Data Types and Variables

Numeral Types, Text Types and Type Conversion

greenwich.edu.vn

UNIVERSITY of GREENWICH

Alliance with FPT Education

Table of Contents

- Data Types and Variables
- Integer and Real Number Type
- Type Conversion
- Boolean Type
- Character and String Type

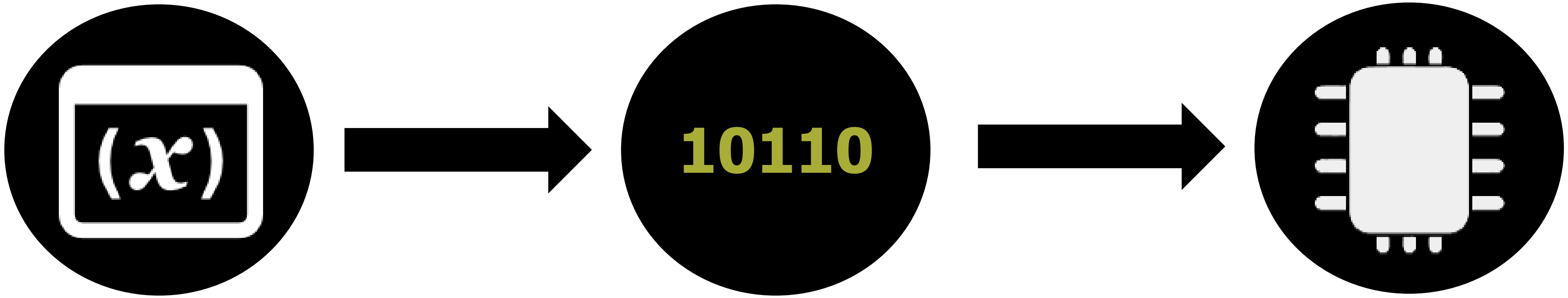
UNIVERSITY of GREENWICH

Alliance with FPT Education

DATA TYPES AND VARIABLES

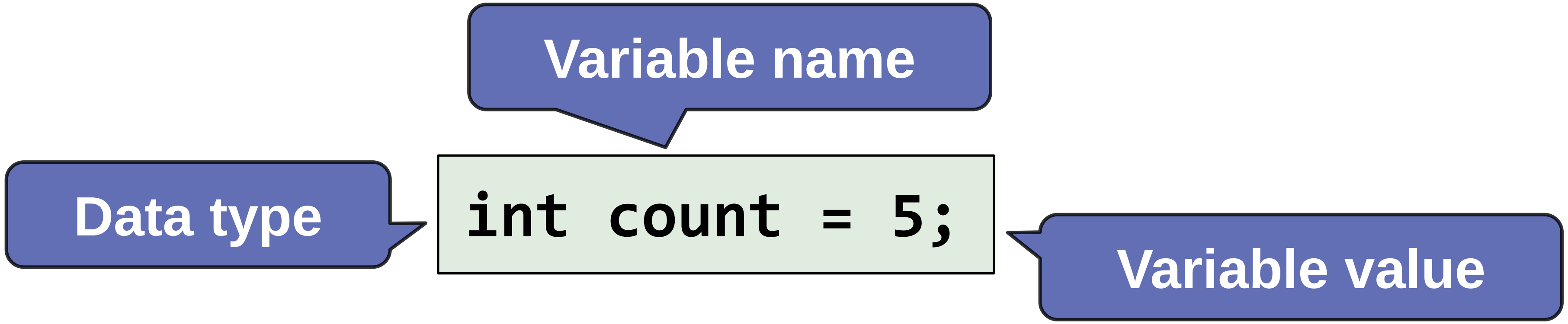
How Computing Works?

- Computers are machines that process data
  - – Instructions and data are stored in the computer memory



Variables

- Variables have name, data type and value
  - Assignment is done by the operator "="
  - Example of variable definition and assignment in C#



Variable name

Data type

```
int count = 5;
```

Variable value

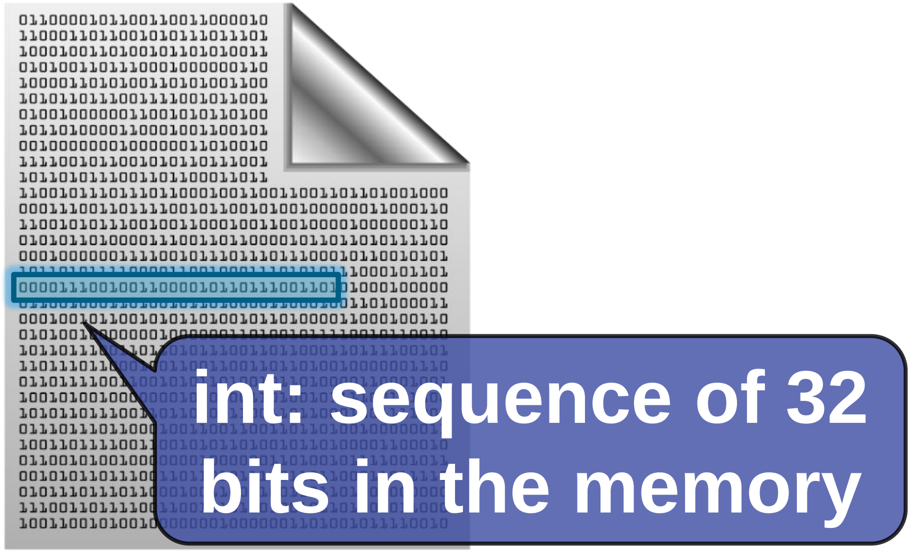
- When processed, data is stored back into variables

What is a Data Type?

- A data type:
  - Is a domain of values of similar characteristics
  - Defines the type of information stored in the computer memory (in a variable)
- Examples:
  - Positive integers: 1, 2, 3, …
  - Alphabetical characters: a, b, c, …
  - Days of week: Monday, Tuesday, …

Data Type Characteristics

- A data type has:
  - – Name (C# keyword or .NET type)
  - – Size (how much memory is used)
  - – Default value
- Example:
  - – Integer numbers in C#
  - – Name: int
  - – Size: 32 bits (4 bytes)
  - – Default value: 0



int: sequence of 32 bits in the memory

int: 4 sequential bytes in the memory

Naming Variables

- Always refer to the naming conventions of a programming language – for C# use camelCase
- Preferred form: [Noun] or [Adjective] + [Noun]
- Should explain the purpose of the variable (Always ask yourself "What this variable contains?")

✓

```
firstName, report, config, fontSize, maxSpeed
```

🚫

```
foo, bar, p, p1, LastName, last_name, LAST_NAME
```

Variable Scope and Lifetime

- Scope == where you can access a variable (global, local)
- Lifetime == how long a variable stays in memory

Accessible in the Main()

```
string outer = "I'm inside the Main()";
for (int i = 0; i < 10; i++)
{
   string inner = "I'm inside the loop";
}
Console.WriteLine(outer);
// Console.WriteLine(inner); Error
```

Accessible only in the loop

Variable Span

- Variable span is how long before a variable is called
- Always declare a variable as late as possible (e.g. shorter span)

```
static void Main()
{
  string outer = "I'm inside the Main()";
  for (int i = 0; i < 10; i++)
    string inner = "I'm inside the loop";
  Console.WriteLine(outer);
  //Console.WriteLine(inner); Error
}
```

"outer" variable span

Keep Variable Span Short

- Shorter span simplifies the code
  - Improves its readability and maintainability

```
for (int i = 0; i < 10; i++)
{
   string inner = "I'm inside the loop";
}
string outer = "I'm inside the Main()";
Console.WriteLine(outer);
// Console.WriteLine(inner); Error
```

"outer" variable span – reduced

UNIVERSITY of GREENWICH

Alliance with FPT Education

INTEGER TYPES

Integer Types

Type	Default Value	Min Value	Max Value	Size
sbyte	0	-128 (-2^7)	127 (2^7-1)	8 bit
byte	0	0	255 (2^8-1)	8 bit
short	0	-32768 (-2^{15})	32767 (2^{15} - 1)	16 bit
ushort	0	0	65535 (2^{16}-1)	16 bit
int	0	-2147483648 (-2^{31})	2147483647 (2^{31} – 1)	32 bit
uint	0	0	4294967295 (2^{32}-1)	32 bit
long	0	-9223372036854775808 (-2^{63})	9223372036854775807 (2^{63}-1)	64 bit
ulong	0	0	18446744073709551615 (2^{64}-1)	64 bit

Centuries – Example

- Depending on the unit of measure we can use different data types:

```
byte centuries = 20;
ushort years = 2000;
uint days = 730484;
ulong hours = 17531616;
Console.WriteLine(
  "{0} centuries = {1} years = {2} days = {3} hours.",
  centuries, years, days, hours);
     //20 centuries = 2000 years = 730484 days = 17531616 hours.
```

Beware of Integer Overflow!

- Integers have range (minimal and maximal value)
- Integers could overflow → this leads to incorrect values

```
byte counter = 0;
for (int i = 0; i < 260; i++)
{
  counter++;
  Console.WriteLine(counter);
}
```

→

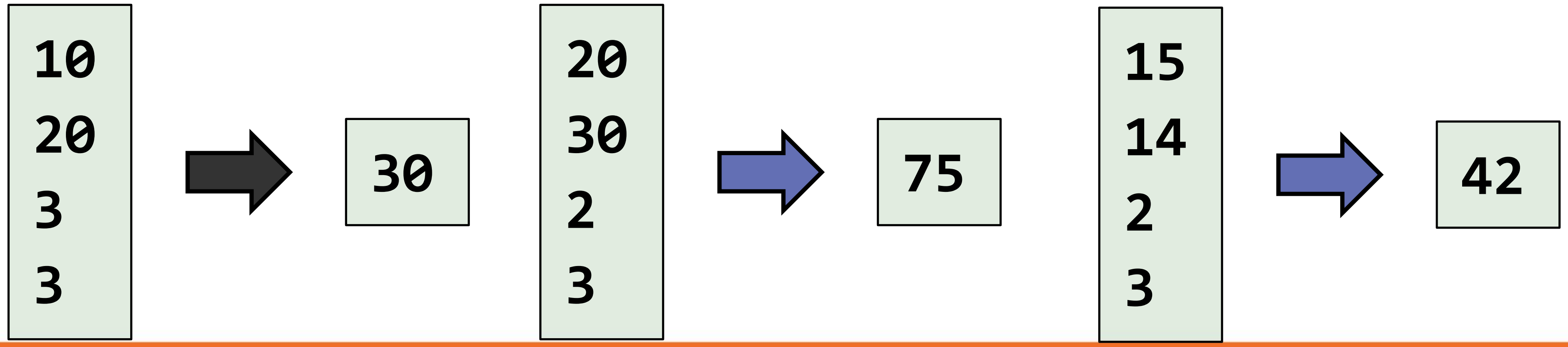
```
1
2
…
255
0
1
```

Integer Literals

- Examples of integer literals:
  - The '0x' and '0X' prefixes mean a hexadecimal value
    - E.g. 0xFE, 0xA8F1, 0xFFFFFFFF
  - The 'u' and 'U' suffixes mean a ulong or uint type
    - E.g. 12345678U, 0U
  - The 'l' and 'L' suffixes mean a long
    - E.g. 9876543L, 0L

Problem: Integer Operations

- Read four integers
  - – Add first to the second
  - – Divide the sum by the third number (integer division)
  - – Multiply it by the fourth number
  - – Print the result



```
10
20
3
3
```

```
30
```

```
20
30
2
3
```

```
75
```

```
15
14
2
3
```

```
42
```

REAL NUMBER TYPES

What are Floating-Point Types?

- Floating-point types:
  - Represent real numbers, e.g. 1.25, -0.38
  - Have range and precision depending on the memory used
  - Sometimes behave abnormally in the calculations
  - May hold very small and very big values like 0.00000000000001 and 10000000000000000000000000000000000.0

Floating-Point Numbers

- Floating-point types are:
  - – float (±1.5 × 10−45 to ±3.4 × 1038)
    - 32-bits, precision of 7 digits
  - – double (±5.0 × 10−324 to ±1.7 × 10308)
    - 64-bits, precision of 15-16 digits
- The default value of floating-point types:
  - – Is 0.0F for the float type
  - – Is 0.0D for the double type

PI Precision – Example

- Difference in precision when using float and double:

```
float floatPI = 3.141592653589793238f;
double doublePI = 3.141592653589793238;
Console.WriteLine("Float PI is: {0}", floatPI);
Console.WriteLine("Double PI is: {0}", doublePI);
```

3.141593

3.14159265358979

- NOTE: The "f" suffix in the first statement
  - Real numbers are by default interpreted as double
  - One should explicitly convert them to float

Problem: Circle Area (12 Digits Precision)

- Write program to enter a radius r (real number) and prints the area of the circle with exactly 12 digits after the decimal point:

```
2.5
```
→
```
19.634954084936
```

```
1.2
```
→
```
4.523893421169
```

- Sample solution:

```
double r = double.Parse(Console.ReadLine());
Console.WriteLine("{0:F12}", Math.PI * r * r);
```

Scientific Notation

- Floating-point numbers can use scientific notation, e.g.
  - 1e+34, 1E34, 20e-3, 1e-12, -6.02e28

```
double d = 10000000000000000000000000000000000.0;
Console.WriteLine(d); // 1E+34

double d2 = 20e-3;
Console.WriteLine(d2); // 0.02

double d3 = double.MaxValue;
Console.WriteLine(d3); // 1.79769313486232E+308
```

Floating-Point Division

- Integral division and floating-point division are different:

```
Console.WriteLine(10 / 4);    // 2 (integral division)
Console.WriteLine(10 / 4.0);  // 2.5 (real division)

Console.WriteLine(10 / 0.0);  // Infinity
Console.WriteLine(-10 / 0.0); // -Infinity

Console.WriteLine(0 / 0.0);   // NaN (not a number)
Console.WriteLine(8 % 2.5);   // 0.5 (3 * 2.5 + 0.5 = 8)
```

Floating-Point Calculations – Abnormalities

- Sometimes floating-point numbers work incorrectly!

```
Console.WriteLine(100000000000000.0 + 0.3);
// 100000000000000 (loss of precision)
double a = 1.0f, b = 0.33f, sum = 1.33;
Console.WriteLine("a+b={0} sum={1} equal={2}",
  a+b, sum, (a+b == sum));
// a+b=1.33000001311302 sum=1.33 equal = False
double one = 0;
for (int i = 0; i < 10000; i++) one += 0.0001;
  Console.WriteLine(one); // 0.999999999999906
```

Decimal Floating-Point Type

- There is a special decimal floating-point real number type in C#:
  - decimal (±1,0 × 10-28 to ±7,9 × 1028)
    - 128-bits, precision of 28-29 digits
  - Used for financial calculations
  - Almost no round-off errors
  - Almost no loss of precision
- The default value of decimal type is:
  - 0.0M (M is the suffix for decimal numbers)

Problem: Exact Sum of Real Numbers

- Write program to enter n numbers and print their exact sum:

```
2
1000000000000000000
5
```

```
1000000000000000005
```

```
2
0.00000000003
333333333333.3
```

```
333333333333.30000000003
```

TYPE CONVERSION

Type Conversion

- Variables hold values of certain type
- Type can be changed (converted) to another type
  - Implicit type conversion (lossless): variable of bigger type (e.g. double) takes smaller value (e.g. float)

```
float heightInMeters = 1.74f;
double maxHeight = heightInMeters;
```

Implicit conversion

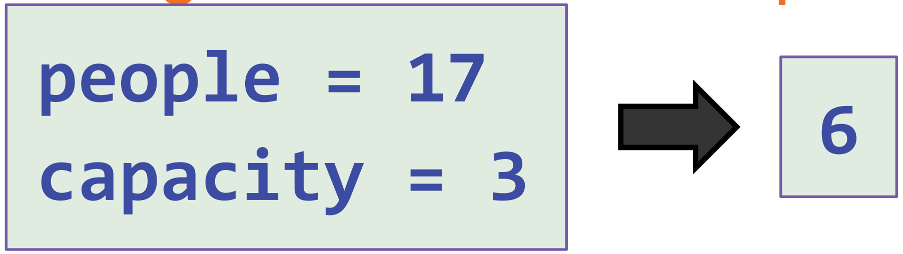
  - Explicit type conversion (lossy) – when precision can be lost:

```
double size = 3.14;
int intSize = (int) size;
```

Explicit conversion

Problem: Elevator

- Calculate how many courses will be needed to elevate n people by using an elevator of capacity of p people.



```
people = 17
capacity = 3
```

```
6
```

- Sample solution:

```
int n = int.Parse(Console.ReadLine());
int p = int.Parse(Console.ReadLine());
int courses = (int) Math.Ceiling((double)n / p);
Console.WriteLine(courses);
```

UNIVERSITY of GREENWICH
Alliance with FPT Education

Problem: Centuries to Minutes

Write program to enter an integer number of centuries and convert it to years, days, hours and minutes

```
Centuries = 1
```

```
1 centuries = 100 years = 36524 days = 876576 hours = 52594560 minutes
```

```
Centuries = 5
```

```
5 centuries = 500 years = 182621 days = 4382904 hours = 262974240 minutes
```

The output is on one row

Solution: Centuries to Minutes

```
Console.Write("Centuries = ");
int centuries = int.Parse(Console.ReadLine());
int years = centuries * 100;
int days = (int) (years * 365.2422);
int hours = 24 * days;
int minutes = 60 * hours;
Console.WriteLine(
  "{0} centuries = {1} years = {2} days = {3} hours = {4} minutes",
  centuries, years, days, hours, minutes);
```

Tropical year has 365.2422 days

(int) converts double to int

BOOLEAN TYPE

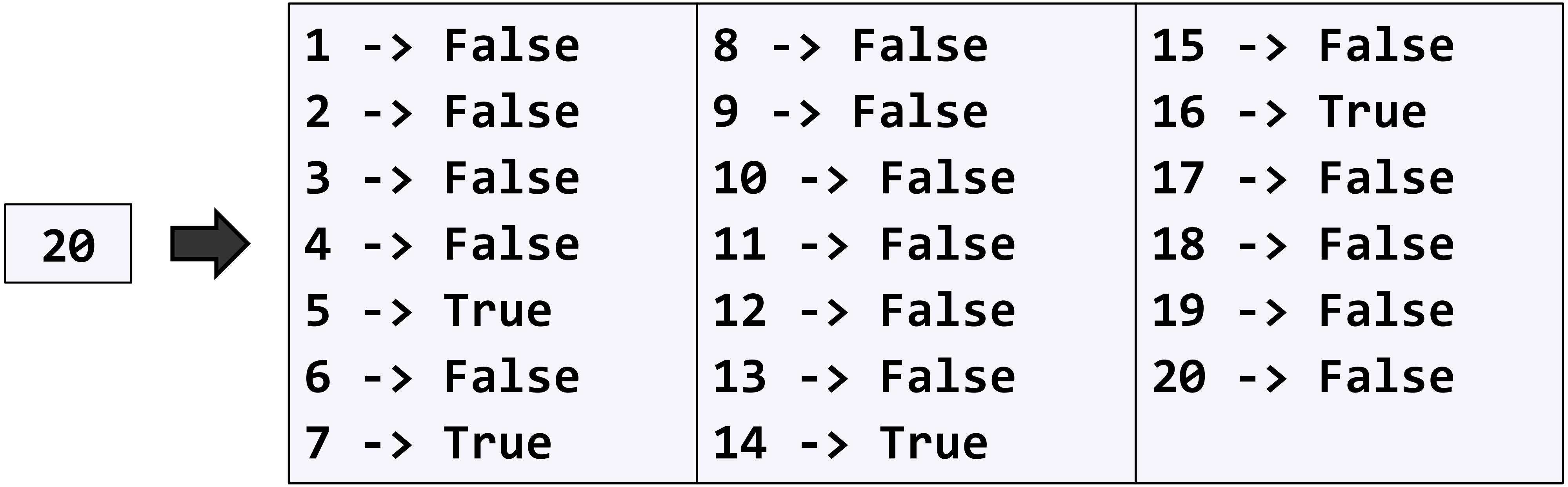
Boolean Type

- Boolean variables (bool) hold true or false:

```
int a = 1;
int b = 2;
bool greaterAB = (a > b);
Console.WriteLine(greaterAB);  // False
bool equalA1 = (a == 1);
Console.WriteLine(equalA1);    // True
```

Problem: Special Numbers

- A number is special when its sum of digits is 5, 7 or 11
  - For all numbers … print the number and if it is special



```
20
```

```
1 -> False
2 -> False
3 -> False
4 -> False
5 -> True
6 -> False
7 -> True
8 -> False
9 -> False
10 -> False
11 -> False
12 -> False
13 -> False
14 -> True
15 -> False
16 -> True
17 -> False
18 -> False
19 -> False
20 -> False
```

Solution: Special Numbers

```
int n = int.Parse(Console.ReadLine());
for (int num = 1; num <= n; num++)
{
  int sumOfDigits = 0;
  int digits = num;
  while (digits > 0)
  {
    sumOfDigits += digits % 10;
    digits = digits / 10;
  }
  // TODO: check whether the sum is special
}
```

CHARACTER TYPE

The Character Data Type

- The character data type in C#
  - – Represents symbolic information
  - – Is declared by the char keyword
  - – Gives each symbol a corresponding integer code
  - – Has a '\0' default value
  - – Takes 16 bits of memory (from U+0000 to U+FFFF)
  - – Holds a single Unicode character (or part of character)

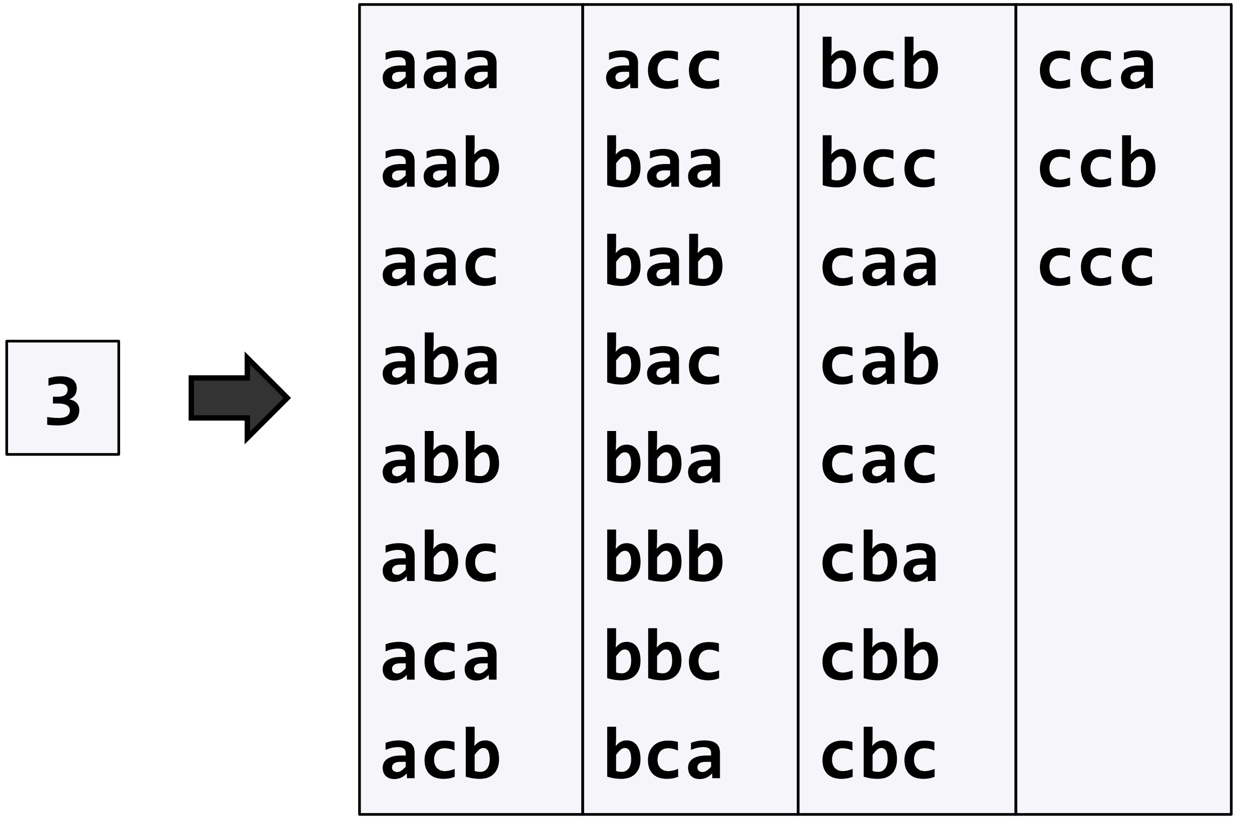
Characters and Codes

- Each character has an unique Unicode value (int):

```
char ch = 'a';
Console.WriteLine("The code of '{0}' is: {1}", ch, (int) ch);
ch = 'b';
Console.WriteLine("The code of '{0}' is: {1}", ch, (int) ch);
ch = 'A';
Console.WriteLine("The code of '{0}' is: {1}", ch, (int) ch);
```

Problem: Triples of Latin Letters

- Write a program to read an integer n and print all triples of the first n small Latin letters, ordered alphabetically:



```
3
```

```
aaa   acc   bcb   cca
aab   baa   bcc   ccb
aac   bab   caa   ccc
aba   bac   cab
abb   bba   cac
abc   bbb   cba
aca   bbc   cbb
acb   bca   cbc
```

Solution: Triples of Latin Letters

```
int n = int.Parse(Console.ReadLine());
 for (int i1 = 0; i1 < n; i1++)
  for (int i2 = 0; i2 < n; i2++)
   for (int i3 = 0; i3 < n; i3++)
   {
    char letter1 = (char)('a' + i1);
    char letter2 = // TODO: finish this
    char letter3 = // TODO: finish this
    Console.WriteLine("{0}{1}{2}",
     letter1, letter2, letter3);
   }
```

Escaping Characters

- Escaping sequences are:
  - Represent a special character like ', " or \n (new line)
  - Represent system characters (like the [TAB] character \t)
- Commonly used escaping sequences are:
  - \' → for single quote \" → for double quote
  - \\ → for backslash \n → for new line
  - \uXXXX → for denoting any other Unicode symbol

Character Literals – Example

```
char symbol = 'a'; // An ordinary character
symbol = '\u006F'; // Unicode character code in a
                   // hexadecimal format (letter 'o')
symbol = '\u8449'; // 葉 (Leaf in Traditional Chinese)
symbol = '\''; // Assigning the single quote character
symbol = '\\'; // Assigning the backslash character
symbol = '\n'; // Assigning new line character
symbol = '\t'; // Assigning TAB character
symbol = "a";  // Incorrect: use single quotes!
```

UNIVERSITY of GREENWICH

Alliance with FPT Education

STRING

Sequence of Characters

The String Data Type

- The string data type in C#
  - Represents a sequence of characters
  - Is declared by the string keyword
  - Has a default value null (no value)
- Strings are enclosed in quotes:

```
string text = "Hello, C#";
```

- Strings can be concatenated
  - Using the + operator

Verbatim and Interpolated Strings

- Strings are enclosed in quotes "":

```
string file = "C:\\Windows\\win.ini";
```

The backslash \ is escaped by \\

- Strings can be verbatim (no escaping):

```
string file = @"C:\Windows\win.ini";
```

The backslash \ is not escaped

- You can use verbatim strings with interpolation:

```
string os = "Windows";
string file = "win.ini";
string path = $@"C:\{os}\{file}";
```

Problem: Concat Names

- Read first and last name and delimiter
- Print the first and last name joined by the delimiter

```
John
Smith
->
```

→

```
John->Smith
```

```
Linda
Terry
=>
```

→

```
Linda=>Terry
```

```
Jan
White
<->
```

→

```
Jan<->White
```

```
Lee
Lewis
---
```

→

```
Lee---Lewis
```

Summary

- Variables – store data
- Numeral types:
- Represent numbers
- Have specific ranges for every type
- String and text types:
- Represent text
- Sequences of Unicode characters
- Type conversion: implicit and explicit