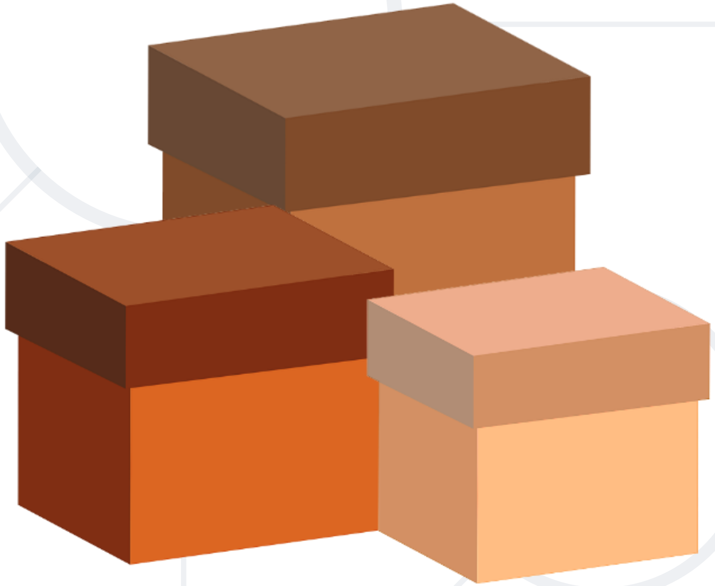
Data Types and Variables

Numeral Types, Text Types and Type Conversion



SoftUni Team

Technical Trainers

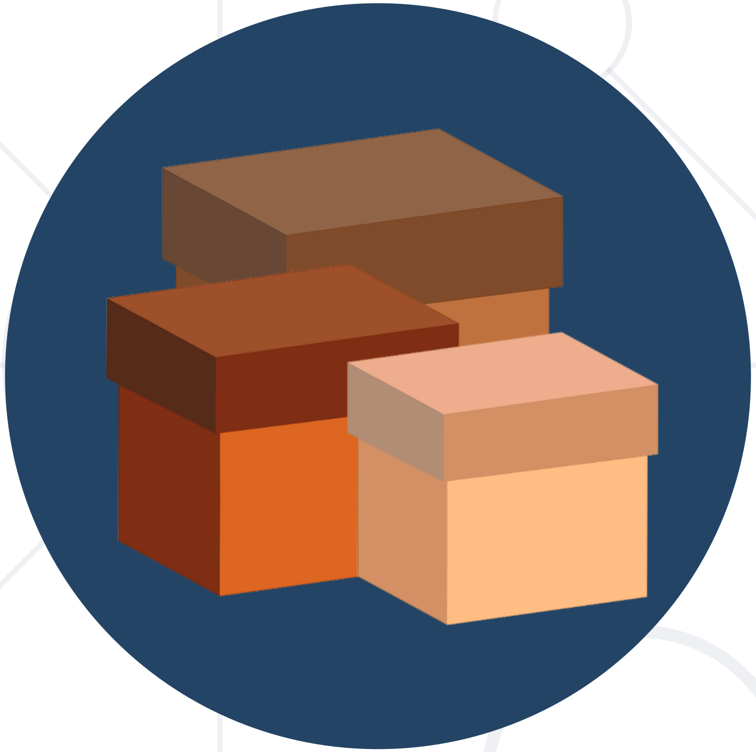
Software University

Software University

https://softuni.bg

Table of Contents

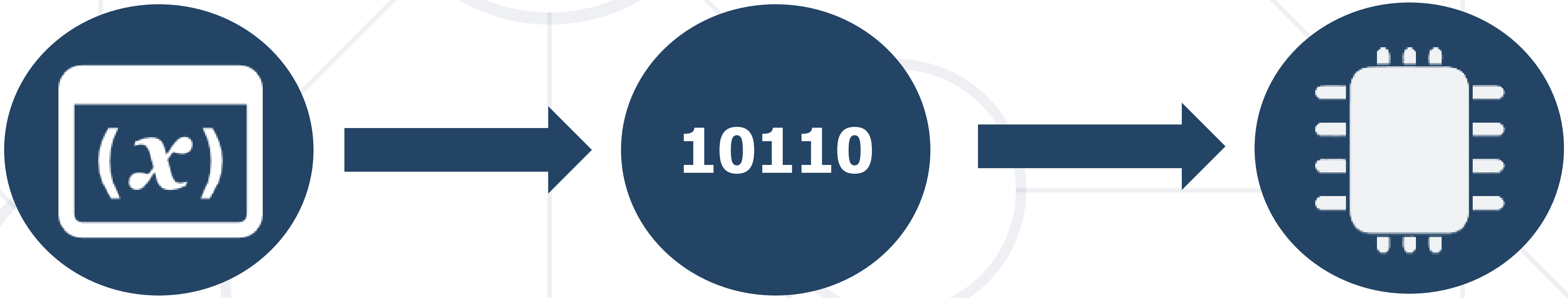
1. Data Types and Variables
2. Integer Types
3. Real Number Types
4. Type Conversion
5. Boolean Type
6. Character Type
7. String Type



Data Types and Variables

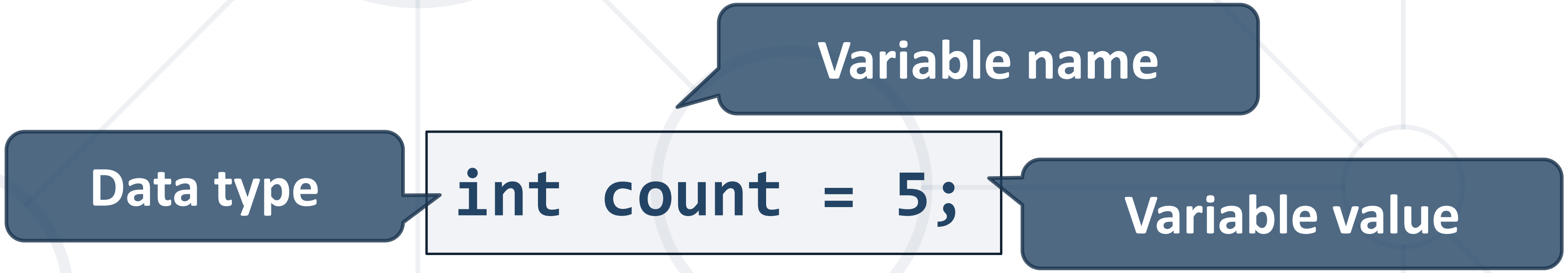
How Computing Works?

- 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#



```
int count = 5;
```

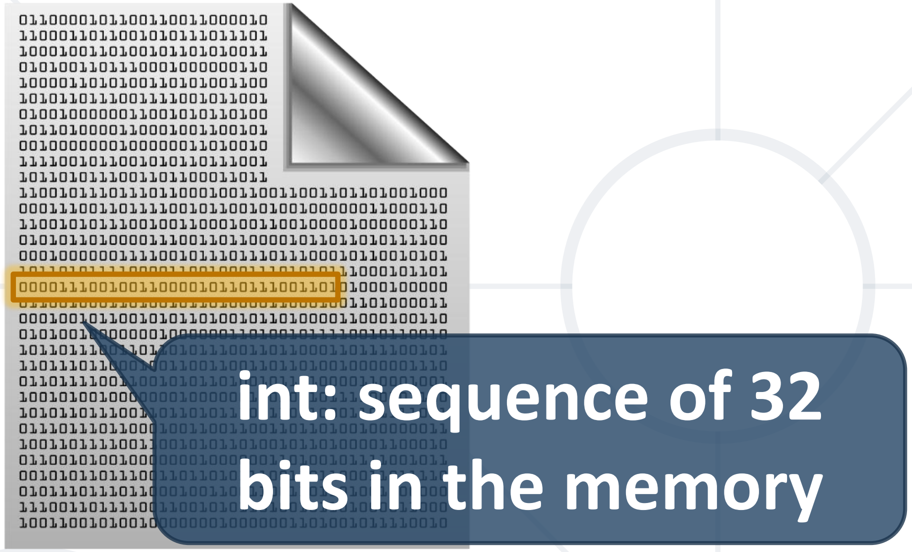
- When processed, **data** is **stored** back **into variables**

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 does this variable contain?**")

✓

```
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** == for 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
}
```

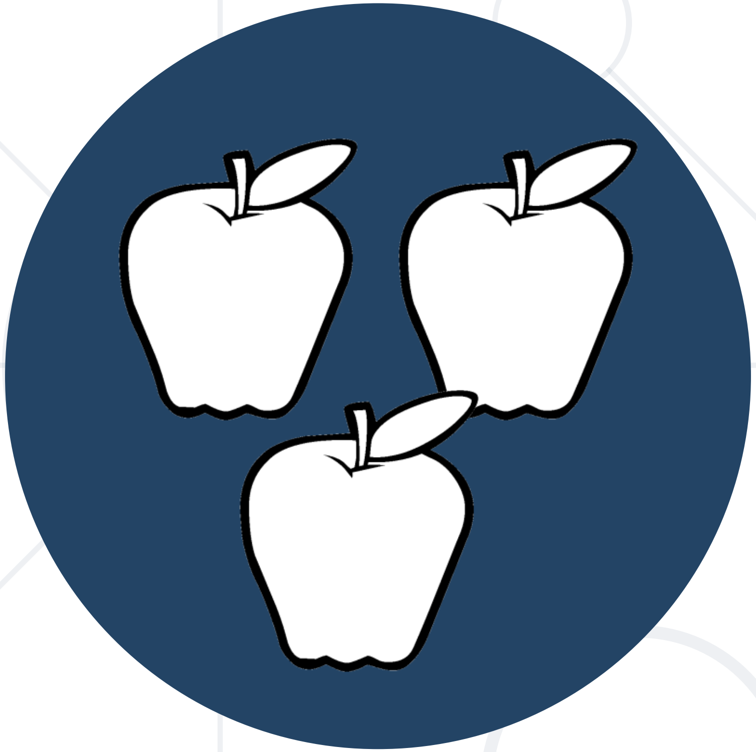
"outer" variable span

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



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 indicate a hexadecimal value
    - e.g., 0xFE, 0xA8F1, 0xFFFFFFFF
  - The 'u' and 'U' suffixes indicate a ulong or uint type
    - e.g., 12345678U, 0U
  - The 'l' and 'L' suffixes indicate long type
    - e.g., 9876543L, 0L

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 × 10^{38})
    - 32-bits, precision of 7 digits
  - `double` (±5.0 × 10^{-324} to ±1.7 × 10^{308})
    - 64-bits, precision of 15-16 digits
- The default value for floating-point types
  - `0.0F` for the `float` type
  - `0.0D` for the `double` type

PI Precision – Example

- Difference in precision when using **float** and **double**:

```
float floatPI = 3.14159265358979323 8f;
double doublePI = 3.141592653589793238;
Console.WriteLine("Float PI is: {0}", floatPI);
Console.WriteLine("Double PI is: {0}", doublePI);
```

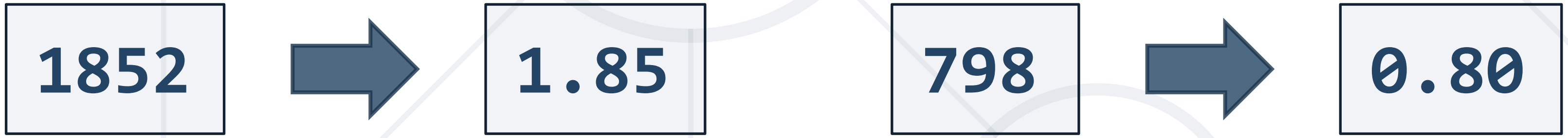
3.141593

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: Convert Meters to Kilometres

- Write a program that converts meters to kilometers formatted to the second decimal point
- Examples:



1852 → 1.85

798 → 0.80

```
int meters = int.Parse(Console.ReadLine());
float kilometers = meters / 1000.0f;
Console.WriteLine($"{kilometers:f2}");
```

Check your solution here: https://alpha.judge.softuni.org/contests/data-types-and-variables-lab/1192/practice#0

Problem: Pounds to Dollars

Software University

- Write a program that converts British pounds to US dollars formatted to 3th decimal point
  - 1 British Pound = 1.31 Dollars

```
80 → 104.800
```

```
39 → 51.090
```

```
double num = double.Parse(Console.ReadLine());
double result = num * 1.31;
Console.WriteLine($"{result:f3}");
```

Check your solution here: https://alpha.judge.softuni.org/contests/data-types-and-variables-lab/1192/practice#1

Scientific Notation

- Floating-point numbers can use scientific notation
  - 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); // 0.9999999999999906
```

Decimal Floating-Point Type

- There is a special decimal floating-point real number type in C#
  - `decimal` (±1,0 × 10^{-28} to ±7,9 × 10^{28})
    - 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
```

Check your solution here: https://alpha.judge.softuni.org/contests/data-types-and-variables-lab/1192/practice#2

Solution: Exact Sum of Real Numbers

- This code works, but makes rounding mistakes sometimes:

```
int n = int.Parse(Console.ReadLine());
double sum = 0;
for (int i = 0; i < n; i++)
  sum += double.Parse(Console.ReadLine());
Console.WriteLine(sum);
```

- Change **double** with **decimal** and check the differences

Check your solution here: https://alpha.judge.softuni.org/contests/data-types-and-variables-lab/1192/practice#2

Integer and Real Numbers

Live Exercises

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: 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

Check your solution here: https://alpha.judge.softuni.org/contests/data-types-and-variables-lab/1192/practice#3

Solution: Centuries to Minutes

```
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

Check your solution here: https://alpha.judge.softuni.org/contests/data-types-and-variables-lab/1192/practice#3

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 **1…n** print the number and whether it is special or not

```
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
```

Check your solution here: https://alpha.judge.softuni.org/contests/data-types-and-variables-lab/1192/practice#4

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
}
```

Check your solution here: https://alpha.judge.softuni.org/contests/data-types-and-variables-lab/1192/practice#4

'A'

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)

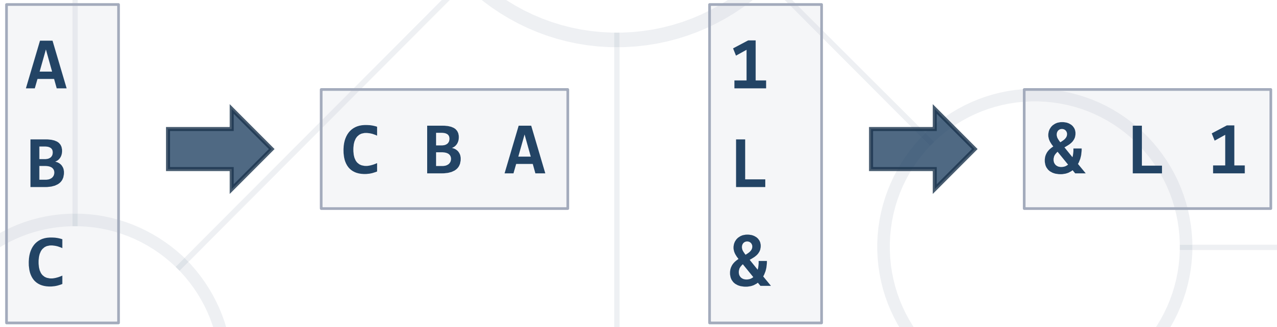
Characters and Codes

- Each **character** has a 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);
ch = 'щ';  // Cyrillic letter 'sht'
Console.WriteLine("The code of '{0}' is: {1}", ch, (int) ch);
```

Problem: Reversed Chars

- Write a program that takes 3 lines of characters and prints them in reversed order with a space between them
- Examples



A
B
C

C B A

1
L
&

& L 1

Check your solution here: https://alpha.judge.softuni.org/contests/data-types-and-variables-lab/1192/practice#5

Solution: Reversed Chars

```
char firstChar = char.Parse(Console.ReadLine());
char secondChar = char.Parse(Console.ReadLine());
char thirdChar = char.Parse(Console.ReadLine());

Console.WriteLine($"{thirdChar} {secondChar}
{firstChar}");
```

Check your solution here: https://alpha.judge.softuni.org/contests/data-types-and-variables-lab/1192/practice#5

Escaping Characters

- Escaping sequences
  - 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!
```

"ABC"

Sequence of Characters

String

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
```

Check your solution here: https://alpha.judge.softuni.org/contests/data-types-and-variables-lab/1192/practice#6

Solution: Concat Names

```
string firstName = Console.ReadLine();
string lastName = Console.ReadLine();
string delimiter = Console.ReadLine();

string result = firstName + delimiter + lastName;
Console.WriteLine(result);
```

```
Jan<->White
```

Live Exercises

Data Types

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**