

CGT 456 Lecture 2

Programming Basics

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

1

Resources

- MSDN Library:
 - <http://msdn.microsoft.com/en-us/library>

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

2

Visual Programming Concepts

- Controls
 - Placed on the form
 - Every item you place on the form is called a **Control**
 - Likewise, because you are placing an item, a thing, on the form, it is called an **Object**
 - Each control, or object, has methods and properties
 - Methods perform actions
 - Properties contain data about the object

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

3

Variable

- An identifier (usually a letter, word, or phrase) that is linked to a value stored in the system's memory or an expression that can be evaluated
- A symbolic name associated with a value and whose associated value may be changed
- Example:
 - numResults is a variable that contains a number

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

4

Naming Conventions

- Variable names shall describe their use, shall be in mixed case with an initial upper case character for each word, and shall have 'Hungarian' prefixes that describe their type and scope. Abbreviations shall be avoided, but where they are used they are to be used consistently throughout the entire application
- Hungarian prefixes not only identify the type of a variable but also have the advantage of avoiding confusion between variables and functions, methods or properties

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

5

Naming Conventions (cont.)

- Example names:
 - lblAddress (a Label control)
 - tbAddress (a TextBox control)
 - pnlBusinessCard (a Panel control)
 - frmMain (a Form control)
 - picBackground (a PictureBox control)
 - btnResetGame (a Button control)

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

6

Naming Conventions (cont.)


- When confronted with the need for a new name in a program, a good programmer will generally consider the following factors to reach a decision:
 - Mnemonic value—so that the programmer can remember the name.
 - Suggestive value—so that others can read the code.
 - "Consistency"—this is often viewed as an aesthetic idea, yet it also has to do with the information efficiency of the program text. Roughly speaking, we want similar names for similar quantities.
 - Speed of the decision—we cannot spend too much time pondering the name of a single quantity, nor is there time for typing and editing extremely long variable names.


1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

7

Naming Conventions (cont.)

- Camel Casing 
 - use camel casing (`documentFormatType`) for variable names, where the first letter of each word except the first is capitalized.

- Pascal Casing 
 - use Pascal casing (`CalculateInvoiceTotal`) for routine names (method names, function names) where the first letter of each word is capitalized.

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

8

Comments

- The best programmers also document their work well.
- The easiest programs to read are those that are well commented.
- The purpose of a comment is to explain the code to a person who is reading it.
- Comments are important to a programmer, but the program itself ignores them.

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

9

Comments

- A single line comment begins with `//`
`//this is a comment`
- Everything that follows the `//` on that one line is a comment.
- It is common to place comments after short statements:
`int counter; //count the number of records`

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

10

Comments

- A multi-line comment begins with `/*` and ends with `*/`

```
/* this is a multi-line comment
you can write as much as you want.
you can comment out an entire program.
then end the comment with */
```

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

11

Bookend Comments

- ```
////////////////////////////////////
// GenerateInventory Function
////////////////////////////////////
public void GenerateInventory()
{
 //do something
}
////////////////////////////////////
// End GenerateInventory Function
////////////////////////////////////
```

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

12

## Ending Comments

```
□ public void GenerateInventory()
 {
 while(...)
 {
 if(...)
 {
 ...
 } //end if
 } //end while
 } //end GenerateInventory
```

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

13

## Good Programming Practice 3.6

- Following the closing right brace of a method body or class declaration with a comment indicating the method or class declaration to which the brace belongs improves application readability.

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

14

## When to use comments

- Beginning of a program
  - Describe what the program is and what it does
  - Include the author of the program and the date
    - You might include the date of original authorship along with modification dates – especially include the last modification date
- Within the program
  - Our rule of thumb is: one line of comment for every line of code

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

15

## Declaration

- To define the name and data type of a variable or other programming construct.
- Declaring a variable is the process of allocating space in memory to store some data in.
- Variables must be declared before you can use them.
  - This restriction is called **strictly typed**

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

16

## Data Types

- There are several simple types that can be used to declare variables. Here are some of the most common:
  - int
  - string
  - bool
  - decimal
  - float
  - double
  - byte
  - long
  - short
  - char
  - We'll discuss declaring objects later...

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

17

## Declaration (cont.)

```
■ int myInt; //declares an integer; precision: 32-bit
□ -2,147,483,648 to 2,147,483,647
■ short myShortInt; //an integer; precision: 16-bit
□ -32,768 to 32,767
■ long myLongInt; //an integer; precision: 64-bit
□ -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
■ byte myByte; //declares an integer byte; precision: 0 to 255
■ string sName; //declares a string (text)
■ char chrLetter; //declares a character - holds a single letter
■ bool flag; //declares a boolean (true or false)
■ float fProduct; //floating point/decimal number; precision: 7 digits
■ double dblMyNum; //a decimal number; precision: 15-16 digits
■ decimal decGrade; //decimal number; precision: 28-29 significant digits
```

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

18

## Good Programming Practice 3.8

- Declare each variable on a separate line.
- This format allows a comment to be easily inserted next to each declaration.

```
■ int numResults; //total number of results
```

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

19

## Assignment Operator (=)

- Assigns the value on the right into the location on the left
- lblAddress.Text = "401 N. Grant St.";
  - The two pieces of information above are known as operands.... operand 1 = operand 2

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

20

## Assignment Operator (=)

- `int counter; //declares a variable named counter`
- `counter = 0; //initializes the variable, counter, to be equal to zero`
- `counter = counter + 1; // adds 1 to counter. counter now equals 1`
- `counter = counter + 1; // adds 1 to counter. counter now equals 2`
- `counter = counter + 1; // adds 1 to counter. counter now equals 3`

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

21

## Initialization (and assignment)

- To assign a starting value to a variable.
  - `myInt = 1000000000;`
  - `myShortInt = 10000;`
  - `myLongInt = 1000000000000;`
  - `myByte = 210;`
  - `sName = "Ronald J. Glotzbach";`
  - `chrLetter = 'a'; //note the use of single quotes`
  - `flag = true;`
  - `fProduct = 3.5F; //the f/F is needed to specify float`
  - `dblMyNum = 7.6; //for int: dblMyNum = 3D;`
  - `decGrade = 300.5m; //the m/M is needed to specify decimal`

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

22

## Good Programming Practice

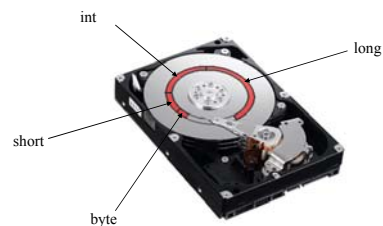
- Always line up your *equal signs*, as demonstrated on the previous slide.
- Always line up your *comments*, as demonstrated on the previous slide.

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

23

## Storage Size / Memory Allocation



1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

24

## Dot operator (dot notation)

- The dot operator is a period (.) that appears between two words or phrases.
- It can often be read as: the item on the right 'belongs to' the item on the left
- For example
  - `tbAddress.Text` is accessing the `Text` property of the object `tbAddress`
    - `Text` belongs to the `tbAddress` object

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

25

## Initializing form values

- `lblCardName.Text = "";`
- `lblCardTitle.Text = "";`
- `lblCardAddress.Text = "";`
  
- Or
- `lblCardName.Text = "Ronald J. Glotzbach";`

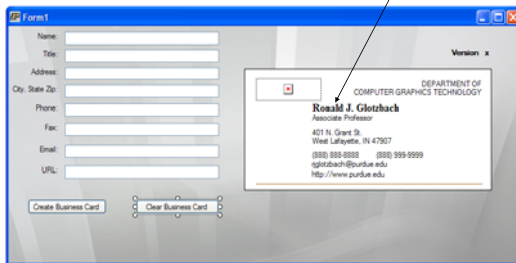
1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

26

## Initializing form values

`lblCardName.Text = "Ronald J. Glotzbach";`



1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

27

## Numeric Literal

- `myInt = 1000000000;`
- `myByte = 210;`
- `dblMyNum = 7.6;`
  
- Any number that you, yourself, write by hand is called a **numeric literal**.
- This is also considered 'hard coding' a number, because the value is not generated dynamically

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

28

## Memory Locations

```
int number1;
int number2;
int sum;

number1=45;
number2=72;

sum = number1+number2;
```

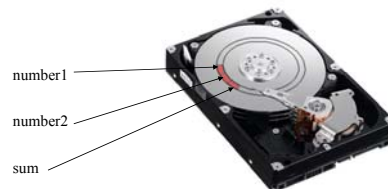
Fig. 3.19 | Memory location showing the name and value of variable `number1`.

Fig. 3.20 | Memory locations after storing values for `number1` and `number2`.

Fig. 3.21 | Memory locations after calculating and storing the sum of `number1` and `number2`.

1/12/2010

## Memory Allocation



1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

30

## Text Literal

- ❑ lblAddress.Text = "401 N. Grant St.";
- ❑ tbName.Text = "Ronald J. Glotzbach";
- ❑ pictureBox1.ImageLocation = "kitten.jpg";
  
- ❑ Any text that you, yourself, write by hand and put in quotation marks is called a **text literal**.
- ❑ This is also considered 'hard coding' text because the value is not generated dynamically

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

31

## Empty String

- ❑ ""
- ❑ When initializing strings or emptying strings, you often initialize using the **empty string**
  - lblAddress.Text = "";
  - title = "";
  - phone = "";
- ❑ This removes any text from the property/variable so that it is *empty*
- ❑ The empty string is a text literal

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

32

## Clearing a TextBox

- ❑ A **TextBox** is special in that there are two ways to clear it. Both ways perform the same action.
- ❑ TextBox myTextBox;  
myTextBox.Text = "something";
- ❑ *The first way:*
  - ❑ myTextBox.Text = ""; //empty string
- ❑ *The second way:*
  - ❑ myTextBox.Clear(); //call the clear method

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

33

## Concatenation

- ❑ Concatenation is performed using the +
- ❑ string greeting = ""
- ❑ greeting = "hello." + "how are you?";
  - greeting now contains: "hello.how are you?"
- ❑ string part1 = "Hey!";
- ❑ string part2 = "How are you doing?";
- ❑ string space = " ";
- ❑ greeting = part1 + space + part2;
  - greeting now contains: "Hey! How are you doing?"

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

34

## ToString()

- ❑ Access the actual text value of an object, property or variable by adding .ToString() to the end of it.
  - string address;
  - string city;
  
  - address = tbAddress.Text.ToString();
  - city = tbCity.Text.ToString();
  
  - lblCardAddress.Text = address.ToString();
  - lblCardCity.Text = city.ToString();

1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

35

## Event Handlers

- ❑ When you click someplace on the form, an event happens.
- ❑ These events are handled by **event handlers**

```
private void pictureBox1_Click(object sender, EventArgs e)
{
 //do something
}
```

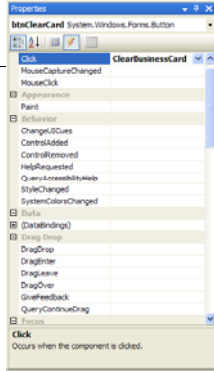
1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

36

## Event Handlers (cont.)

- An event handler can be added through the visual interface.
- Make sure the lightning bolt is selected in the Properties box.
- Simply double click in the white space to the right of the word Click.



1/12/2010

CGT 456  
Copyright © 2009 Ronald J. Glotzbach

37

## Event Handlers (cont.)

- ```
private void ClearBusinessCard(object sender, EventArgs e)
{
    //do something
}
```
- All event handlers will have (**object sender, EventArgs e**)
 - **object sender** refers to the actual object (control) on the form that triggered the event
 - **EventArgs e** refers to any arguments that were passed from the control when the event was triggered
 - We will learn more about these later...

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

38

Arithmetic Operators

Operation	Arithmetic Operator	C# Expression
Addition	+	f + 7
Subtraction	-	p - c
Multiplication	*	b * m
Division	/	x / y
Remainder	%	r % s

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

39

Order of operation

Operators	Operations	Order of evaluation
<i>Evaluated First</i>		
*	Multiplication	If there are several operators of this type, they are evaluated from left to right.
/	Division	
%	Remainder	
<i>Evaluated Next</i>		
+	Addition	If there are several operators of this type, they are evaluated from left to right.
-	Subtraction	

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

40

Proper use of parentheses

- To make sure arithmetic operations execute in the order you intend them to, use parentheses
 - a * (b + c)
 - ((a + b) * c)

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

41

Relational operators

Standard algebraic equality and relational operators	C# equality or relational operator	Sample C# condition	Meaning of C# condition
<i>Equality operators</i>			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
<i>Relational operators</i>			
>	>	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

Precedence and Associativity

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

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

43

Increment & Decrement Operators

- Pre-increment
 - ++a;
- Post-increment
 - a++;
- Pre-decrement
 - --b;
- Post-decrement
 - b--;
- Increment **a** by 1 then use the new value of **a** in the expression in which **a** resides.
- Use the current value of **a** in the expression in which **a** resides, then increment **a** by 1.
- Decrement **b** by 1 then use the new value of **b** in the expression in which **b** resides.
- Use the current value of **b** in the expression in which **b** resides, then decrement **b** by 1.

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

44

Post-increment example

- int versionNum;
- versionNum = 0;

- versionNum++;
is the same as
versionNum = versionNum + 1;

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

45

Post-decrement example

- int versionNum;
- versionNum = 10;

- versionNum--;
is the same as
versionNum = versionNum - 1;

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

46

Important Note

- In C#
 - If you use ++ or -- alone on a line like this:
 - a++;
 - You will **not** see a difference between a++ or ++a
 - This applies to for loops as well (pre and post act the same)
 - However, if you use ++ or -- in an equation like:
 - a = x++;
 - a = ++x;
 - Then you will see a difference between the two.
 - *Show Increment/Decrement application*

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

47

Increment & Decrement Operators

- Be mindful of where you use pre/post - increment/decrement operators. Using one in the wrong location can have unintended consequences.

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

48

Escape Sequences

- The backslash `\` is called an *escape character*
- It indicates to the program that there is a special character in the string.
- When a `\` appears, it combines with the next character to form an *escape sequence*
 - `\r` carriage return (needed for Windows Forms)
 - `\n` new line (needed for console applications)
 - `\t` Horizontal tab
 - `\\` backslash (inserts a backslash into the string)
 - `\"` double quote

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

49

Escape Sequences

```
tbAddress.Text = "401 N. Grant \r Knoy 319 \r";
```

In the textbox, prints:

```
401 N. Grant  
Knoy 319
```

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

50

Escape Sequences

- You might want to ensure that your program works correctly in both a console app and a windows form... so you could add both a `\r` and a `\n`

```
tbAddress.Text = "401 N. Grant \r\n Knoy 319 \r\n";
```

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

51

Escaping a drive path

- `string fileLocation = "";`
- `fileLocation = "C:\\inetpub\\wwwroot\\";`
- Produces:
 - `C:\inetpub\wwwroot\`

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

52

Convert class

- A number stored in a textbox is stored as text, not as a number. Therefore, you must convert the text to a number before storing it into a strongly typed variable.

```
int versionNum;  
versionNum = Convert.ToInt32(tbVersionNum.Text.ToString());
```

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

53

Convert class

- The Convert class holds many types of conversions to help you change types. To list a few:
 - `ToInt16()`
 - `ToInt32()`
 - `ToInt64()`
 - `ToBoolean()`
 - `ToByte()`
 - `ToChar()`
 - `ToDecimal()`
 - `ToDouble()`
 - `ToDateTime()`
 - ...and several more...

1/12/2010

CGT 456
Copyright © 2009 Ronald J. Glotzbach

54