CS313D: ADVANCED PROGRAMMING

Computer Science department

Lecture 2: Introduction

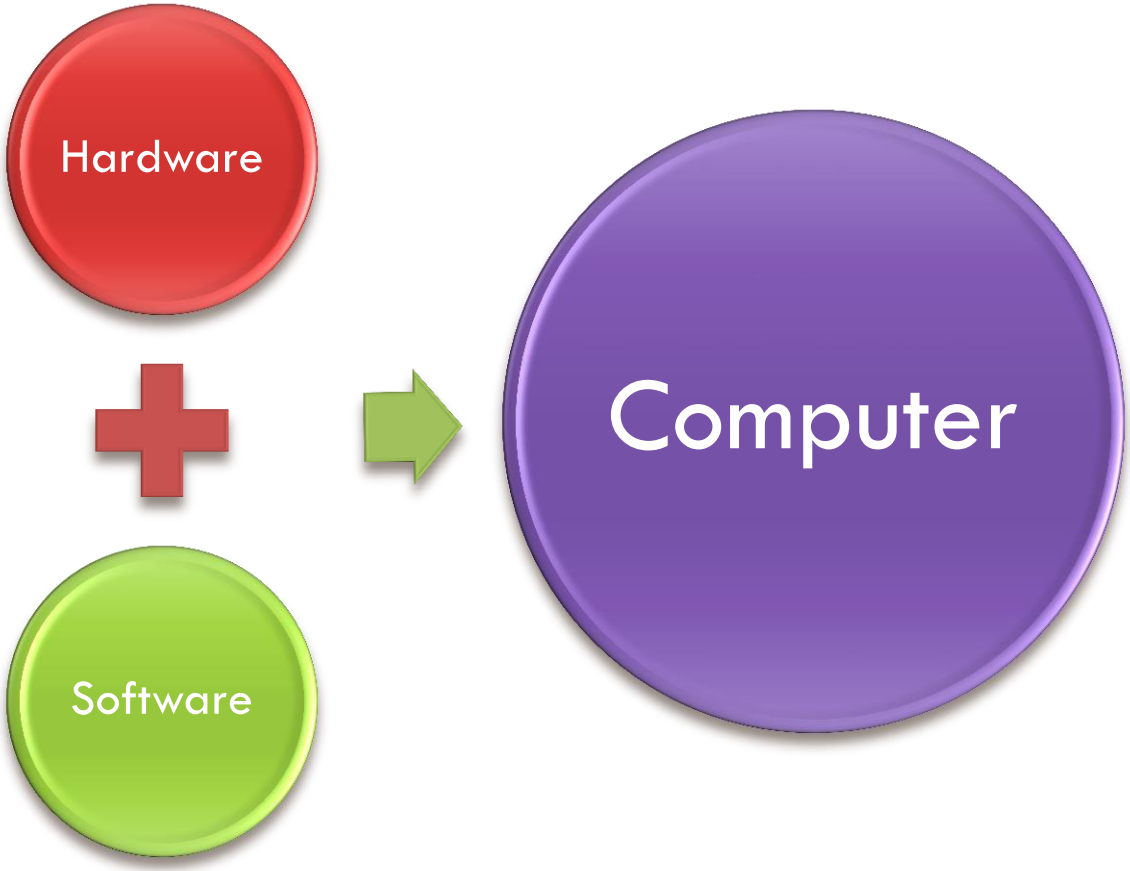
Lecture Contents

- Elements of a Computer system.
- The data hierarchy
- Evolution of programming languages
- OOP
- The C# language
  - First program
  - Variables and constants
  - Input/output
  - Expressions and casting

Introduction

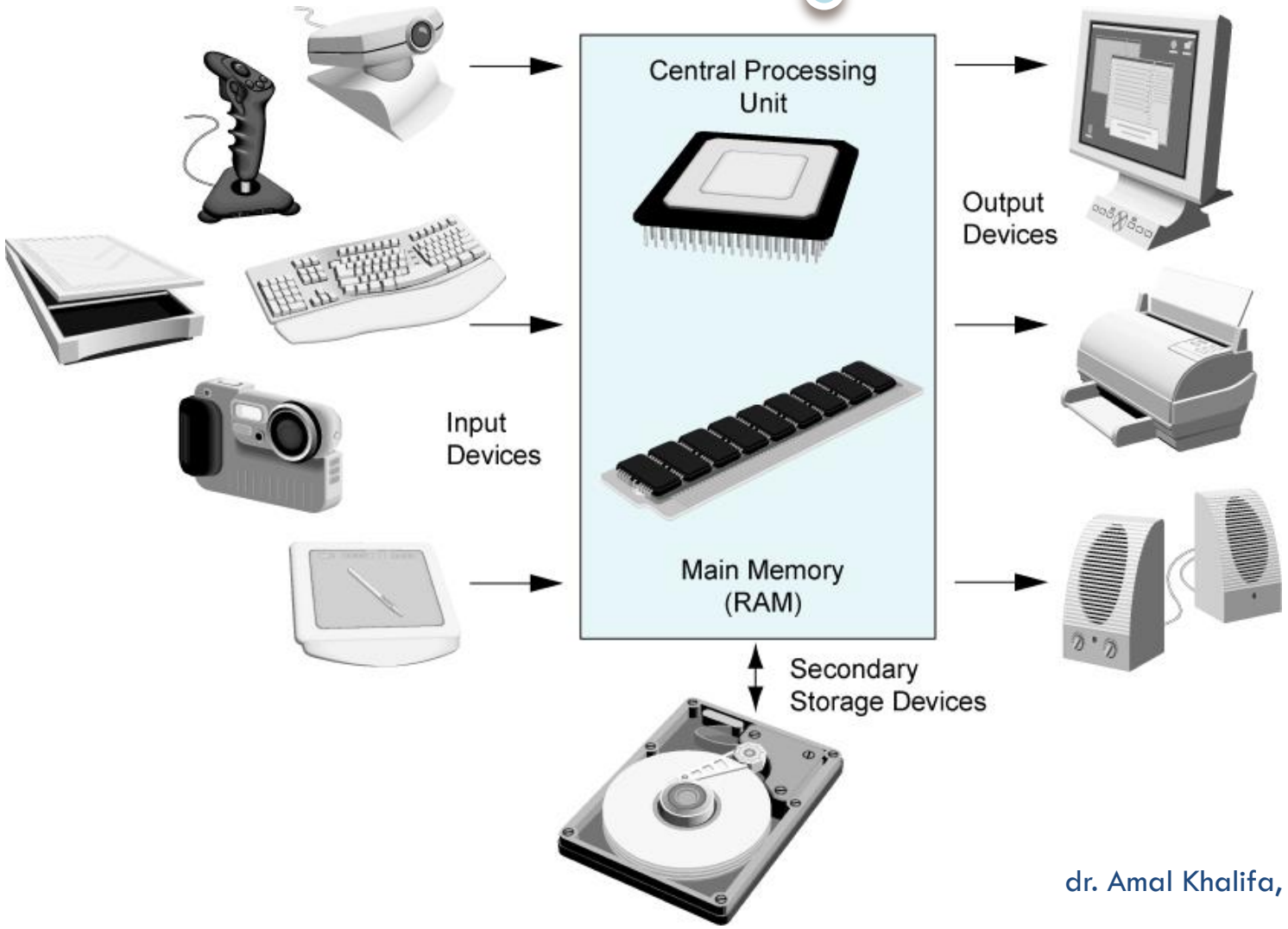
chapter 1 (pages 1 - 10)

Elements of a Computer System



Hardware

A multi-core processor implements multiple processors on a single "microchip



Software

System programs

- loads first when you turn on your PC
- Also called the operating system.
- The operating system monitors the overall activity of the computer and provides services, such as memory management, input/output activities, and storage management.

Application programs

- perform specific tasks
- Examples :
  - Word processors
  - spreadsheets, and
  - games

Both operating systems and application programs are written in programming languages.

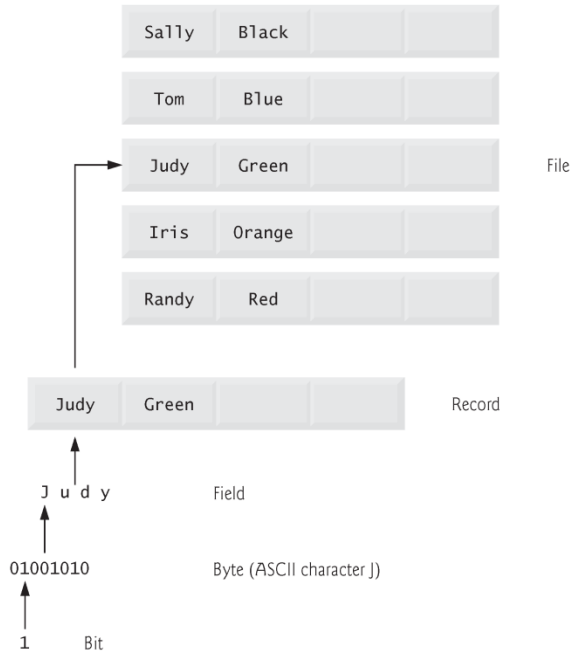


Fig. 1.1 | Data hierarchy.

Data!!

Database, big data

TABLE 1-1 Binary Units

Unit	Symbol	Bits/Bytes
Byte		8 bits
Kilobyte	KB	2^{10} bytes = 1024 bytes
Megabyte	MB	1024 KB = 2^{10} KB = 2^{20} bytes = 1,048,576 bytes
Gigabyte	GB	1024 MB = 2^{10} MB = 2^{30} bytes = 1,073,741,824 bytes
Terabyte	TB	1024 GB = 2^{10} GB = 2^{40} bytes = 1,099,511,627,776 bytes
Petabyte	PB	1024 TB = 2^{10} TB = 2^{50} bytes = 1,125,899,906,842,624 bytes
Exabyte	EB	1024 PB = 2^{10} PB = 2^{60} bytes = 1,152,921,504,606,846,976 bytes
Zettabyte	ZB	1024 EB = 2^{10} EB = 2^{70} bytes = 1,180,591,620,717,411,303,424 bytes

Binary Data

Bit: A binary digit 0 or 1.

A sequence of eight bits is called a byte.

ASCII Code

Every letter, number, or special symbol (such as * or {) on your keyboard is encoded as a sequence of bits, each having a unique representation.

The most commonly used American Standard Code for Information Interchange (ASCII).

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	SP
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?

Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|
111 1101	175	125	7D	}
111 1110	176	126	7E	~

Introduction to programming

chapter 3 (3.1, 3.2, 3.4, 3.5)

What is Computer Programming?

- *Planning or scheduling a sequence of steps for a computer to follow to perform a task.*
- Basically, telling a computer what to do and how to do it.
- *A program:*
  - *A sequence of steps to be performed by a computer.*
  - Expressed in a *computer language*.

Computer Languages

- A set of
  - Symbols (punctuation),
  - Special words or keywords (vocabulary),
  - And rules (grammar)

  used to construct a program.

Evolution of Programming Languages

- Languages differ in
  - Size (or complexity)
  - Readability
  - Expressivity (or writability)
  - "Level"
    - closeness to instructions for the CPU

Machine Language

- Binary-coded instructions
- Used directly by the CPU
- Lowest level language
- Every program step is ultimately a machine language instruction

Address	Contents
2034	10010110
2035	11101010
2036	00010010
2037	10101010
2038	10010110
2039	11101010
2040	11111111
2041	01010101
2042	10101101

Assembly Language

- Each CPU instruction is labeled with a *mnemonic*.
- Very-low level language
  - Almost 1 to 1 correspondence with machine language
- Assembler: A program that translates a program written in assembly language into an equivalent program in machine language.

Mnemonic	Instruction
ADD	10010011

Sample Program

```
MUL X,10
ADD X,Y
STO Z,20
SUB X,Z
```

Assembly Language	Machine Language
LOAD	100100
STOR	100010
MULT	100110
ADD	100101
SUB	100011

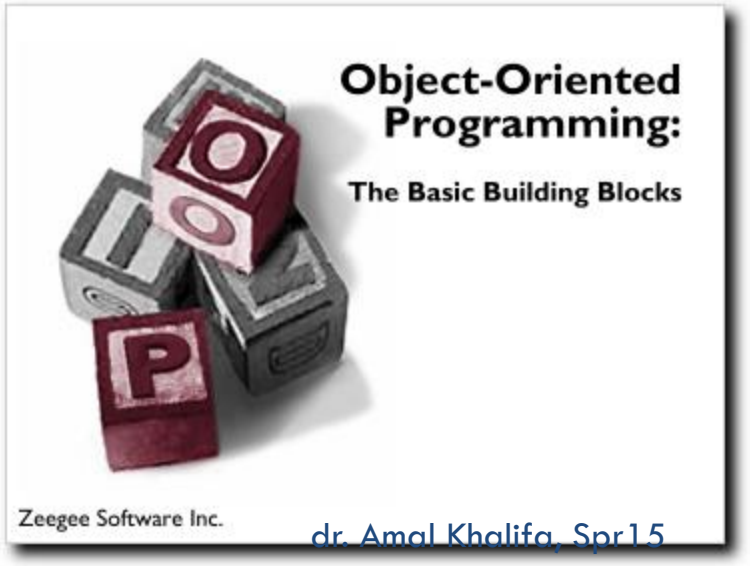
Examples of Instructions in Assembly Language and Machine Language

High-Level Languages

- Closer to natural language
- Each step maps to several machine language instruction
  - Easier to state and solve problems
- Compiler: A program that translates a program written in a high-level language into the equivalent machine language.

Object Technology== Reusability…

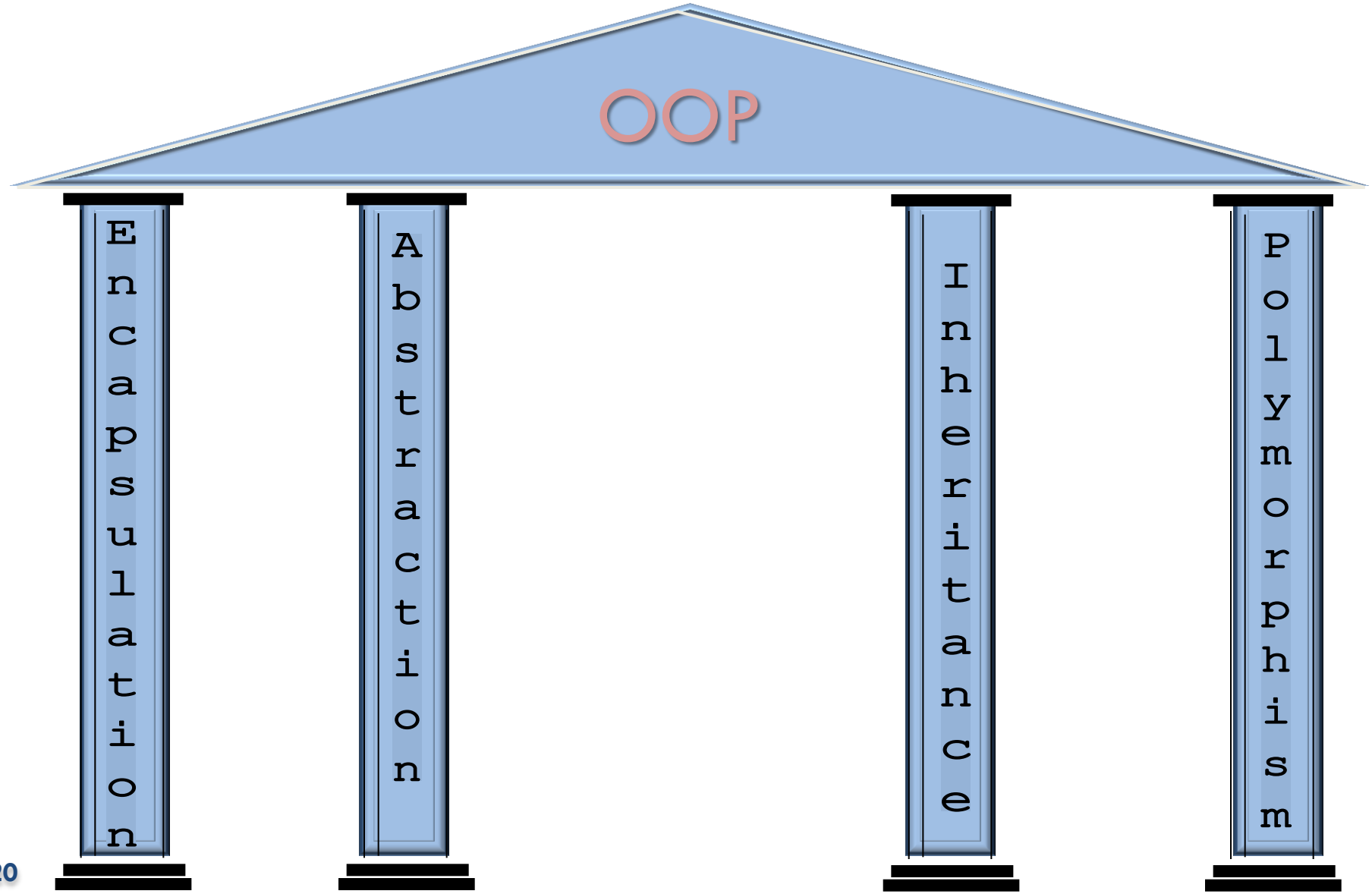
- Keep data *near* the relevant code.
- Provide a nice packaging mechanism for related code.
- Reuse is *not the same as "cut and paste"*



OOP

- 4 Key OOP Concepts
  - Encapsulation
  - Inheritance
  - Abstraction
  - Polymorphism

It will take you a long time to really understand these concepts

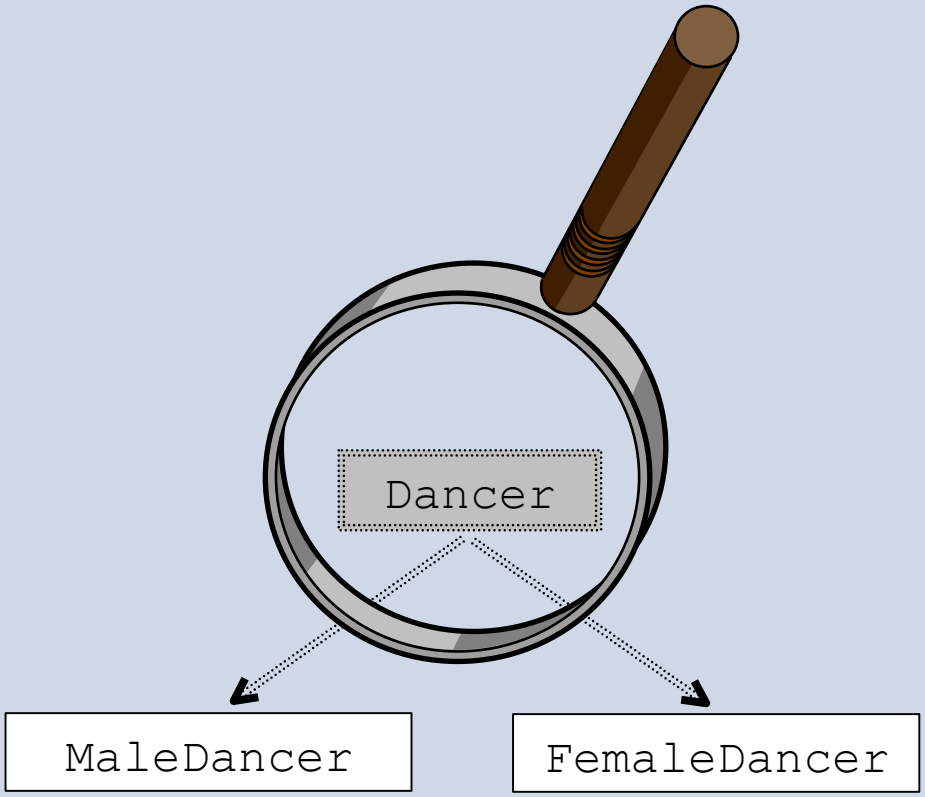

OOP
Encapsulation
Abstraction
Inheritance
Polymorphism

OOP – Encapsulation

- Put the details in one place: an object
  - group related data and operations in an object
  - Object has its own data and knows how to use it
  - information hiding

OOP - Inheritance

- A class can extend another class, inheriting all its data members and methods while redefining some of them and/or adding its own.



OOP – Abstraction

Abstraction means ignoring irrelevant features, properties, or functions and emphasizing the relevant ones...

OOP - Polymorphism

- Literally: “many shapes”
- Informally: same instruction means different things to different agents

C# basics

Chapter 3 (3.6 → 3.9)

Dr. Amal Khalifa, Spr 2015

The C# language

- C# has roots in C, C++ and Java.
- Performing a task in a program requires a method.

Console application

Saved in File ClassName.cs

`using` directive

`System` namespace

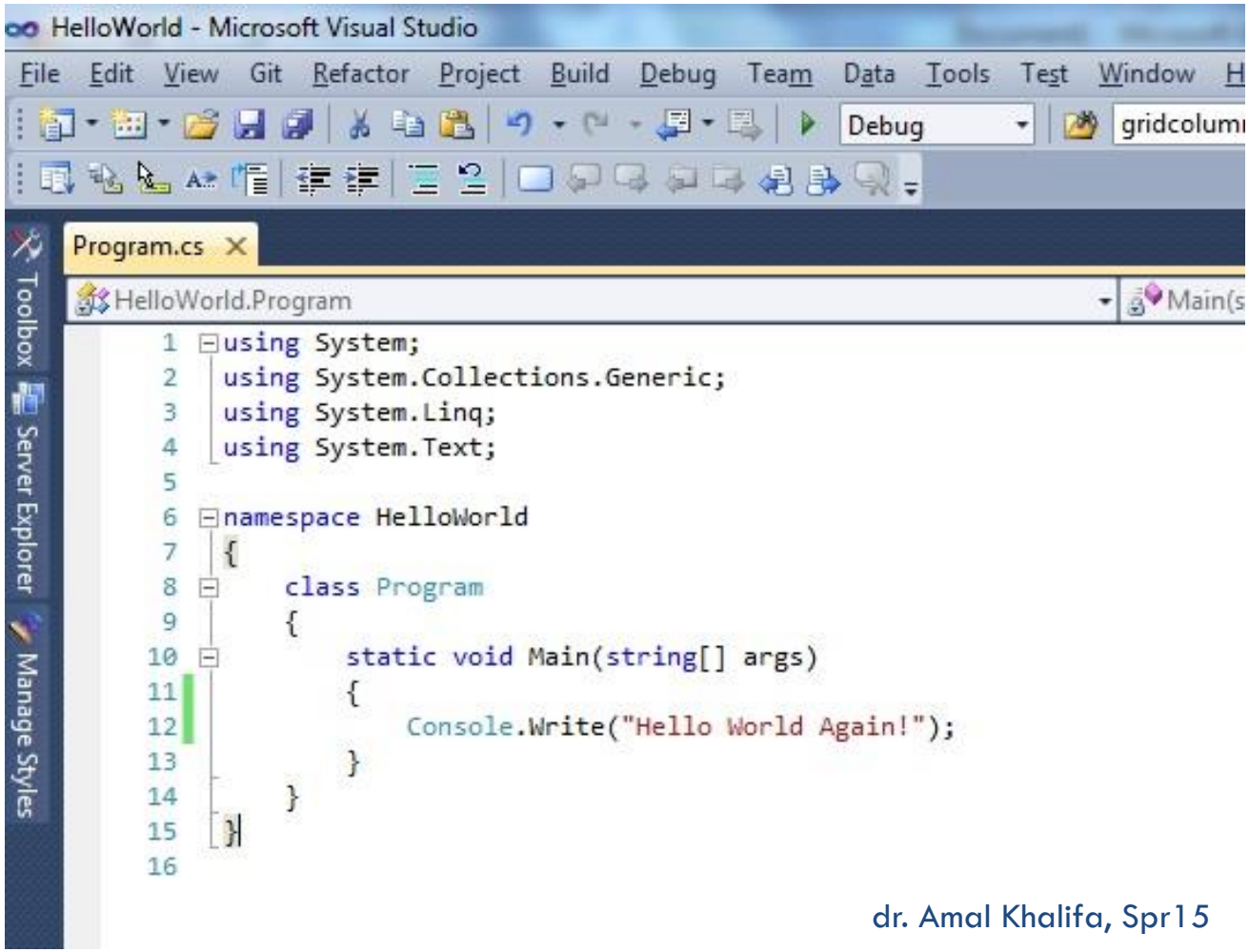
main??

Keywords

case sensitive

Braces

Strings



Example 1

comments

A class name is an identifier→ Series of letters, digits and (_), cannot begin with a digit, and does not contain spaces.

```
1  // Fig. 3.1: Welcome1.cs
2  // Text-displaying app.
3  using System;
4
5  public class Welcome1
6  {
7     // Main method begins execution of C# app
8     public static void Main( string[] args )
9     {
10       Console.WriteLine( "Welcome to C# Programming!" );
11    } // end Main
12 } // end class Welcome1
```

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

Fig. 3.1 | Text-displaying app.

Example2


```
1  // Fig. 3.10: Welcome2.cs
2  // Displaying one line of text with multiple statements.
3  using System;
4
5  public class Welcome2
6  {
7     // Main method begins execution of C# app
8     public static void Main( string[] args )
9     {
10       Console.Write( "Welcome to " );
11       Console.WriteLine( "C# Programming!" );
12    } // end Main
13 } // end class Welcome2
```

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

Fig. 3.10 | Displaying one line of text with multiple statements.

Modify the code to display each word in a separate line

Escape sequence	Description
`\n`	Newline. Positions the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Moves the screen cursor to the next tab stop.
`\"`	Double quote. Used to place a double-quote character (") in a string—e,g., `Console.Write( "\"in quotes\"" );` displays `"in quotes"`.
`\r`	Carriage return. Positions the screen cursor at the beginning of the current line—does not advance the cursor to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to place a backslash character in a string.

Fig. 3.12 | Some common escape sequences.

Formatting text with Escape sequences

Example 3

format string → fixed text and format items.

placeholder

```
1  // Fig. 3.13: Welcome4.cs
2  // Displaying multiple lines of text with string formatting.
3  using System;
4
5  public class Welcome4
6  {
7     // Main method begins execution of C# app
8     public static void Main( string[] args )
9     {
10       Console.WriteLine( "{0}\n{1}", "Welcome to", "C# Programming!" );
11    } // end Main
12 } // end class Welcome4
```

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

Fig. 3.13 | Displaying multiple lines of text with string formatting.

variables

- Every variable has a name, a type, a size and a value.

data type **variable name**

```
int total = 0;        // Initialization

int count, temp, result;
```

Multiple variables can be created in one declaration

Type	Size in bits	Value range	Standard
`bool`	8	`true` or `false`	
`byte`	8	0 to 255, inclusive	
`sbyte`	8	–128 to 127, inclusive	
`char`	16	`'\u0000'` to `'\uFFFF'` (0 to 65535), inclusive	Unicode
`short`	16	–32768 to 32767, inclusive	
`ushort`	16	0 to 65535, inclusive	
`int`	32	–2,147,483,648 to 2,147,483,647, inclusive	
`uint`	32	0 to 4,294,967,295, inclusive	
`float`	32	*Approximate negative range:* –3.4028234663852886E+38 to –1.4012984632481707E–45 *Approximate positive range:* 1.4012984632481707E–45 to 3.4028234663852886E+38 *Other supported values:* positive and negative zero positive and negative infinity not-a-number (NaN)	IEEE 754 IEC 60559
`long`	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807, inclusive	
`ulong`	64	0 to 18,446,744,073,709,551,615, inclusive	

Standard Input

- input is received from Terminal window.
- Input entered while program is executing.
- The Console's `ReadLine` method waits for the user to type a string of characters at the keyboard and press the Enter key.
- `Console.ReadLine()` returns the text the user entered.
  - Use appropriate methods to converts this sequence of characters into a certain data of type

Arithmetic Expressions

- An *expression* is a combination of operators and operands
- *Arithmetic expressions* special methods applied to numerical data objects. They compute numeric results and make use of the arithmetic operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

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

Fig. 3.18 | Arithmetic operators.

Expressions

The remainder operator is most commonly used with integer operands but can also be used with `float`s, `double`s, and `decimal`s.

Common Programming Error 7.4

Confusing the + operator used for string concatenation with the + operator used for addition can lead to strange results. The + operator is left-associative. For example, if integer variable y has the value 5, the expression "y + 2 = " + y + 2 results in the string "y + 2 = 52", not "y + 2 = 7", because first the value of y (5) is concatenated with the string "y + 2 = ", then the value 2 is concatenated with the new larger string "y + 2 = 5". The expression "y + 2 = " + (y + 2) produces the desired result "y + 2 = 7".

String concatination

How Do Data Conversions Happen?

- Explicitly: *Casting*
  - widening / narrowing conversions
    - Examples:

```
 double MyResult;
MyResult = 12.0 / 5.0; //OK
int myInt = (int) MyResult; // truncation
MyResult  = (double)myInt/3.0;
```

Operator precedence


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

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

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

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

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

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

Fig. 3.20 | Order in which a second-degree polynomial is evaluated.

Assignment-related Operators

- Increment and decrement operators: ++, --
- Assignment operators: +=, -=, *=, /=

these three expressions have the same effect

```
count  =  count + 1;
count  += 1;
count  ++;
```

these two expressions have the same effect

```
count  =  count - 10;
count  -= 10;
```

```
1  // Fig. 5.15: Increment.cs
2  // Prefix increment and postfix increment operators.
3  using System;
4
5  public class Increment
6  {
7     public static void Main( string[] args )
8     {
9        int c;
10
11       // demonstrate postfix increment operator
12       c = 5; // assign 5 to c
13       Console.WriteLine( c ); // display 5
14       Console.WriteLine( c++ ); // increment c and display 5
15       Console.WriteLine( c ); // display 6
16
17       Console.WriteLine(); // skip a line
18
19       // demonstrate prefix increment operator
20       c = 5; // assign 5 to c
21       Console.WriteLine( c ); // display 5
22       Console.WriteLine( ++c ); // increment c and display 6
23       Console.WriteLine( c ); // display 6 again
24    } // end Main
25 } // end class Increment
```

```
5
5
6

5
6
6
```

Constants

- A “constant variable” is an identifier that is similar to a variable except that it holds one value for its entire existence
- Why constants:
  - give names to otherwise unclear literal values
  - facilitate changes to the code
  - prevent inadvertent errors
- In C#:

```
const double PI = 3.14159265;
```

That's all !!