

---

# Introduction to Programming

*Computer Hardware/Software;  
Variables, C# Data Types, and IO*

# Outline

---

- ❑ Computer hardware/software
- ❑ Variables and C# data types
- ❑ Input and output

# C# Program Structure (Console)

## □ Program specifications (optional)

```
//=====
//
// File: HelloWorld.cs
//
//
//
// Classes: HelloWorld
// -----
// This program prints the string "Hello World!"
//
//=====
```

## □ Library imports (optional)

```
using System;
```

## □ Class and namespace definitions

```
class HelloWorld
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

# C# Program Structure (Window)

## □ Program specifications (optional)

```
//=====
//
// File: HelloWorld.cs
//
//
//
// Classes: HelloWorld
// -----
// This program shows a message box.
//
//=====
```

## □ Library imports (optional)

```
using System;
using System.Windows.Forms;
```

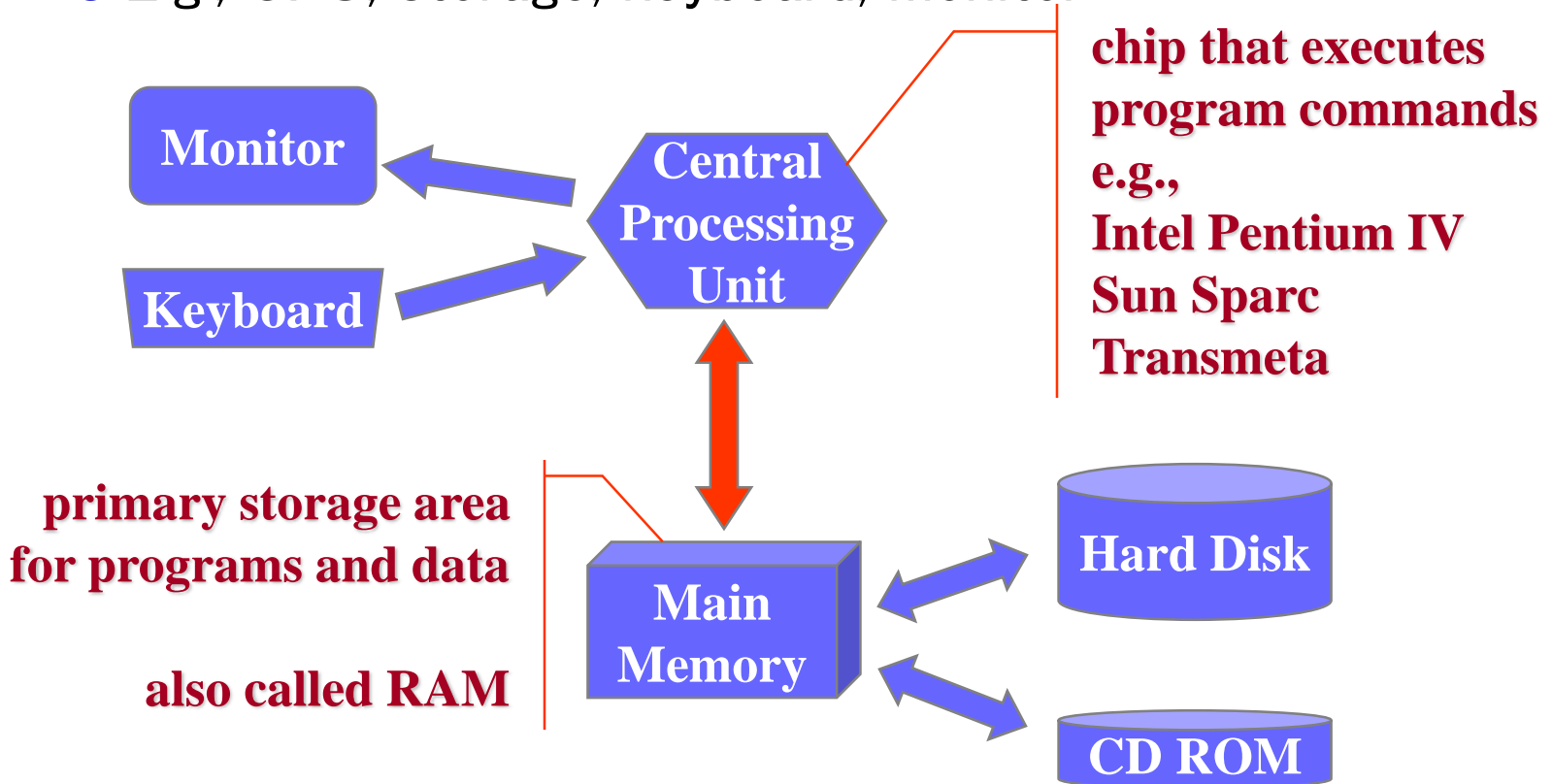
## □ Class and namespace definitions

```
class HelloWorld
{
    static void Main(string[] args)
    {
        MessageBox.Show("Hello World!");
    }
}
```

# Computer Environment: Hardware

## □ Hardware

- the physical, tangible parts of a computer
- E.g., CPU, storage, keyboard, monitor



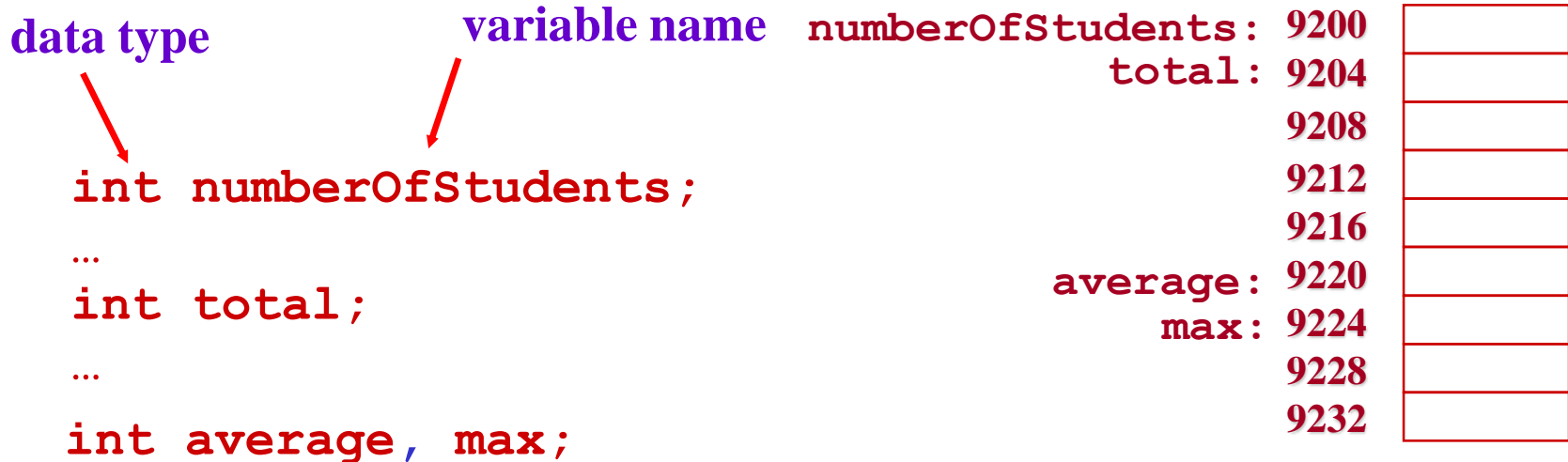
# Variables

- ❑ *Avariable* is a typed name for a location in memory
- ❑ Avariable must be *declared*, specifying the variable's **name** and the **type** of information that will be held in it

**data type**                      **variable name**

```
int numberOfStudents;
...
int total;
...
int average, max;
```

numberOfStudents: 9200  
total: 9204  
9208  
9212  
9216  
average: 9220  
max: 9224  
9228  
9232




Which ones are valid variable names?

myBigVar	VAR1	_test	@test
99bottles	namespace	It's-all-over	

# Assignment

- ❑ An *assignment statement* changes the value of a variable
- ❑ The assignment operator is the = sign

```
int total;  
...  
total = 55;
```



- ❑ The value on the right is stored in the variable on the left
  - The value that was in `total` is overwritten
- ❑ You can only assign a value to a variable that is consistent with the variable's declared type (more later)
- ❑ You can declare and assign initial value to a variable at the same time, e.g.,

```
int total = 55;
```

# Example

```
static void Main(string[] args)
{
    int total;

    total = 15;
    System.Console.Write("total = ");
    System.Console.WriteLine(total);

    total = 55 + 5;
    System.Console.Write("total = ");
    System.Console.WriteLine(total);
}
```



# Constants

---

- ❑ A constant is similar to a variable except that it holds one value for its entire existence
- ❑ The compiler will issue an error if you try to change a constant
- ❑ In C#, we use the `constant` modifier to declare a constant

```
constant int numberOfStudents = 42;
```

- ❑ Why constants?
  - give names to otherwise unclear literal values
  - facilitate changes to the code
  - prevent inadvertent errors

# Token

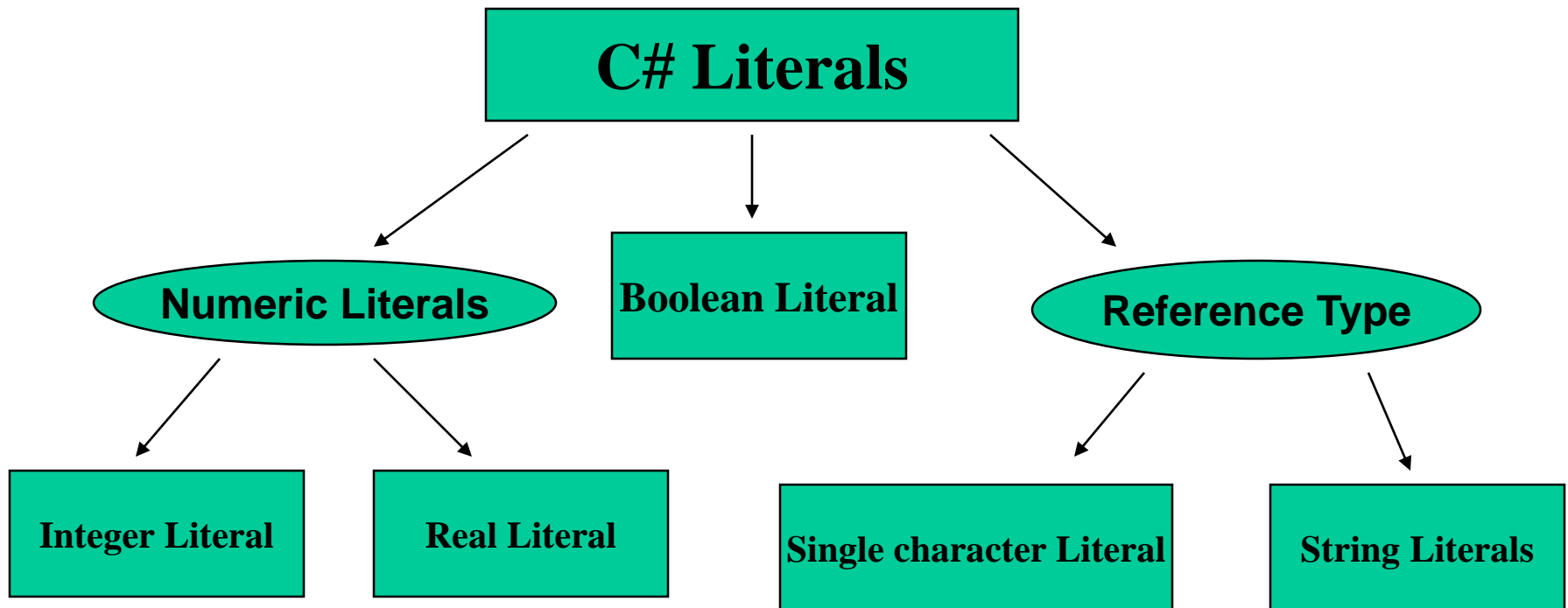
---

The Smallest,non-reducible,textual elements in a program are reffered to as token.There are five types of token.

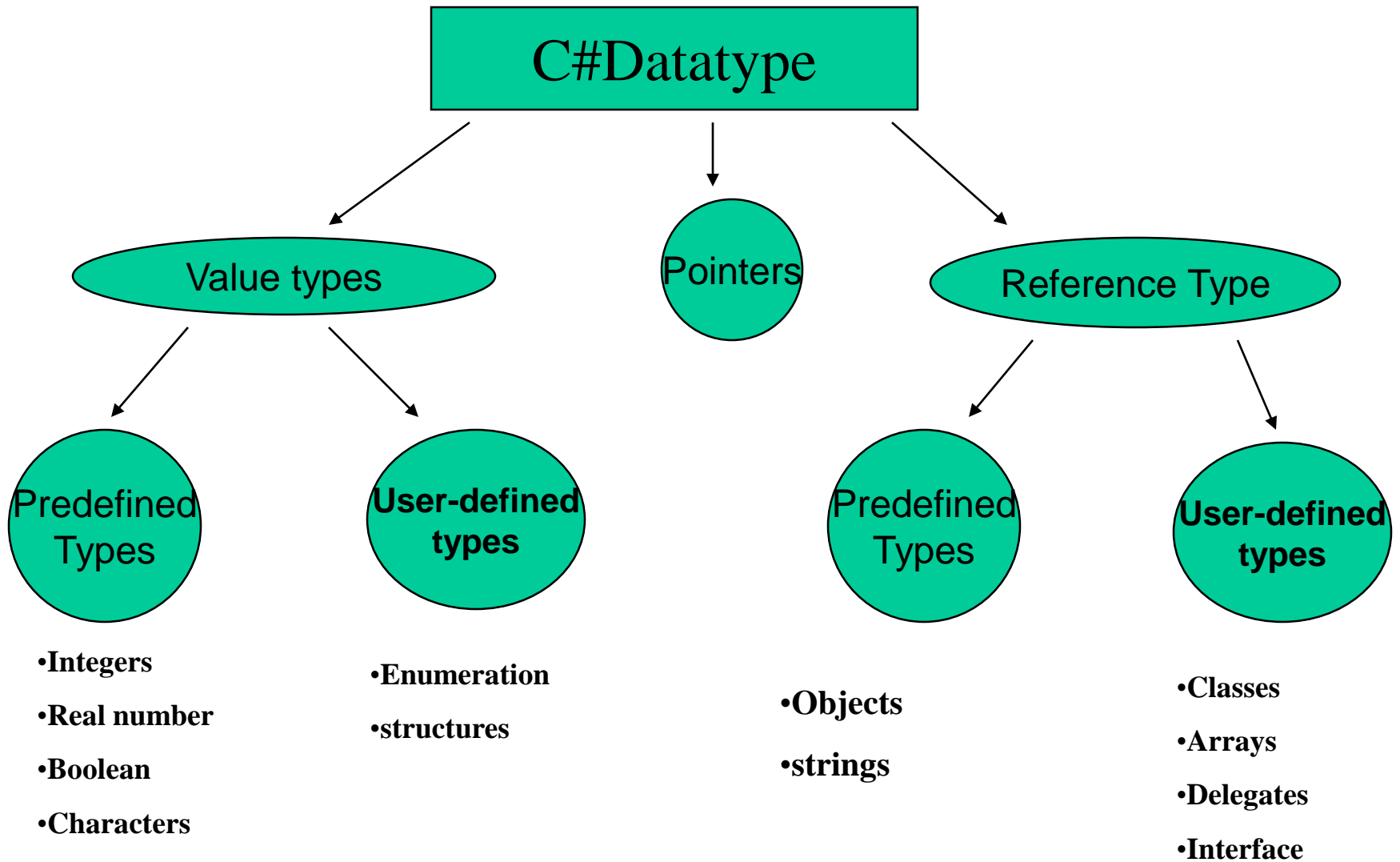
- ❑ Keywords
- ❑ Identifiers
- ❑ Literals
- ❑ Operators
- ❑ Punctuators

# Literals

**Literals are value constant assigned to variables**



# Datatype



# C# Data Types

- ❑ There are 15 data types in C#
- ❑ Eight of them represent integers:
  - byte, sbyte, short, ushort, int, uint, long, ulong
- ❑ Two of them represent floating point numbers
  - float, double
- ❑ One of them represents decimals:
  - decimal
- ❑ One of them represents boolean values:
  - bool
- ❑ One of them represents characters:
  - char
- ❑ One of them represents strings:
  - string
- ❑ One of them represents objects:
  - object

# Numeric Data Types

- The difference between the various numeric types is their size, and therefore the values they can store:

<u>Type</u>	<u>Storage</u>	<u>Range</u>
byte	8 bits	0 - 255
sbyte	8 bits	-128 - 127
short	16 bits	-32,768 - 32767
ushort	16 bits	0 - 65537
int	32 bits	-2,147,483,648 – 2,147,483,647
uint	32 bits	0 – 4,294,967,295
long	64 bits	$-9 \times 10^{18}$ to $9 \times 10^{18}$
ulong	64 bits	0 – $1.8 \times 10^{19}$
decimal	128 bits	$\pm 1.0 \times 10^{-28}$ ; $\pm 7.9 \times 10^{28}$ with 28-29 significant digits
float	32 bits	$\pm 1.5 \times 10^{-45}$ ; $\pm 3.4 \times 10^{38}$ with 7 significant digits
double	64 bits	$\pm 5.0 \times 10^{-324}$ ; $\pm 1.7 \times 10^{308}$ with 15-16 significant digits

# Examples of Numeric Variables

```
int x = 1;
short y = 10;
float pi = 3.14f; // f denotes float
float f3 = 7E-02f; // 0.07
double d1 = 7E-100;
// use m to denote a decimal
decimal microsoftStockPrice = 28.38m;
```

# Boolean

---

- ❑ A `bool` value represents a true or false condition
- ❑ A boolean can also be used to represent any two states, such as a light bulb being on or off
- ❑ The reserved words `true` and `false` are the only valid values for a boolean type

```
bool doAgain = true;
```



# Characters

- ❑ A `char` is a single character from the *a character set*
- ❑ A *character set* is an ordered list of characters; each character is given a unique number
- ❑ C# uses the **Unicode** character set, a superset of ASCII
  - Uses sixteen bits per character allowing for 65,536 unique characters
  - It is an international character set, containing symbols and characters from many languages
- ❑ Character literals are represented in a program by delimiting with single quotes, e.g.,

```
'a' 'X' '7' '$' ','
```

```
char response = 'Y';
```

# Common Escape Sequences

Escape sequence	Description
<code>\n</code>	Newline. Position the screen cursor to the beginning of the next line.
<code>\t</code>	Horizontal tab. Move the screen cursor to the next tab stop.
<code>\r</code>	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.
<code>\'</code>	Used to print a single quote
<code>\\</code>	Backslash. Used to print a backslash character.
<code>\"</code>	Double quote. Used to print a double quote (") character.

# string

---

- A `string` represents a sequence of characters, e.g.,

```
string message = "Hello World";
```

- Strings can be created with verbatim string literals by starting with `@`, e.g.,

```
string a2 = @"\\server\fileshare\Hello.cs";
```

# Boxing and Unboxing

---

When the compiler finds a value type where it needs a reference type, it creates an object 'box' into which it places the value of the value type.

# Example Of Boxing:

---

```
int m = 10;
```

```
object om = m; // creates a box to hold m
```

```
m = 20;
```

```
Console.WriteLine(m);           // m=20
```

```
Console.WriteLine(om);         // m=10
```

When a code changes the value of **m**,the value of **om** is not effected.

# Example Of Unboxing:

---

```
int m = 10;
```

```
Object om = m; // creates a box to hold m
```

```
int n = (int) om; // unbox om back to an int
```

If you write this, it will produce a run time error

```
int m = 500;
```

```
Object om = m;
```

```
byte n = (byte)om;
```

# Data Input

---

## ❑ Console.ReadLine()

- Used to get a value from the user input
- Example

```
string myString = Console.ReadLine();
```

## ❑ Convert from string to the correct data type

### ○ Int32.Parse()

- Used to convert a string argument to an integer
- Allows math to be performed once the string is converted
- Example:

```
string myString = "1023";  
int myInt = Int32.Parse( myString );
```

### ○ Double.Parse()

### ○ Single.Parse()

# Output

- ❑ `Console.WriteLine(variableName)` will print the variable
- ❑ You can use the values of some variables at some positions of a string:

```
System.Console.WriteLine("{0} {1}.", iAmVar0, iAmVar1);
```

- ❑ You can control the output format by using the format specifiers:

```
float price = 2.5f;  
System.Console.WriteLine("Price = {0:C}.", price);
```

```
Price = $2.50.
```