Chapter 9

How to work with dates and strings

Objectives

Applied

1. Given the date-handling requirements of an application, write the code that satisfies the requirements.
2. Given the string-handling requirements of an application, write the code that satisfies the requirements.
3. Given the formatting requirements of an application, use the Format method of the String class or interpolated strings to provide the formatting.

Knowledge

1. Describe the way a DateTime variable is stored.
2. Explain how a String object differs from a StringBuilder object.

The syntax for creating a DateTime value

```
DateTime variableName = new DateTime(year, month, day
    [, hour, minute, second[, millisecond]]);
```

How to create a DateTime value using the Parse method

```
DateTime variableName = DateTime.Parse(string);
```

How to create a DateTime value using the TryParse method

```
DateTime variableName;
DateTime.TryParse(string, out variableName);
```

Statements that create DateTime values

```
DateTime startDate =
    new DateTime(2016, 01, 30);                 // 1/30/2016 12:00 AM
DateTime startDateAndTime =
    new DateTime(2016, 1, 30, 14, 15, 0);       // 1/30/2016 2:15 PM

DateTime startDate =
    DateTime.Parse("01/30/16");                 // 1/30/2016 12:00 AM
DateTime startDateAndTime
    = DateTime.Parse("Jan 30, 2016 2:15 PM");  // 1/30/2016 2:15 PM
DateTime invoiceDate = DateTime.Parse(txtInvoiceDate.Text);

DateTime invoiceDate;
DateTime.TryParse(txtInvoiceDate.Text, out invoiceDate);
```

Valid date formats

```
01/30/2016
1/30/16
01-01-2016
1-1-16
2016-01-01
Jan 30 2016
January 30, 2016
```

Valid time formats

```
2:15 PM
14:15
02:15:30 AM
```

DateTime properties for getting the current date and time

Property	Description
`Now`	Returns the current date and time.
`Today`	Returns the current date.

Statements that get the current date and time

```
DateTime currentDateTime = DateTime.Now; // 1/30/2016 4:24:59 AM
DateTime currentDate = DateTime.Today;   // 1/30/2016 12:00:00 AM
```

DateTime methods for formatting a date or time

Method	Description
`ToLongDateString()`	Converts the DateTime value to a string that includes the day of the week name, the month name, the day of the month, and the year.
`ToShortDateString()`	Converts the DateTime value to a string that includes the numeric month, day, and year.
`ToLongTimeString()`	Converts the DateTime value to a string that includes the hours, minutes, and seconds.
`ToShortTimeString()`	Converts the DateTime value to a string that includes the hours and minutes.

Statements that format dates and times

```
string longDate =
    currentDateTime.ToLongDateString();     // Saturday, January 30, 2016
string shortDate =
    currentDateTime.ToShortDateString();    // 1/30/2016
string longTime =
    currentDateTime.ToLongTimeString();     // 4:24:59 AM
string ShortTime =
    currentDateTime.ToShortTimeString();    // 4:24 AM
```

Note

- The format that's used for a date or time depends on your computer's regional settings.

Properties for working with dates and times

- **Date**
- **Month**
- **Day**
- **Year**
- **Hour**
- **Minute**
- **Second**
- **TimeOfDay**
- **DayOfWeek**
- **DayOfYear**

Methods for working with dates and times

- **DaysInMonth(**year, month**)**
- **IsLeapYear(**year**)**

Statements that get information about a date or time

```
DateTime currentDateTime = DateTime.Now;   // 1/30/2016 10:26:35 AM
int month = currentDateTime.Month;                 // 1
int hour = currentDateTime.Hour;                   // 10
int dayOfYear = currentDateTime.DayOfYear;         // 30
int daysInMonth = DateTime.DaysInMonth(2016, 2);   // 29
bool isLeapYear = DateTime.IsLeapYear(2016);       // true
```

Code that uses the DayOfWeek property and enumeration

```
DayOfWeek dayOfWeek = currentDateTime.DayOfWeek;
string message = "";
if (dayOfWeek == DayOfWeek.Saturday ||
    dayOfWeek == DayOfWeek.Sunday)
{
    message = "Weekend";
}
else
{
    message = "Weekday";
}
```

Methods for performing operations on dates and times

```
AddDays(days)
AddMonths(months)
AddYears(years)
AddHours(hours)
AddMinutes(minutes)
AddSeconds(seconds)
Add(timespan)
Subtract(datetime)
```

Statements that perform operations on dates and times

```
DateTime dateTime =
    DateTime.Parse("3/1/2016 13:28");         // 3/1/2016 1:28:00 PM
DateTime dueDate = dateTime.AddMonths(2);     // 5/1/2016 1:28:00 PM
dueDate = dateTime.AddDays(60);               // 4/30/2016 1:28:00 PM
DateTime runTime = dateTime.AddMinutes(30);   // 3/1/2016 1:58:00 PM
runTime = dateTime.AddHours(12);              // 3/2/2016 1:28:00 AM
```

Code that results in a TimeSpan value

```
DateTime currentDate = DateTime.Today;    // 1/30/2016
dueDate = DateTime.Parse("2/15/2016");    // 2/15/2016
TimeSpan timeTillDue =
    dueDate.Subtract(currentDate);        // 16:00:00:00
int daysTillDue = timeTillDue.Days;       // 16
```

Description

- A TimeSpan value represents a period of time stored as ticks. You can use the Days, Hours, Minutes, and Seconds properties of a TimeSpan value to get portions of that value.

A statement that uses the – operator to subtract two dates

```
TimeSpan timeTillDue = dueDate - currentDate;    // 16:00:00:00
```

An if statement that uses the > operator on DateTime values

```
bool pastDue = false;
if (currentDate > dueDate)
    pastDue = true;
```

Common properties of the String class

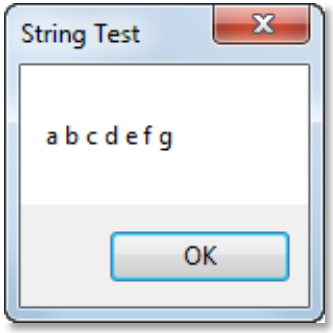
```
[index]
Length
```

Code that uses an index to access a character in a string

```
string chars = "abcdefg";
char a = chars[0];                  // 'a'
char b = chars[1];                  // 'b'
```

Code that uses a for loop to access each character in the string

```
string charsAndSpaces = "";
for (int i = 0; i < chars.Length; i++)
    charsAndSpaces += chars[i] + " ";
MessageBox.Show(charsAndSpaces, "String Test");
```



Code that uses a foreach loop to access each character in the string

```
string charsAndSpaces = "";
foreach (char c in chars)
    charsAndSpaces += c + " ";
MessageBox.Show(charsAndSpaces, "String Test");
```

Common methods of the String class

```
StartsWith(string)
EndsWith(string)
IndexOf(string[, startIndex])
LastIndexOf(string[, startIndex])
Insert(startIndex, string)
PadLeft(totalWidth)
PadRight(totalWidth)
Remove(startIndex, count)
Replace(oldString, newString)
Substring(startIndex[, length])
ToLower()
ToUpper()
Trim()
Split(splitCharacters)
```

Code that uses the StartsWith and EndsWith methods

```
bool startsWithABC = chars.StartsWith("abc");  // true
bool endsWithABC = chars.EndsWith("abc");      // false
```

Code that uses the IndexOf method

```
string companyName = "Mike Murach and Associates";
int index1 = companyName.IndexOf(" ");      // 4
int index2 = companyName.IndexOf(' ');      // 4
int index3 = companyName.IndexOf("Inc.");   // -1
int index4 = companyName.LastIndexOf(" ");  // 15
```

Code that uses the Remove, Insert, and Replace methods

```
companyName = companyName.Remove(0, index1 + 1);
companyName = companyName.Insert(companyName.Length, ", Inc.");
companyName = companyName.Replace(
    "and", "And");                // Murach And Associates, Inc.
```

Code that uses the Substring, ToUpper, and ToLower methods

```
string firstName = "anne";
string firstLetter = firstName.Substring(0, 1).ToUpper();
string otherLetters = firstName.Substring(1).ToLower();
firstName = firstLetter + otherLetters;       // Anne
```

Code that copies one string to another string

```
string s1 = "abc";
string s2 = s1;    // this copies the value stored in s1 to s2
s2 = "def";        // this doesn't change the value stored in s1
```

Code that parses a first name from a name string

```
string fullName = " Edward C Koop   ";         // " Edward C Koop   "
fullName = fullName.Trim();                    // "Edward C Koop"
int firstSpace = fullName.IndexOf(" ");        // 6
string firstName = "";
if (firstSpace == -1)
    firstName = fullName;
else
    firstName = fullName.Substring(0, firstSpace);    // Edward
```

Code that parses a string that contains an address

```
string address = " |805 Main Street|Dallas|TX|12345| ";
address = address.Trim();
if (address.StartsWith("|"))
    address = address.Remove(0, 1);
if (address.EndsWith("|"))
    address = address.Remove(address.Length - 1, 1);
int cityIndex = address.IndexOf("|") + 1;
int stateIndex = address.IndexOf("|", cityIndex) + 1;
int zipIndex = address.IndexOf("|", stateIndex) + 1;
string street = address.Substring(0, cityIndex - 1);
string city = address.Substring(
    cityIndex, stateIndex - cityIndex - 1);
string state = address.Substring(
    stateIndex, zipIndex - stateIndex - 1);
string zipCode = address.Substring(zipIndex);
```

Code that uses the Split method to parse the name string

```
string fullName = " Edward C Koop   ";
fullName = fullName.Trim();
string[] names = fullName.Split(' ');
string firstName = names[0];              // Edward
```

Code that uses the Split method to parse the address string

```
address = address.Trim();
if (address.StartsWith("|"))
    address = address.Remove(0, 1);
string[] columns = address.Split('|');
string street = columns[0];               // 805 Main Street
string city = columns[1];                 // Dallas
string state = columns[2];                // TX
string zipCode = columns[3];              // 12345
```

Code that adds hyphens to a phone number

```
string phoneNumber = "9775551212";
phoneNumber = phoneNumber.Insert(3, "-");
phoneNumber = phoneNumber.Insert(7, "-");  // 977-555-1212
```

Code that replaces the hyphens in a date with slashes

```
string date = "12-27-2015";
date = date.Replace("-", "/");             // 12/27/2015
```

The syntax of the Parse and TryParse methods for validating decimal values

```
Parse(string, style1[ | style2]...)
TryParse(string, style1[ | style2]..., formatProvider, result)
```

Common members of the NumberStyles enumeration for decimal values

Member	Characters allowed
AllowCurrencySymbol	Digits and a $ sign
AllowDecimalPoint	Digits and a decimal point
AllowLeadingWhiteSpace	Digits and leading whitespace
AllowTrailingWhiteSpace	Digits and trailing whitespace
AllowLeadingSign	Digits and a leading sign
AllowTrailingSign	Digits and a trailing sign

Common members of the NumberStyles enumeration for decimal values (cont.)

Member	Characters allowed
`AllowParentheses`	Parentheses around the digits to indicate a negative value
`AllowThousands`	Digits and commas used as thousands separators
`Currency`	Digits and all of the characters listed above
`Number`	Digits and all of the characters listed above except for a $ sign and parentheses
`None`	Digits only

A Parse method that allows entries with any decimal characters

```
decimal invoiceTotal = Decimal.Parse(txtTotal.Text,
NumberStyles.Currency);
```

A TryParse method that allows entries with a $ sign and a decimal point

```
decimal invoiceTotal = 0m;
Decimal.TryParse(txtTotal.Text,
    NumberStyles.AllowCurrencySymbol |
NumberStyles.AllowDecimalPoint),
    CultureInfo.CurrentCulture, out invoiceTotal;
```

The syntax for creating a StringBuilder object

```
StringBuilder variableName = new
StringBuilder([value][,][capacity]);
```

Statements that create and initialize StringBuilder objects

```
StringBuilder address1 = new StringBuilder();      // Capacity is 16
StringBuilder address2 = new StringBuilder(10);    // Capacity is 10
StringBuilder phoneNumber1 =
    new StringBuilder("9775551212");               // Capacity is 16
StringBuilder phoneNumber2 =
    new StringBuilder("9775551212", 10);           // Capacity is 10
```

A statement that simplifies references to the StringBuilder class

```
using System.Text;
```

Common properties of the StringBuilder class

```
[index]
Length
Capacity
```

Common methods of the StringBuilder class

```
Append(string)
Insert(index, string)
Remove(startIndex, count)
Replace(oldString, newString)
ToString()
```

Code that creates a phone number and inserts dashes

```
StringBuilder phoneNumber = new StringBuilder(10); // Capacity is 10
phoneNumber.Append("9775551212");                  // Capacity is 10
phoneNumber.Insert(3, ".");                        // Capacity is 20
phoneNumber.Insert(7, ".");                        // 977.555.1212
phoneNumber.Remove(0, 4);                          // 555.1212
phoneNumber.Replace(".", "-");                     // 555-1212
lblPhoneNumber.Text = phoneNumber.ToString();      // 555-1212
```

The syntax of the Format method of the String class

```
Format(string, value1[, value2]...)
```

The syntax of a format specification within the string argument

```
{N[, M][:formatString]}
```

Explanation

N	An integer that indicates the value to be formatted.
M	The width of the formatted value as an integer. If M is negative, the value will be left-justified. If it's positive, it will be right-justified.
formatString	A string of formatting codes.

The syntax of a format string

```
positiveformat[;negativeformat[;zeroformat]]
```

Standard numeric formatting codes

Code	Description
C or **c**	Formats the number as currency with the specified number of decimal places.
D or **d**	Formats an integer with the specified number of digits.
E or **e**	Formats the number in scientific (exponential) notation with the specified number of decimal places.
F or **f**	Formats the number as a decimal with the specified number of decimal places.
G or **g**	Formats the number as a decimal or in scientific notation depending on which is more compact.
N or **n**	Formats the number with thousands separators and the specified number of decimal places.
P or **p**	Formats the number as a percent with the specified number of decimal places.

Custom numeric formatting codes

Code	Description
`0`	Zero placeholder
`#`	Digit placeholder
`.`	Decimal point
`,`	Thousands separator
`%`	Percentage placeholder
`;`	Section separator

Statements that format a single number

```
string balance1 = String.Format("{0:c}", 1234.56);      // $1,234.56
string balance2 = String.Format(
    "{0:$#,##0.00;($#,##0.00)}", -1234.56);      // ($1,234.56)
string balance3 =
    String.Format("{0:$#,##0.00;($#,##0.00);Zero}", 0); // Zero
string quantity = String.Format("{0:d3}", 43);           // 043
string payment =  String.Format("{0:f2}", 432.8175);     // 432.82
```

A statement that formats two numbers

```
string totalDue = String.Format(
    "Invoice total: {0:c}; Amount due: {1:c}.", 354.75, 20);
// Invoice total: $354.75; Amount due: $20.00.
```

Standard DateTime formatting codes

Code	Description
`d`	Short date
`D`	Long date
`t`	Short time
`T`	Long time
`f`	Long date, short time
`F`	Long date, long time
`g`	Short date, short time
`G`	Short date, long time

Custom DateTime formatting codes

Code	Description
`d`	Day of the month without leading zeros
`dd`	Day of the month with leading zeros
`ddd`	Abbreviated day name
`dddd`	Full day name
`M`	Month without leading zeros
`MM`	Month with leading zeros
`MMM`	Abbreviated month name
`MMMM`	Full month name
`y`	Two-digit year without leading zero
`yy`	Two-digit year with leading zero
`yyyy`	Four-digit year
`/`	Date separator

Custom DateTime formatting codes (cont.)

Code	Description
`h`	Hour without leading zeros
`hh`	Hour with leading zeros
`H`	Hour on a 24-hour clock without leading zeros
`HH`	Hour on a 24-hour clock with leading zeros
`m`	Minutes without leading zeros
`mm`	Minutes with leading zeros
`s`	Seconds without leading zeros
`ss`	Seconds with leading zeros
`f`	Fractions of seconds (one *f* for each decimal place)
`t`	First character of AM/PM designator
`tt`	Full AM/PM designator
`:`	Time separator

Statements that format dates and times

```
DateTime currentDate = DateTime.Now;          // 1/30/2016 10:37:32 PM
String.Format("{0:d}", currentDate)           // 1/30/2016
String.Format("{0:D}", currentDate)    // Saturday, January 30, 2016
String.Format("{0:t}", currentDate)           // 10:37 PM
String.Format("{0:T}", currentDate)           // 10:37:32 PM
String.Format("{0:ddd, MMM d, yyyy}",
    currentDate)                              // Sat, Jan 30, 2016
String.Format("{0:M/d/yy}", currentDate)      // 1/30/16
String.Format("{0:HH:mm:ss}", currentDate)    // 22:37:32
```

The basic syntax of an interpolated string

```
$string
```

The syntax of an interpolated expression within the string

```
{value[, M][:formatString]}}
```

Explanation

`value`	The value to be formatted
`M`	An integer that indicates the width of the formatted value. If M is negative, the value will be left-justified. If it's positive, it will be right-justified.
`formatString`	A string of numeric or DateTime formatting codes.

Statements that format numbers

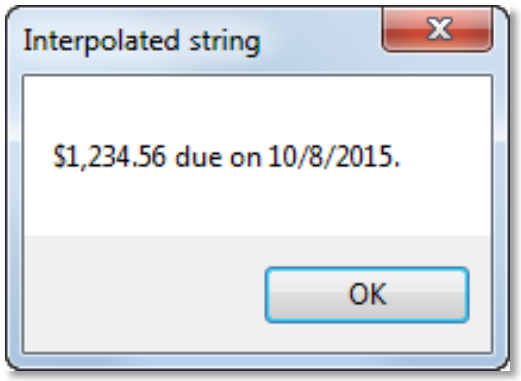
```
string balance1 = $"{1234.56:c}";           // $1,234.56
string balance2 =
    $"{-1234.56:$#,##0.00;($#,##0.00)}";    // ($1,234.56)
```

A statement that formats a date

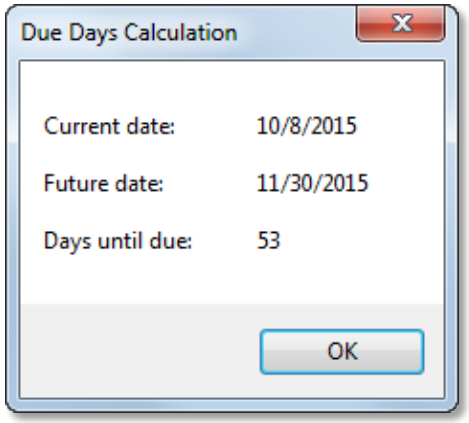
```
string date1 = $"{DateTime.Now:D}";
    // Thursday, October 08, 2015
```

Code that formats a number and a date

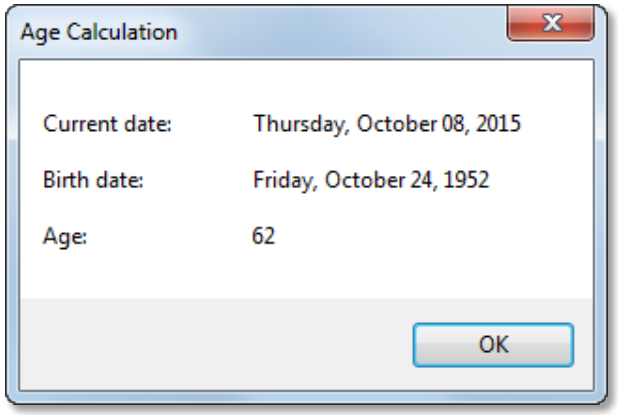
```
decimal balanceDue = 1234.56m;
DateTime dateDue = DateTime.Now;
MessageBox.Show($"{balanceDue:c} due on {dateDue:d}.",
    "Interpolated string");
```



Exercise 9-1 Work with dates and times

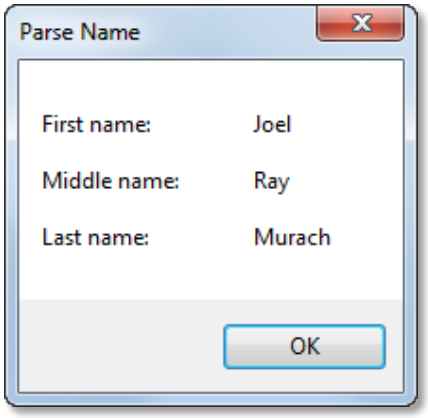


Calculate the due days for a given future date.

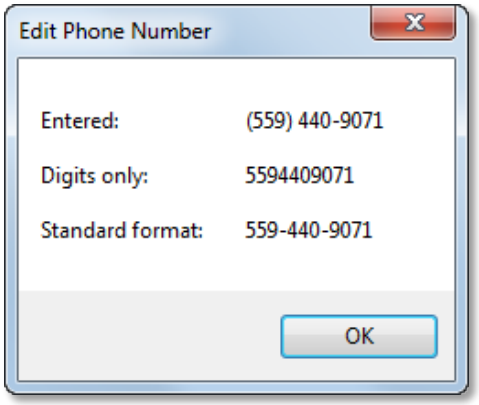


Calculate the age for a given a birth date.

Exercise 9-2 Work with strings

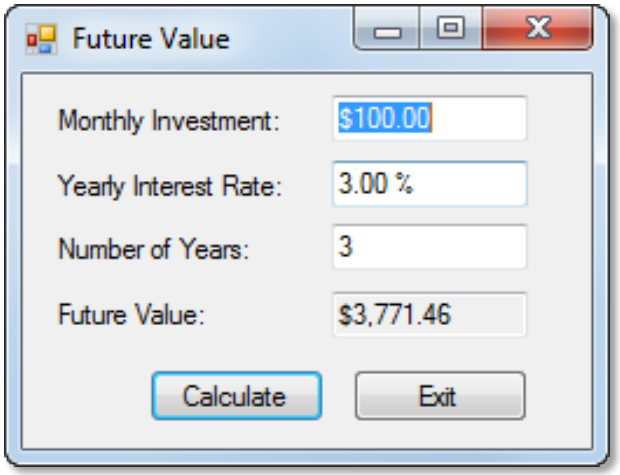


Parse a name into first, middle, and last names.



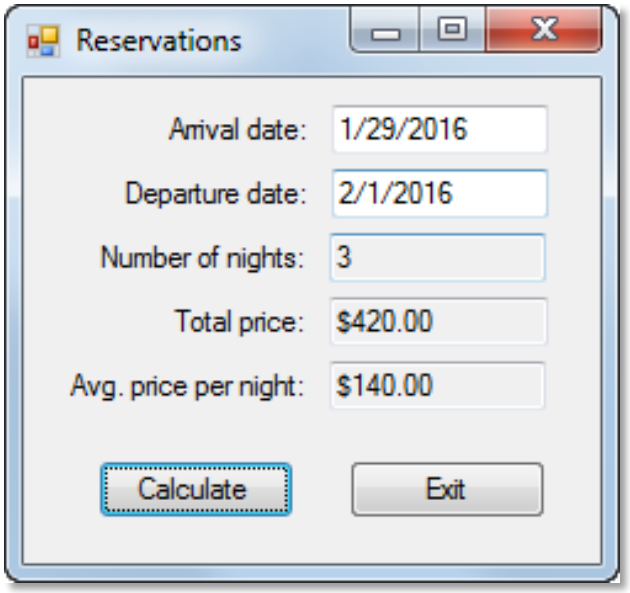
Edit a phone number so it uses a standard format.

Exercise 9-3 Enhance the Future Value application



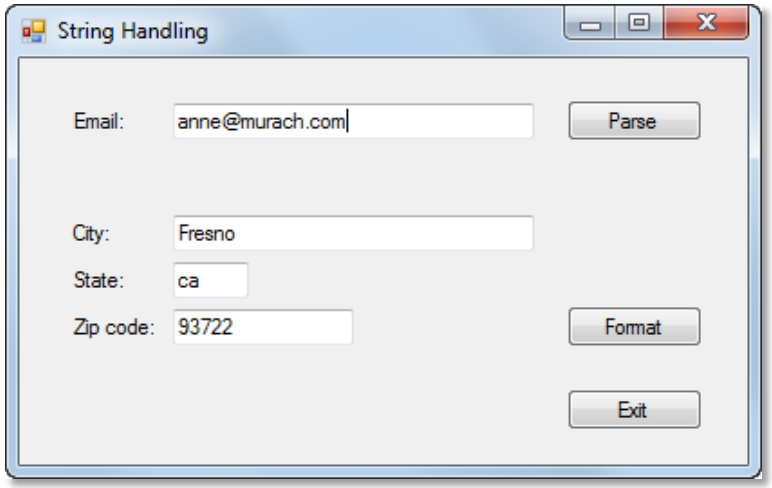
Use the Parse method to validate numeric entries.

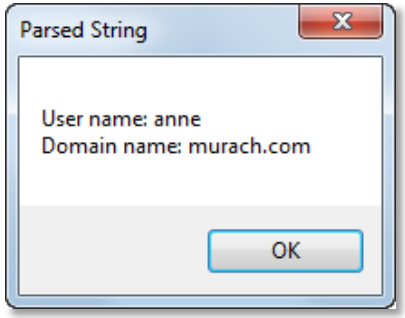
Extra 9-1 Calculate reservation totals

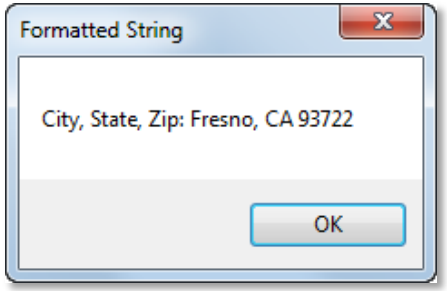


Calculate the Number of nights, total price, and average price for a reservation based on the arrival and departure dates.

Extra 9-2 Work with strings

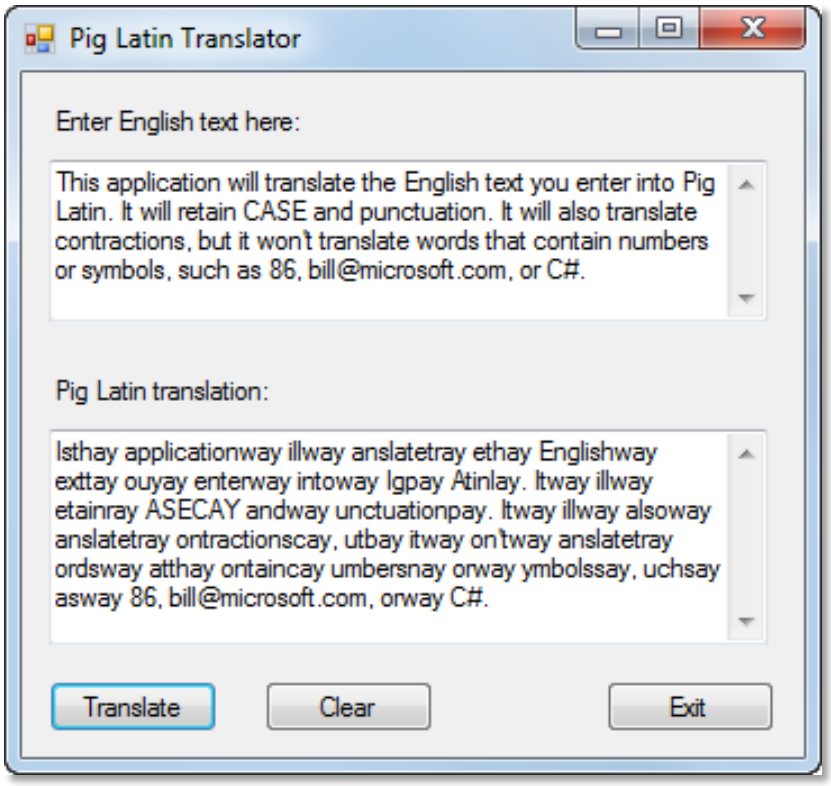






Parse an email address and format the city, state, and zip code portion of an address.

Project 2-1 Translate English to Pig Latin



Create a form that accepts a text entry and converts it to Pig Latin.