

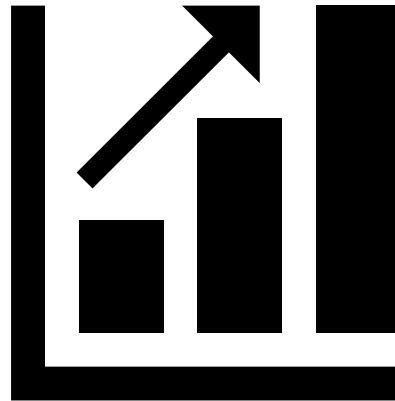
CHAPTER 2: VARIABLES AND TYPES

Fall 2019 – CSC 180 – Introduction to Programming



KEEP UP!

- Make sure to keep up, we are building up in this course
- If you missed a class meeting, make sure to review before you come to class!



2.1 MORE WRITING

- You can put as many **statements** as you want in Main;
 - to print more than one line ... one could use two statements ...
- You need to document your code – use **meaningful comments**
 - you can put comments at the end of a line OR on a line by themselves
 - the comments should show your intent – why did you write that line?

- The phrases that appear in double quotation marks are called **strings**, because they are made up of a sequence (string) of characters.

- **WriteLine** vs **Write** ...

- Spaces that appear outside of quotation marks generally do not affect the behavior of the program.

```
1 using System;
2
3 namespace ThinkSharp {
4     public class Program {
5         // Generates some simple output.
6         public static void Main(string[] args) {
7             Console.WriteLine("Hello, world."); // print one line
8             Console.WriteLine("How are you?"); // print another
9         }
10    }
11 }
```

```
1 using System; namespace HelloWorld { public class Program { public static
2 void Main(string[] args) { Console.Write("Goodbye, "); Console.WriteLine
3 ("cruel world!"); }}}}
```

COMMENTS

- C# ignores them when compiling your code
- **Single line comments**: start with `//` and continue until the end of the line
- **Block-comments**: includes everything between `/*` and the first matching `*/`
- **Your code should contain comments.**
 - After the second week of classes I will automatically assign a 0 to any code that contains no meaningful comments and/or does not compile and/or crashes at start.



2.2 VARIABLES

- A **variable** is a named location that stores a **value**.
- **Values** are things that can be written, stored and operated on.
 - For example: “Hello, world.” is a string value
- To **store** a value, you have to **create a variable**.
 - To store a **string** value, one could **declare a string variable**:
 - **bob** is the name of the variable (so we can refer back to it)
 - **string** is the type of the variable (type determines what kind of values one can store in that variable)
 - To store an **whole number**, one could **declare an int variable**
- Note: bob is not a **meaningful name**. Please use variable names that show intent for those variables
 - And please use **camelCase** for variables

```
string bob;
```

```
int bob;
```

```
string firstName;  
string lastName;  
int hour, minute;
```



IDENTIFIERS

- In C# an **identifier** is a **name that the programmer chooses** for something in the program.
 - The **name of a variable** is more properly called an **identifier**.
 - We will see other places where we create identifiers (for example, method names).
- C# has some **rules about what constitutes a valid identifier**:
 - All identifiers names must start with a letter or an underscore (_).
 - After the letter you can have either letters or numbers or the underscore "_" character.
 - An identifier for a variable should not be one of the **keywords** that Visual C# reserves for its own use.

```
float averageIceCreamPriceInPence;
```
- Upper and lower case letters are different, i.e. **Fred** and **fred** are different identifiers
- The convention in C# for the kind of variables that we are creating at the moment is to mix upper and lower case letters so that each word (except first) in the identifier starts with a capital (**camel case**)
 - C# is **Case Sensitive**! That means **Car**, **car**, and **CAR** are different.



KEYWORDS

- **Keywords** (sometimes called **reserved words**) are reserved for use by C# and are always spelled with all *lowercase letters*. Because they're reserved, they **can't be used as identifiers**.
 - See <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/index>
 - And http://www.infocodify.com/csharp/csharp_identifiers

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

EXERCISES

Variable Name	Legal or Illegal?	Why?
dayOfWeek	Legal	
3dGraph	Illegal	
june1997	Legal	
mixture#3	Illegal	
week day	Illegal	
class	Illegal	
case	Illegal	
_case	Legal	



2.3 ASSIGNMENT

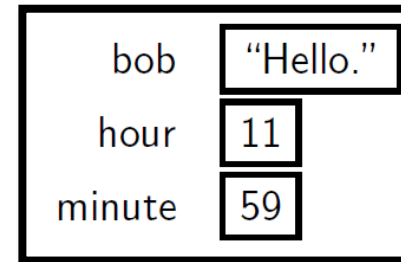
- To store values we use the **assignment statement**.

- The left side of an assignment has to be a variable name, not an expression. The left side indicates the storage location where the result will go.

```
bob = "Hello.";           // give bob the value "Hello."  
hour = 11;                // assign the value 11 to hour  
minute = 59;              // set minute to 59
```

- It is very important to note:

- When you **declare** a variable, you **create a named storage location**.
- When you **make an assignment** to a variable, **you give it a value**.



- A variable has to **have the same type** as the value you assign it.

```
bob = "123";              // legal  
bob = 123;                 // not legal
```

- What do the following display?

```
Console.WriteLine("firstLine");
```

```
string firstLine;  
firstLine = "Hello, again!";  
Console.Write("The value of firstLine is ");  
Console.WriteLine(firstLine);
```

```
int hour, minute;  
hour = 11;  
minute = 59;  
Console.Write("The current time is ");  
Console.Write(hour);  
Console.Write(":");  
Console.Write(minute);  
Console.WriteLine(".");
```



2.6 OPERATORS

- **Operators** are symbols used to represent **computations** like addition and multiplication.

- **Expressions** can contain both **variable names** and **numbers (literals)**.
- **Variables** are replaced with their **values** before the computation is performed.

- Examples: `1 + 1` `hour - 1` `hour * 60 + minute` `minute / 60`

- Beware if the **integer division**!

- When **both operands** are **integers** (operands are the things operators operate on), **the result is also an integer**, and by convention integer division always rounds down.

```
int hour, minute;
hour = 11;
minute = 59;
Console.WriteLine("Number of minutes since midnight: ");
Console.WriteLine(hour * 60 + minute);
Console.WriteLine("Fraction of the hour that has passed: ");
Console.WriteLine(minute / 60);
```

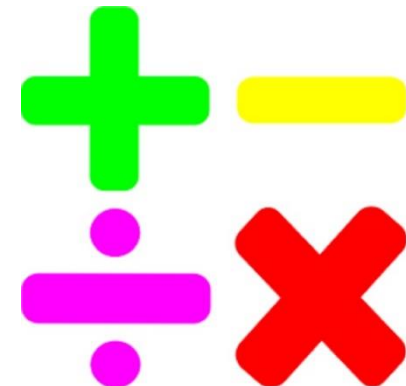
- Another important operator is **% (modulus operator)**.

- Examples: `9%5 = ?` `5%9 = ?`
- Important example: how can one check if a number is even?



2.7 ORDER OF OPERATIONS

- When more than one operator appears in an expression, the **order of evaluation** depends on the **rules of precedence**.
 - **Multiplication** and **division** happen before **addition** and **subtraction**.
 - $5 + 3 * 2 = ?$
 - $2 / 3 - 1 = ?$
 - If the operators have the **same precedence** they are **evaluated from left to right**.
 - $\text{minute} * 100 / 60 = ?$ ← assume minutes = 30
 - Any time you want to **override the rules of precedence** (or you are not sure what they are) you can **use parentheses**. Expressions in parentheses are evaluated first.
 - $2 * (3 - 1) = ?$



OPERATORS FOR STRINGS, COMPOSITION

- you cannot perform mathematical operations on strings, even if the strings look like numbers.
 - The following are illegal (if we know that bob has type string):
bob - 1 "Hello" / 123 bob * "Hello"
- the **+** operator does work with strings, it represents **concatenation**, which means joining up the two operands by linking them end-to-end.
 - "Hello, " + "world." yields the string "Hello, world."
 - bob + "ism" adds the suffix ism to the end of whatever bob is
- One of the most useful features of programming languages is their ability to take small building blocks and **compose** them.
 - For example,

```
Console.WriteLine(17 * 3);
```

```
Console.WriteLine(hour * 60 + minute);
```



WHAT ARE DATA TYPES?

- A **variable** holds data of a specific type. When you **declare** a variable to store data in an application, you need to choose an appropriate data type for that data.
- Visual C# is a type-safe language, which means that the compiler guarantees that values stored in variables are always of the appropriate type.
 - **int** – whole numbers
 - **long** – whole numbers (bigger range)
 - **float** – number with a decimal point
 - **double** – number with a decimal point
 - **decimal** – monetary values
 - **char** – single character (use single quotes ' ')
 - **bool** – either **true** or **false**
 - **DateTime** – moments in time
 - **string** – sequence of characters (use double " ")

Type	Examples
<i>int</i>	<ul style="list-style-type: none">• 576• 20140214• -873• 0
<i>bool</i>	<ul style="list-style-type: none">• true• false
<i>double</i>	<ul style="list-style-type: none">• 7.51954• 985.0• -33.614
<i>char</i>	<ul style="list-style-type: none">• '8'• 'z'• ','• ' '
<i>string</i>	<ul style="list-style-type: none">• "Are you excited?"• "z"• ""• " "

PRIMITIVE TYPES - SKIP

Type	Description	Size (bytes)	Range
int	Whole numbers	4	−2,147,483,648 to 2,147,483,647
long	Whole numbers (bigger range)	8	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	Floating-point numbers	4	+/−3.4 × 10 ³⁸
double	Double precision (more accurate) floating-point numbers	8	+/−1.7 × 10 ³⁰⁸
decimal	Monetary values	16	28 significant figures
char	Single character	2	N/A
bool	Boolean	1	True or false
DateTime	Moments in time	8	0:00:00 on 01/01/2001 to 23:59:59 on 12/31/9999
string	Sequence of characters	2 per character	N/A

TYPES (EXAMPLES)

- When we program, we have to take real-world ideas like miles, or money, or names and student numbers, and figure out how best to work with these in C#. Which C# primitive type would be best to work with:

1. Your name? _____
2. Your age in years? _____
3. Whether you are alive? _____
4. Your grade (A, B, C, D, or E) for an assignment? _____
5. The length of a line in centimetres? _____
6. The number of pages in a book? _____
7. The exact price of a loaf of bread? _____
8. Whether you have eaten anything in the last 6 hours? _____
9. The second letter of your first name? _____
10. The number of grains of sand in a jar? _____
11. The square root of 2? _____
12. Your reasons for reading this book? _____
13. The colour of your eyes? _____

STORING INTEGER VALUES - SKIP

- C# provides a variety of integer types, depending on the range of values you would like to store:

- Declaring a variable of type int:

```
int numberOfSheep;
```

- An integer literal is expressed as a sequence of digits with no decimal Point:

```
23
numberOfSheep = 23 ;
sbyte tinyVal = 127;
```

- Since the maximum value that an sbyte can hold is 127 the following will not compile:

```
sbyte tinyVal = 128;
```

sbyte	8 bits	-128 to 127
byte	8 bits	0 to 255
short	16 bits	-32,768 to 32,767
ushort	16 bits	0 to 65,535
int	32 bits	-2,147,483,648 to 2,147,483,647
uint	32 bits	0 to 4,294,967,295
long	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	64 bits	0 to 18,446,744,073,709,551,615
char	16 bits	0 to 65,535



STORING REAL VALUES - SKIP

- "Real" is a generic term for numbers which are not integers. They have a decimal point and a fractional part.
- A standard **float** value has a range of 1.5E-45 to 3.4E48 with a precision of only 7 digits (i.e. not as good as most pocket calculators).
- If you want more precision you can use a **double** instead (double is an abbreviation for double precision). This takes up more computer memory (and make your program slower) but it has a range of 5.0E-324 to 1.7E308 and a precision of 15 digits.
- **Declaring** variables:

```
float averageIceCreamPriceInPence;  
double univWidthInInches;
```
- Real **literal** values
 - 2.5 //double by default
 - 2.5f
 - 9.4605284E15



STORING TEXT - SKIP

Character	Escape Sequence name
\'	Single quote
\"	Double quote
\\	Backslash
\0	Null
\a	Alert
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical quote

- A **char** is a type of variable which can hold a single character.
 - Declaring char variables: `char commandKey;`
 - char literal values `'A'`
 - Make sure to use single quotes!!!
 - Character **Escape Sequences** (used to represent certain special characters ...)
 - C# uses a character set called **Unicode** which can handle over 65,000 different character designs including a wide range of foreign characters. These characters are stored as numbers. Hence once can also represent characters using hexadecimal (numbers base 16): `char capitalA = '\x0041' ;`
- A **string** is a type that can hold a string of text (from a short one, such as "SMU", to a long one, for example one that stores the entire text from a book)
 - Declaring string variables: `string commandLine;`
 - string literal values `"this is a string"` `"\x0041BCDE\a"` `@"\x0041BCDE\a"`
 - Make sure to use double quotes!!!
 - Note: @ can be used for **verbatim** string.
 - Note: what string literal can you use to represent the following path: C:\Windows\assembly ?
 - Note: Make sure to make distinction between `"A"` and `'A'`!!!! What does `"A\\B\nC//D"` represent?



STORING STATE USING BOOLEANS - SKIP

- A **bool** (short for boolean) is a type of variable which whether or not something is true.
 - Declaring char variables: `bool networkOK;`
 - bool literal values `networkOK = true ;`



EXERCISES

- There is something wrong with each of these lines of code. What is it? → → → → → → →

11. `INT CHEMISTRY = 100;`

12. `bool able, able;`

13. `int articulate = 8; bool ampersand = articulate;`

14. `string o'brien = "Connor";`

1. `in a;`

2. `string a b;`

3. `bool a`

4. `int a, string b;`

5. `int 3d;`

6. `int a = 3.0;`

7. `double a, b = 5.0, c = "6.0";`

8. `char zoology = "";`

9. `char biology = "E";`

10. `bool botany = True;`

- What type would we need to fill in to define `x` in each line below? → → → → → → → →

1. _____ `x = "Yes";`

2. _____ `x = 213;`

3. _____ `x = 0.0;`

4. _____ `x = 'y';`

5. _____ `x = "y";`

6. _____ `x = true;`

7. _____ `x = 5 + 3;`

8. _____ `x = "false";`

9. _____ `x = 3.2/5.1;`

10. _____ `x = false;`

11. _____ `x = "20/5";`

DISPLAY TO CONSOLE (OUTPUT)

- **WriteLine** is so called because, after each line it adds a special character, called a **newline**, that moves the cursor to the next line of the display.
 - The next time **WriteLine** is invoked, the new text appears on the next line.
 - To display the output from multiple print statements all on one line, use **Write**:
- **Very important! Note the difference between:**
 - `Console.Write(firstLine);`
 - `Console.Write("firstLine");`

```
1 using System;
2
3 namespace HelloWorld {
4     public class Program {
5         // Generates some simple output.
6         public static void Main(string[] args) {
7             Console.Write("Goodbye, ");
8             Console.WriteLine("cruel world!");
9         }
10    }
11 }
```

```
string firstLine;
firstLine = "Hello, again!";
Console.Write("The value of firstLine is ");
Console.WriteLine(firstLine);
```



CONSOLE INPUT/OUTPUT

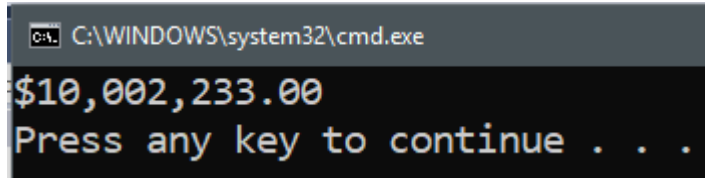
- To **display to the console** use Console.Write and Console.WriteLine methods
 - Console.**Write**("Hello World!"); //displays a string literal
 - Console.**WriteLine**("Your input: {0}",a); //displays a string literal and a variable
 - Console.WriteLine("The two numbers were: {0}, {1}",a, b); //displays a string literal and two variables
 - Console.WriteLine("The two numbers were: {a}, {b}"); //displays a string literal and two variables
 - Console.WriteLine("The two numbers were: " + a + ", " + b); //same as above, but less optimal
- To **read input from the console** (from the user) we can use the following:
 - val = Console.**ReadLine**(); //reads an entire line from the console and saves it into **val**
 - int a = Convert.**ToInt32**(val); //converts a value (from variable **val**) to an integer and saves it into **a**
 - double b = Convert.**ToDouble**(val); //converts a value (from variable **val**) to a double and saves it into **b**
- See also this web resouce: <https://www.programiz.com/csharp-programming/basic-input-output>



FORMAT OUTPUT – SAVE THIS!!!

■ Examples:

```
decimal pay = 10002233;  
Console.WriteLine("{0:c}",pay);
```



```
C:\WINDOWS\system32\cmd.exe  
$10,002,233.00  
Press any key to continue . . .
```

FORMAT SPECIFIER	DESCRIPTION	EXAMPLES	RESULT
C or c	Currency	string s = \$"{2.5:C}";	\$2.50
		string s = \$"{-2.5:C}";	(\$2.50)
D or d	Decimal	string s = \$"{25:D5}";	00025
E or e	Exponential	string s = \$"{250000:E2}";	2.50E+005
F or f	Fixed-point	string s = \$"{2.5:F2}";	2.50
		string s = \$"{2.5:F0}";	3
G or g	General	string s = \$"{2.5:G}";	2.5
N or n	Numeric	string s = \$"{2500000:N}";	2,500,000.00
P or p	Percent	string s = \$"{0.25:P}";	25.00%
R or r	Round-trip	string s = \$"{2.5:R}";	2.5
X or x	Hexadecimal	string s = \$"{250:X}";	FA
		string s = \$"{0xffff:X}";	FFFF

EXERCISES

- **Exercise 1:** Write a small program that prints your name, your major, and the name of the University
 - Use multiple WriteLine statements
- **Exercise 2:** Write a small program that prints your name, your major, and the name of the University
 - Use only one WriteLine statement
- **Exercise 3:** Convert the following “recipe” to C# code. Be sure to declare the appropriate variables:
Store 30 in the **speed** variable.
Store 10 in the **time** variable.
Multiply **speed** by **time** and store the result in a **distance** variable.
Display the value of **distance** variable.
- **Exercise 4:** Write a program that declares the following variables:
a string variable named **name**
an int variable named **age**
a decimal variable named **annualPay**
Store some literals into these variables. Then, the program should display these values on the screen in a manner similar to:
**My name is St Martin, my age is 27, and
I hope to earn \$100000.00 per year.**

Hint: use formatting ...



HOMework FOR CHAPTER 2

- Requirements: see moodle for details
- Deadline: see moodle
- **Reminder: If your code does not compile, crashes at start, or contains no meaningful comments, it will automatically be graded with 0!**

