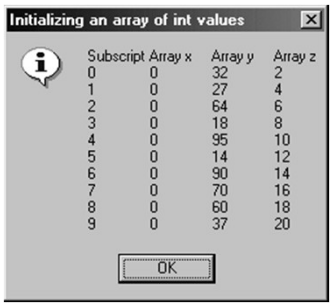
Initialize z to be of size ARRAY_SIZE

```
34          // output values for each array
35          for ( int i = 0; i < ARRAY_SIZE; i++ )
36             output += i + "\t" + x[ i ] + "\t" + y[ i ] +
37                "\t" + z[ i ] + "\n";
38
39          MessageBox.Show( output,
40             "Initializing an array of int values",
41             MessageBoxButtons.OK, MessageBoxIcon.Information );
42
43       } // end Main
44
45    } // end class InitArray
```

InitArray.cs

Add values in the arrays to output

Program Output

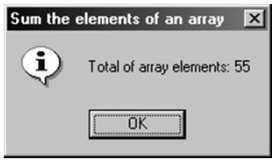


```
1    // Fig. 7.4: SumArray.cs
2    // Computing the sum of the elements in an array.
3
4    using System;
5    using System.Windows.Forms;
6
7    class SumArray
8    {
9       // main entry point for application
10      static void Main( string[] args )
11      {
12         int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
13         int total = 0;
14
15         for ( int i = 0; i < a.Length; i++ )
16            total += a[ i ];
17
18         MessageBox.Show( "Total of array elements: " + total,
19            "Sum the elements of an array",
20            MessageBoxButtons.OK, MessageBoxIcon.Information );
21
22      } // end Main
23
24   } // end class SumArray
```

Declare integer array a and initialize it

Total the contents of array a

SumArray.cs



Program Output

Histogram.cs

```
1    // Fig. 7.5: Histogram.cs
2    // Using data to create a histogram.
3
4    using System;
5    using System.Windows.Forms;
6
7    class Histogram
8    {
9       // main entry point for application
10      static void Main( string[] args )
11      {
12         int[] n = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
13         string output = "";
14
15         output += "Element\tvalue\tHistogram\n";
16
17         // build output
18         for ( int i = 0; i < n.Length; i++ )
19         {
20            output += "\n" + i + "\t" + n[ i ] + "\t";
21
22            for ( int j = 1; j <= n[ i ]; j++ ) // print a bar
23               output += "*";
24         }
25
26         MessageBox.Show( output, "Histogram Printing Program",
27            MessageBoxButtons.OK, MessageBoxIcon.Information );
28
29      } // end Main
30
31   } // end class Histogram
```

Declare an integer array n and initialize it

Create a bar for each element in n

Print a bar consisting of asterisks, corresponding to the value of the element in n

`Histogram.cs`
Program Output

7.4.4 استفاده از عناصر یک آرایه به عنوان شمارنده

- استفاده از عناصر آرایه برای پیگیری تعداد تکرار
- به عنوان مثال – برنامه نورد تاس
  - استفاده از ارزش نورد تاس به عنوان اندیس برای آرایه.
  - افزایش عنصر آرایه مربوط به زمانی که یک مقدار قالب نورد است.

RollDie.cs

```
1    // Fig. 7.6: RollDie.cs
2    // Rolling 12 dice.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10   using System.IO;
11
12   public class RollDie : System.Windows.Forms.Form
13   {
14      private System.Windows.Forms.Button rollButton;
15
16      private System.Windows.Forms.RichTextBox displayTextBox;
17
18      private System.Windows.Forms.Label dieLabel1;
19      private System.Windows.Forms.Label dieLabel2;
20      private System.Windows.Forms.Label dieLabel3;
21      private System.Windows.Forms.Label dieLabel4;
22      private System.Windows.Forms.Label dieLabel5;
23      private System.Windows.Forms.Label dieLabel6;
24      private System.Windows.Forms.Label dieLabel7;
25      private System.Windows.Forms.Label dieLabel8;
26      private System.Windows.Forms.Label dieLabel9;
27      private System.Windows.Forms.Label dieLabel10;
28      private System.Windows.Forms.Label dieLabel11;
29      private System.Windows.Forms.Label dieLabel12;
30
31      private System.ComponentModel.Container components =
32
33      Random randomNumber = new Random();
34      int[] frequency = new int[ 7 ];
35
```

Create a Random object

Declare an integer array frequency and allocate it enough memory to hold 7 integers

RollDie.cs

```
36      public RollDie()
37      {
38         InitializeComponent();
39      }
40
41      // Visual Studio .NET generated code
42
43      [STAThread]
44      static void Main()
45      {
46         Application.Run( new RollDie() );
47      }
48
49      private void rollButton_Click(
50         object sender, System.EventArgs e )
51      {
52         // pass the labels to a method that will
53         // randomly assign a face to each die
54         DisplayDie( dieLabel1 );
55         DisplayDie( dieLabel2 );
56         DisplayDie( dieLabel3 );
57         DisplayDie( dieLabel4 );
58         DisplayDie( dieLabel5 );
59         DisplayDie( dieLabel6 );
60         DisplayDie( dieLabel7 );
61         DisplayDie( dieLabel8 );
62         DisplayDie( dieLabel9 );
63         DisplayDie( dieLabel10 );
64         DisplayDie( dieLabel11 );
65         DisplayDie( dieLabel12 );
66
67         double total = 0;
68
69         for ( int i = 1; i < 7; i++ )
70            total += frequency[ i ];
```

Event handler for the rollButton Click event

Call method DisplayDie once for each Label

Total the number of times the dice have been rolled

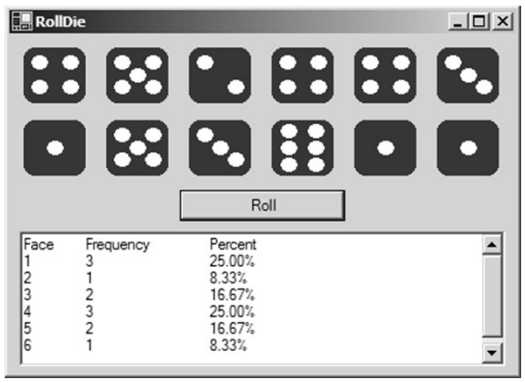
RollDie.cs

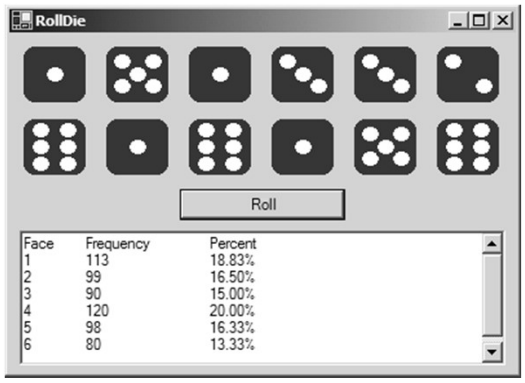
```
71
72          displayTextBox.Text = "Face\tFrequency\tPercent\n";
73
74          // output frequency values
75          for ( int x = 1; x < frequency.Length; x++ )
76          {
77             displayTextBox.Text += x + "\t" +
78                frequency[ x ] + "\t\t" + String.Format( "{0:N}",
79                frequency[ x ] / total * 100 ) + "%\n";
80          }
81
82       } // end Main
83
84       // simulates roll, display proper
85       // image and increment frequency
86       public void DisplayDie( Label dieLabel )
87       {
88          int face = randomNumber.Next( 1, 7 );
89
90          dieLabel.Image = Image.FromFile(
91             Directory.GetCurrentDirectory() +
92             "\\images\\die" + face + ".gif" );
93
94          frequency[ face ]++;
95       }
96
97    } // end class RollDie
```

Output die valu

Get a random number from 1 to 6

Display die image corresponding to the number rolles





`RollDie.cs`
Program Output

StudentPoll.cs

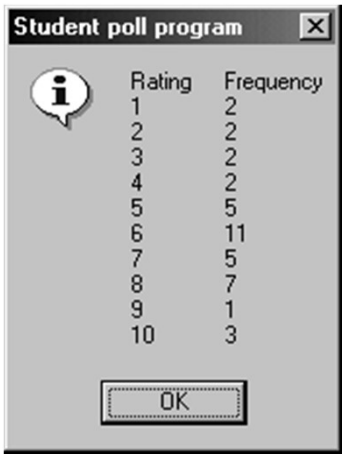
```
1    // Fig. 7.7: StudentPoll.cs
2    // A student poll program.
3
4    using System;
5    using System.Windows.Forms;
6
7    class StudentPoll
8    {
9       // main entry point for application
10      static void Main( string[] args )
11      {
12         int[] responses = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10, 1,
13            6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7,
14            5, 6, 6, 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
15
16         int[] frequency = new int[ 11 ];
17         string output = "";
18
19         // increment the frequency for each response
20         for ( int answer = 0; answer < responses.Length; answer++ )
21            ++frequency[ responses[ answer ] ];
22
23         output += "Rating\tFrequency\n";
24
25         // output results
26         for ( int rating = 1; rating < frequency.Length; rating++ )
27            output += rating + "\t" + frequency[ rating ] + "\n";
28
29         MessageBox.Show( output, "Student poll program",
30            MessageBoxButtons.OK, MessageBoxIcon.Information );
31
32      } // end method Main
33
34   } // end class StudentPoll
```

Declare and initialize integer array responses

For every element in responses, increment the frequency element that corresponds to the answer

Output the number of times each response appeared

`StudentPoll.cs`
Program Output



7.5 ارسال آرایه به یک تابع

- ارسال آرایه به عنوان آرگومان به متد با مشخص کردن نام آرایه (بدون براکت)
- آرایه های با ارجاع ارسال می شوند.
- عناصر آرایه بصورت فردی با مقدار ارسال می شوند

206

PassArray.cs

```
1    // Fig. 7.8: PassArray.cs
2    // Passing arrays and individual elements to methods.
3    using System;
4    using System.Drawing;
5    using System.Collections;
6    using System.ComponentModel;
7    using System.Windows.Forms;
8    using System.Data;
9
10   public class PassArray : System.Windows.Forms.Form
11   {
12      private System.Windows.Forms.Button showOutputButton;
13      private System.Windows.Forms.Label outputLabel;
14
15      // Visual Studio .NET generated code
16
17      [STAThread]
18      static void Main()
19      {
20         Application.Run( new PassArray() );
21      }
22
23      private void showOutputButton_Click( object sender,
24         System.EventArgs e )
25      {
26         int[] a = { 1, 2, 3, 4, 5 };
27
28         outputLabel.Text = "Effects of passing entire array " +
29            "call-by-reference:\n\nThe values of the original " +
30            "array are:\n\t";
31
32         for ( int i = 0; i < a.Length; i++ )
33            outputLabel.Text += "   " + a[ i ];
34
35         ModifyArray( a );   // array is passed by reference
```

Declare and initialize integer array a

Call method ModifyArray, pass array a as an argument by reference

Output contents of array a

PassArray.cs

```
36
37          outputLabel.Text +=
38             "\n\nThe values of the modified array are:\n\t";
39
40          // display elements of array a
41          for ( int i = 0; i < a.Length; i++ )
42             outputLabel.Text += "   " + a[ i ];
43
44          outputLabel.Text += "\n\nEffects of passing array " +
45             "element call-by-value:\n\na[ 3 ] before " +
46             "ModifyElement: " + a[ 3 ];
47
48          // array element passed call-by-value
49          ModifyElement( a[ 3 ] );
50
51          outputLabel.Text +=
52             "\na[ 3 ] after ModifyElement: " + a[ 3 ];
53       }
54
55       // method modifies the array it receives,
56       // original will be modified
57       public void ModifyArray( int[] b )
58       {
59          for ( int j = 0; j < b.Length; j++ )
60             b[ j ] *= 2;
61       }
62
63       // method modifies the integer passed to it
64       // original will not be modified
65       public void ModifyElement( int e )
66       {
67          outputLabel.Text +=
68             "\nvalue received in ModifyElement: " + e;
69
```

Replace every element in array by twice its value

Output array a after ModifyArray changed the contents

Call method ModifyElement, pass element of array a that is at index 3

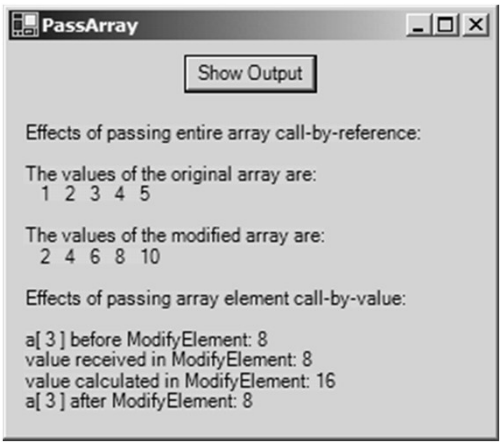
```
70          e *= 2;
71
72          outputLabel.Text +=
73             "\nvalue calculated in ModifyElement: " + e;
74       }
75    }
```

PassArray.cs

Multiply argument by two

This does not change value of element in original array, because the element was passed by value

Program Output



7.6 Passing Arrays by Value and by Reference

- Variables that "store" object, actually store references to those objects
- A reference is a location in computer's memory where the object itself is stored
- Passing value types to methods
  - A copy of the variable is made
  - Any changes to variable in method do not effect the original variable
- Passing reference types to methods
  - A copy of the reference to the object is made
  - Any changes to the reference in the method do not effect the original variable
  - Any changes to the contents of the object in the method, **do** effect the object outside the method

7.6 Passing Arrays by Value and by Reference

- Keyword **ref** may be used to pass arguments to method by reference
  - Value type variables are not copied – modifying the variable in the method will modify the variable outside the method
  - References to objects are not copied – modifying the reference in the method will modify the reference outside the method
- Programmers have to be careful when using ref
  - May lead to references being set to null
  - May lead to methods modifying variable values and references in ways that are not desired

```
1    // Fig. 7.9: ArrayReferenceTest.cs
2    // Testing the effects of passing array references
3    // by value and by reference.
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10
11   public class ArrayReferenceTest : System.Windows.Forms.Form
12   {
13      private System.Windows.Forms.Label outputLabel;
14      private System.Windows.Forms.Button showOutputButton;
15
16      [STAThread]
17      static void Main()
18      {
19         Application.Run( new Arr
20      }
21
22      private void showOutputButton_Click( object sender,
23         System.EventArgs e )
24      {
25         // create and initialize firstArray
26         int[] firstArray = { 1, 2, 3 };
27
28         // copy firstArray reference
29         int[] firstArrayCopy = firstArray;
30
31         outputLabel.Text +=
32            "Test passing firstArray reference by value";
33
34         outputLabel.Text += "\n\nContents of firstArray " +
35            "before calling FirstDouble:\n\t";
```

Declare and initialize integer array firstArray

Declare integer array firstArrayCopy and have it reference firstArray

ArrayReferenceTest.cs

```
36
37          // print contents of firstArray
38          for ( int i = 0; i < firstArray.Length; i++ )
39             outputLabel.Text += firstArray[ i ] + " ";
40
41          // pass reference firstArray by value to FirstDouble
42          FirstDouble( firstArray );
43
44          outputLabel.Text += "\n\nContents of
45             "calling FirstDouble\n\t";
46
47          // print contents of firstArray
48          for ( int i = 0; i < firstArray.Length; i++ )
49             outputLabel.Text += firstArray[ i ] + " ";
50
51          // test whether reference was changed by FirstDouble
52          if ( firstArray == firstArrayCopy )
53             outputLabel.Text +=
54                "\n\nThe references refer to the same array\n";
55          else
56             outputLabel.Text +=
57                "\n\nThe references refer to different arrays\n";
58
59          // create and initialize secondArray
60          int[] secondArray = { 1, 2, 3 };
61
62          // copy secondArray reference
63          int[] secondArrayCopy = secondArray;
64
65          outputLabel.Text += "\nTest passing secondArray " +
66             "reference by reference";
67
68          outputLabel.Text += "\n\nContents of secondArray " +
69             "before calling SecondDouble:\n\t";
70
```

ArrayReferenceTest.cs

Test whether firstArray and firstArrayCopy reference the same object

Declare integer array secondArrayCopy and set it to reference secondArray

Declare and initialize integer arry secondArray

Call method FirstDouble on firstArray

Output contents of firstArray

ArrayReferenceTest.cs

```
71          // print contents of secondArray before method call
72          for ( int i = 0; i < secondArray.Length; i++ )
73             outputLabel.Text += secondArray[ i ] + " ";
74
75          SecondDouble( ref secondArray );
76
77          outputLabel.Text += "\n\nContents of
78             "after calling SecondDouble:\n\n";
79
80          // print contents of secondArray after method call
81          for ( int i = 0; i < secondArray.Length; i++ )
82             outputLabel.Text += secondArray[ i ] + " ";
83
84          // test whether reference was changed by SecondDouble
85          if ( secondArray == secondArrayCopy )
86             outputLabel.Text +=
87                "\n\nThe references refer to the same array\n";
88          else
89             outputLabel.Text +=
90                "\n\nThe references refer to different arrays\n";
91
92       } // end method showOutputButton_Click
93
94       // modify elements of array and attempt to modify
95       // reference
96       void FirstDouble( int[] array )
97       {
98          // double each element's value
99          for ( int i = 0; i < array.Length; i++ )
100            array[ i ] *= 2;
101
102         // create new reference and assign it to array
103         array = new int[] { 11, 12, 13 };
104      }
105
```

Test whether secondArray and secondArrayCopy reference the same object

Replace each element in the array by twice its value

Set array to reference a new integer array containing the values 11, 12 and 13

and pass secondArray by reference

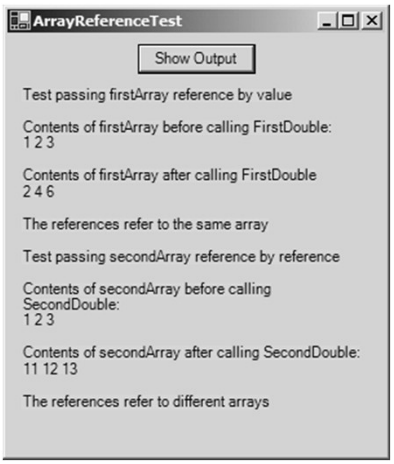
Output contents of secondArray

```
106     // modify elements of array and change reference array
107     // to refer to a new array
108     void SecondDouble( ref int[] array )
109     {
110        // double each element's value
111        for ( int i = 0; i < array.Length; i++ )
112           array[ i ] *= 2;
113
114        // create new reference and assign it to array
115        array = new int[] { 11, 12, 13 };
116     }
117  }
```

Set array to reference a new integer array containing the values 11, 12 and 13

ArrayReferenceTest.cs

Program Output



7.7 Sorting Arrays

- Sorting data is important in many applications
- Bubble Sort – array of size n
  - Make n passes through the array
  - For each pass, compare every pair of successful elements
    - If the first is larger then the second, swap the elements
  - Easy to program
  - Runs slowly
- .NET Framework includes high-speed sorting capabilities

BubbleSorter.cs

```
1    // Fig. 7.10: BubbleSorter.cs
2    // Sorting an array's values into ascending order.
3    using System;
4    using System.Drawing;
5    using System.Collections;
6    using System.ComponentModel;
7    using System.Windows.Forms;
8    using System.Data;
9
10   public class BubbleSorter : System.Windows.Forms.Form
11   {
12      private System.Windows.Forms.Button sortButton;
13      private System.Windows.Forms.Label outputLabel;
14
15      // Visual Studio .NET generated code
16
17      [STAThread]
18      static void Main()
19      {
20         Application.Run( new BubbleSorter() );
21      }
22
23      private void sortButton_Click( object sender,
24            System.EventArgs e )
25      {
26         int[] a =  { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
27
28         outputLabel.Text += "Data items in original order\n";
29
30         for ( int i = 0; i < a.Length; i++ )
31            outputLabel.Text += "   " + a[ i ];
32
33         // sort elements in array a
34         BubbleSort( a );
35
```

Declare and initialize array a

Output the contents of array a

Call method Bubble sort on array a

```
36          outputLabel.Text += "\n\nData items in ascending order\n";
37
38          for ( int i = 0; i < a.Length; i++ )
39             outputLabel.Text += "   " + a[ i ];
40
41       } // end method sortButton_Click
42
43       // sort the elements of an array with bu
44       public void BubbleSort( int[] b )
45       {
46          for ( int pass = 1; pass < b.Length; pass++ ) // passes
47
48             for ( int i = 0; i < b.Length - 1; i++ )   // one pass
49
50                if ( b[ i ] > b[ i + 1 ] )      // one comparison
51                   Swap( b, i );                // one swap
52       }
53
54       // swap two elements of an array
55       public void Swap( int[] c, int first )
56       {
57          int hold;      // temporary holding area for swap
58
59          hold = c[ first ];
60          c[ first ] = c[ first + 1 ];
61          c[ first + 1 ] = hold;
62       }
63    }
```

bbleSorter.cs

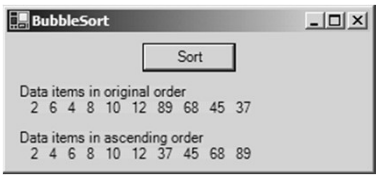
Swaps two elements of an array

If an given element is bigger then the following element, swap the elements

rray a

Perform b.Le

Perform contents of for loop for each element of array b



Program Output

7.8 جستجوی آرایه: جستجوی خطی و جستجوی دودویی

- آرایه ها ممکن است بزرگ باشند
- گاهی اوقات لازم است تعیین کنیم که آیا عنصر خاصی در آرایه است یا نه
- جستجوی خطی
- جستجوی دودویی

7.8.1 جستجو در آرایه با جستجوی خطی

- اندیس عنصر جستجو شونده داخل آرایه را برگشت می دهد
- شروع جستجو از ابتدای آرایه می باشد، و پی در پی ادامه می یابد
- به طور متوسط، نیمی از آرایه باید برای پیدا کردن عنصر مورد نظر جستجو شود
- برای آرایه های کوچک و مرتب نشده به خوبی کار می کند

220

LinearSearcher.cs

```
1    // Fig. 7.11: LinearSearcher.cs
2    // Demonstrating linear searching of an array.
3    using System;
4    using System.Drawing;
5    using System.Collections;
6    using System.ComponentModel;
7    using System.Windows.Forms;
8    using System.Data;
9
10   public class LinearSearcher : System.Windows.Forms.Form
11   {
12      private System.Windows.Forms.Button searchButton;
13      private System.Windows.Forms.TextBox inputTextBox;
14      private System.Windows.Forms.Label outputLabel;
15
16      int[] a = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26,
17                  28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50 };
18
19      // Visual Studio .NET generated code
20
21      [STAThread]
22      static void Main()
23      {
24         Application.Run( new LinearSearcher() );
25      }
26
27      private void searchButton_Click( object sender,
28         System.EventArgs e )
29      {
30         int searchKey = Int32.Parse( inputTextBox.Text );
31
32         int elementIndex = LinearSearch( a, searchKey );
33
```

Retrieve the number user input as the search key

Perform linear search for the search key

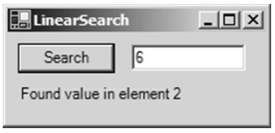
```
34          if ( elementIndex != -1 )
35             outputLabel.Text =
36                "Found value in element " + elementIndex;
37
38          else
39             outputLabel.Text = "Value not found";
40
41       } // end method searchButton_Click
42
43       // search array for the specified key value
44       public int LinearSearch( int[] array, int key )
45       {
46          for ( int n = 0; n < array.Length; n++ )
47          {
48             if ( array[ n ] == key )
49                return n;
50          }
51
52          return -1;
53
54       } // end method LinearSearch
55
56    } // end class LinearSearcher
```

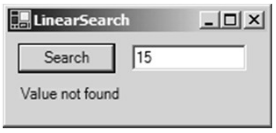
LinearSearcher.cs

If the index of the search key is –1, then element was not found

If search failed, return -1

Start at beginning of array
Check every element to see if it matches the search key.
If it does, return the current index





Program Output

7.8.2 جستجو در آرایه با جستجوی دودویی

- آرایه باید مرتب شده باشد
- از بین بردن نیمی از عناصر جستجو در هر مرحله
- الگوریتم
  - عنصر وسطی را پیدا کن
  - با کلید جستجو مقایسه کن
    - اگر آنها با هم برابرند عنصر یافت شده است، اندیس عنصر وسط را برگشت بده
    - اگر کلید جستجو کمتر از عنصر وسط است، جستجو را در نیمه اول از آرایه ادامه بده
    - اگر کلید جستجو را بزرگتر از عنصر وسط است، جستجو را در نیمه دوم از آرایه ادامه بده
  - بالا را تکرار کنید تا زمانی که کلید جستجو برابر عنصر وسط شود، و یا زیر آرایه جستجو شونده عنصری نداشته باشد (که در این صورت کلید جستجو در آرایه نیست)

223

```
1    // Fig. 7.12: BinarySearchTest.cs
2    // Demonstrating a binary search of an array.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10
11   public class BinarySearchTest : System.Windows.Forms.Form
12   {
13      private System.Windows.Forms.Label promptLabel;
14
15      private System.Windows.Forms.TextBox inputTextBox;
16
17      private System.Windows.Forms.Label resultLabel;
18      private System.Windows.Forms.Label displayLabel;
19      private System.Windows.Forms.Label outputLabel;
20
21      private System.Windows.Forms.Button findButton;
22
23      private System.ComponentModel.Container components = null;
24
25      int[] a = { 0, 2, 4, 6, 8, 10, 12, 14, 16,
26                  18, 20, 22, 24, 26, 28 };
27
28      // Visual Studio .NET generated code
29
30      // main entry point for application
31      [STAThread]
32      static void Main()
33      {
34         Application.Run( new BinarySearchTest() );
35      }
```

Declare and initialize integer array a

BinarySearchTest.cs

BinarySearchTest.cs

```
36
37      // searches for an element by calling
38      // BinarySearch and displaying results
39      private void findButton_Click( object sender,
40         System.EventArgs e )
41      {
42         int searchKey = Int32.Parse( inputTextBox.Text );
43
44         // initialize display string for the new search
45         outputLabel.Text = "Portions of array searched\n";
46
47         // perform the binary search
48         int element = BinarySearch( a, searchKey );
49
50         if ( element != -1 )
51            displayLabel.Text = "Found value in element " +
52               element;
53         else
54            displayLabel.Text = "Value not found";
55
56      } // end findButton_Click
57
58      // searchs array for specified key
59      public int BinarySearch( int[] array, int key )
60      {
61         int low = 0;                  // low subscript
62         int high = array.Length - 1;  // high subscript
63         int middle;                   // middle subscript
64
65         while ( low <= high )
66         {
67            middle = ( low + high ) / 2;
68
```

Retriev

If the low index is less then the high index then try to find element (otherwise, element is not in the array

Compute midpoint of current search space

Call method BinarySearch on array a with the user input as the search key

If –1 was returned, then search key was not found

BinarySearchTest.cs

```
69            // the following line displays the portion
70            // of the array currently being manipulated during
71            // each iteration of the binary search loop
72            BuildOutput( a, low, middle, high );
73
74            if ( key == array[ middle ] )   // match
75               return middle;
76            else if ( key < array[ middle ] )
77               high = middle - 1;   // search low end of array
78            else
79               low = middle + 1;
80
81         } // end BinarySearch
82
83         return -1;  // search key not found
84
85      } // end method BinarySearch
86
87      public void BuildOutput(
88         int[] array, int low, int mid, int high )
89      {
90         for ( int i = 0; i < array.Length; i++ )
91         {
92            if ( i < low || i > high )
93               outputLabel.Text += "    ";
94
95            // mark middle element in output
96            else if ( i == mid )
97               outputLabel.Text +=
98                  array[ i ].ToString( "00" ) + "* ";
```

Output all elements of the array within two indices and mark the middle element. Print spaces for the other elements

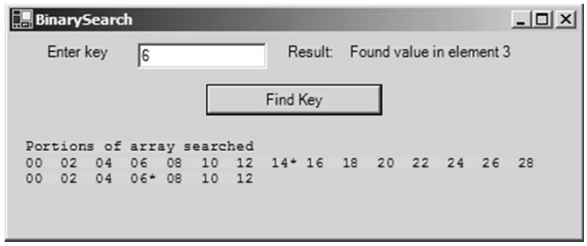
If the middle element search key, return the middle element

If the key value is smaller then the middle element, set the high index to be one less then the current middle index

Otherwise, set the low index to be one more then the middle index

```
99             else
100               outputLabel.Text +=
101                  array[ i ].ToString( "00" ) + "   ";
102            }
103
104            outputLabel.Text += "\n";
105
106         } // end BuildOutput
107
108      } // end class BinarySearchTest
```

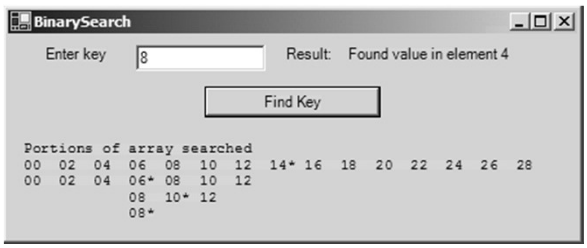
BinarySearchTest.cs

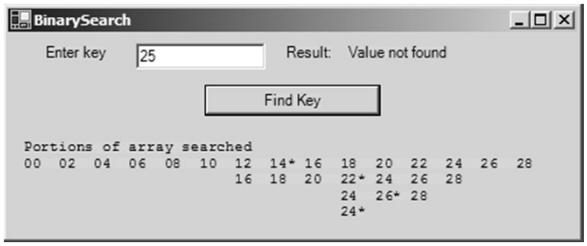


Program Output

BinarySearchTest.cs

Program Output





7.9 آرایه های چند بعدی

- نیازمند دو یا چند اندیس برای شناسایی یک عنصر خاص می باشد
- آرایه هایی که به دو اندیس برای شناسایی عناصر نیاز دارند آرایه های دو بعدی نامیده می شوند.
- آرایه مستطیلی
  - اغلب نشان دهنده جداول که در آن هر سطر به همان اندازه است و هر ستون به همان اندازه است.
  - بر اساس قرارداد، اولین اندیس مشخص کننده شماره ردیف عنصر و دومین اندیس مشخص کننده شماره ستون عنصر می باشد
- Jagged Arrays
  - Arrays of arrays
  - Arrays that compose jagged arrays can be of different lengths

229

7.9 Multiple-Subscripted Arrays

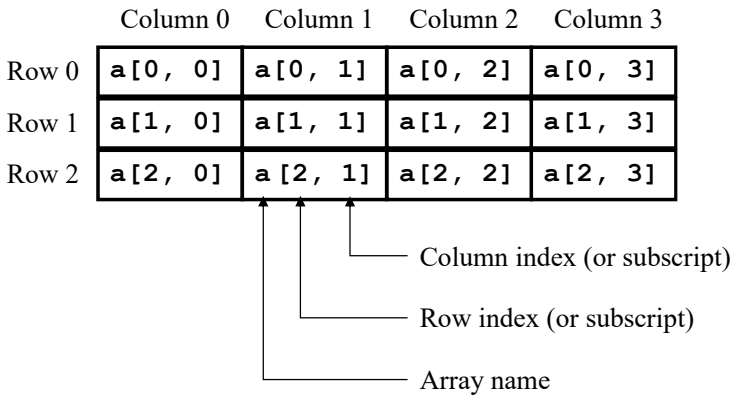


Fig. 7.13 Double-subscripted array with three rows and four columns.

TwoDimensionalArrays.cs

```
1    // Fig. 7.14: TwoDimensionalArrays.cs
2    // Initializing two-dimensional arrays.
3    using System;
4    using System.Drawing;
5    using System.Collections;
6    using System.ComponentModel;
7    using System.Windows.Forms;
8    using System.Data;
9
10   public class TwoDimensionalArrays : System.Windows.Forms.Form
11   {
12      private System.Windows.Forms.Button showOutputButton;
13      private System.Windows.Forms.Label outputLabel;
14
15      // Visual Studio .NET generated code
16
17      [STAThread]
18      static void Main()
19      {
20         Application.Run( new TwoDimensionalArrays() );
21      }
22
23      private void showOutputButton_Click( object sender,
24         System.EventArgs e )
25      {
26         // declaration and initialization of rectangular array
27         int[,] array1 = new int[,] { { 1, 2, 3 }, { 4, 5, 6 } };
28
29         // declaration and initialization of jagged array
30         int[][] array2 = new int[ 3 ][];
31         array2[ 0 ] = new int[] { 1, 2 };
32         array2[ 1 ] = new int[] { 3 };
33         array2[ 2 ] = new int[] { 4, 5, 6 };
34
35         outputLabel.Text += "Values in array1 by row are\n";
```

Initialize the third element in array2 to be an array that contains 3 integers

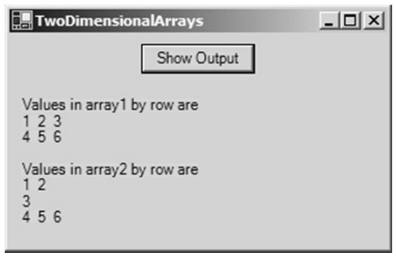
l element in
y that
tw contains 1 integer

lare a jagged array named array2 with 3 rows

Declare and initialize a rectangular integer array named array1

```
36
37          // output values in array1
38          for ( int i = 0; i < array1.GetLength( 0 ); i++ )
39          {
40             for ( int j = 0; j < array1.GetLength( 1 ); j++ )
41                outputLabel.Text += array1[ i, j ] + "  ";
42
43             outputLabel.Text += "\n";
44          }
45
46          outputLabel.Text += "\nValues in array2 by row are\n";
47
48          // output values in array2
49          for ( int i = 0; i < array2.Length; i++ )
50          {
51             for ( int j = 0; j < array2[ i ].Length; j++ )
52                outputLabel.Text += array2[ i ][ j ] + "  ";
53
54             outputLabel.Text += "\n";
55          }
56
57       } // end method showOutputButton_Click
58
59    } // end class TwoDimensionalArrays
```

TwoDimensionalArrays.cs



Program Output

DoubleArray.cs

```
1    // Fig. 7.15: DoubleArray.cs
2    // Manipulating a double-subscripted array.
3    using System;
4    using System.Drawing;
5    using System.Collections;
6    using System.ComponentModel;
7    using System.Windows.Forms;
8    using System.Data;
9
10   public class DoubleArray : System.Windows.Forms.Form
11   {
12      private System.Windows.Forms.Button showOutputButton;
13      private System.Windows.Forms.Label outputLabel;
14
15      int[][] grades;
16      int students, exams;
17
18      // Visual Studio .NET generated code
19
20      [STAThread]
21      static void Main()
22      {
23         Application.Run( new DoubleArray() );
24      }
25
26      private void showOutputButton_Click( object sender,
27         System.EventArgs e )
28
29      {
30         grades = new int[ 3 ][];
31         grades[ 0 ] =  new int[]{ 77, 68, 86, 73 };
32         grades[ 1 ] =  new int[]{ 96, 87, 89, 81 };
33         grades[ 2 ] =  new int[]{ 70, 90, 86, 81 };
34
```

Initialize array grades to have 3 rows

Initialize each element in array grades

```
35          students = grades.Length;      // number of students
36          exams = grades[ 0 ].Length;    // number of exams
37
38          // line up column headings
39          outputLabel.Text += "                    ";
40
41          // output the column headings
42          for ( int i = 0; i < exams; i++ )
43             outputLabel.Text += "[" + i + "]  ";
44
45          // output the rows
46          for ( int i = 0; i < students; i++ )
47          {
48             outputLabel.Text += "\ngrades[" + i + "]   ";
49
50             for ( int j = 0; j < exams; j++ )
51                outputLabel.Text += grades[ i ][ j ] + "    ";
52          }
53
54          outputLabel.Text += "\n\nLowest grade: " + Minimum() +
55             "\nHighest grade: " + Maximum() + "\n";
56
57          for ( int i = 0; i < students; i++ )
58             outputLabel.Text += "\nAverage for student " + i + " is " +
59                Average( grades[ i ] );
60
61       } // end method showOutputButton_Click
62
```

Output each row

Output each element of the row

Output the minimum and maximum grades

Output the average for each row

DoubleArray.cs

DoubleArray.cs

```
63      // find minimum grade in grades array
64      public int Minimum()
65      {
66         int lowGrade = 100;
67
68         for ( int i = 0; i < students; i++ )
69
70            for ( int j = 0; j < exams; j++ )
71
72               if ( grades[ i ][ j ] < lowGrade )
73                  lowGrade = grades[ i ][ j ];
74
75         return lowGrade;
76      }
77
78      // find maximum grade in grades array
79      public int Maximum()
80      {
81         int highGrade = 0;
82
83         for ( int i = 0; i < students; i++ )
84
85            for ( int j = 0; j < exams; j++ )
86
87               if ( grades[ i ][ j ] > highGrade )
88                  highGrade = grades[ i ][ j ];
89
90         return highGrade;
91      }
92
```

Examine each element in grades array

If the current array element higher then the highest grade, set the value of highGrade to be the current element

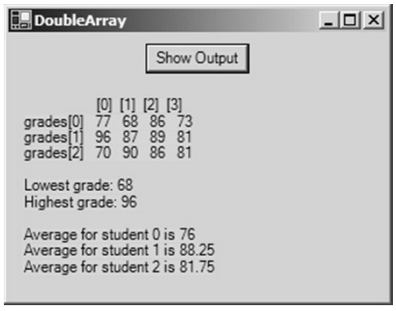
If the current array element is less then the lowest grade, set the value of lowGrade to be the current element

```
93       // determine average grade for a particular student
94       public double Average( int[] setOfGrades )
95       {
96          int total = 0;
97
98          for ( int i = 0; i < setOfGrades.Length; i++ )
99             total += setOfGrades[ i ];
100
101         return ( double ) total / setOfGrades.Length;
102      }
103
104   } // end class DoubleArray
```

DoubleArray.cs

Total the grades for the array

Divide the total by the number of grades



Program Output

7.10 foreach Repetition Structure

- The foreach repetition structure is used to iterate through values in data structures such as arrays
- No counter
- A variable is used to represent the value of each element

ForEach.cs

```
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   }
```

Use the foreach loop to examine each element in the array

If the current array element is smaller then lowGrade, set lowGrade to contain the value of the current element

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