

## C# Fundamentals

- C# is a strongly typed language
- the C# compiler is fairly good at finding and warning about incorrect use of types
  - and uninitialised and unused variables
  - do not ignore warnings, write code simply
  - re-write it if the compiler complains - it is most likely correct!

slide 3  
gaius

## built in data types

Type	No. of bytes	Description
byte	1	unsigned values 0-255
char	2	unicode characters
bool	1	true or false
sbyte	1	signed values -128..127
short	2	signed -32768..32767
ushort	2	unsigned 0..65535
int	4	signed $-2^{31} .. 2^{31} - 1$
uint	4	unsigned $0..2^{32} - 1$
long	8	signed long integers $-2^{63} .. 2^{63} - 1$
ulong	8	unsigned long integer $0..2^{64} - 1$

slide 4  
gaius

## built in data types

Type	No. of bytes	Description
float	4	$\pm 1.5 \times 10^{-45} .. \pm 3.4 \times 10^{38}$ with about 7
double	8	$\pm 5.0 \times 10^{-324} .. \pm 1.8 \times 10^{308}$ with 15 sig
decimal	12	fixed precision up to 28 digits

## Data types

- in general if you are starting with a blank canvas
- when designing your programs, work out the smallest datatype which will contain the largest data values you need
- sometimes library considerations and interfacing to older code might force your hand

## Data types

- in general, do not store numbers in strings
  - use constants if possible
- do not store numbers in chars

## Data types

- objects of a fundamental data types can be converted into another either
  - implicitly
    - happen automatically (the compiler does this for you)
  - explicitly
    - where you tell the compiler the rule for conversion

## Implicit conversion example

- consider the following snippet of code:

```
short x = 6;  
int y = x; // the compiler will convert the short to an
```

## Explicit conversion example

- however try this the other way around:

```
short x;
int y = 400;
x = y; // will not compile
```

- you need to explicitly convert using the cast operator

```
short x;
int y = 400;
x = (short) y; // will compile
```

## Definite assignment

- C# requires definite assignment
  - variables must be initialised before being used

```
using System;

public class TestAssignment
{
    public static void Main (string[] args)
    {
        int i;

        System.Console.WriteLine ("the value of i is {0}",
    }
}
```

- when you compile this program the compiler will generate an error message

## Simplest hello world

```
using System;

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

## C# conditional statements

- very similar to C
  - if {} else constructs

```
if (expression1)
    statement1
if (expression2)
    statement2
else
    statement3
```

## C# conditional statements

```

if (expression3)
{
    statement4;
    statement5;
}
if (expression4)
{
    statement6;
    statement7;
}
else
    statement8;

```

## C# conditional statements

```

if (expression5)
{
    statement9;
    statement10;
}
else
{
    statement11;
    statement12;
}

```

## Cascading if else statements

- often useful to cascade `if else` statements

```

if (expression1)
    statement1
else if (expression2)
    statement2
else
    statement3

```

- this can be extended to embrace multiple statements by using `{ }`

## Constants

- use constants (or enumerations) if possible
- for example if you had to write a program to determine the state of water at different Fahrenheit temperatures
- we know in Fahrenheit water:
  - boils at 212
  - freezes at 32

## Constants

```

using System;

public class WaterState
{
    public static void Main (string[] args)
    {
        const int boiling = 212;
        const int freezing = 32;
        int QueryTemperature;

        Console.WriteLine ("Enter the temperature");
        QueryTemperature = Convert.ToInt32 (Console.ReadLine ());
        if (QueryTemperature >= boiling)
            Console.WriteLine ("at this temperature water becomes stea
else if (QueryTemperature <= freezing)
            Console.WriteLine ("at this temperature water becomes ice"
else
            Console.WriteLine ("at this temperature water is a liquid"
    }
}

```

## Enumerations

- the previous program can be improved if we use an Enumeration instead of constants
  - we should use enumerations for constants which are naturally grouped together

- so we could introduce

```

enum Temperatures
{
    boiling = 212,
    freezing = 32,
}

```

## Enumerations

```

using System;

public class WaterState2
{
    enum Temperatures
    {
        boiling = 212,
        freezing = 32,
    }
}

```

## Enumerations

```

public static void Main (string[] args)
{
    int QueryTemperature;

    Console.WriteLine ("Enter the temperature");
    QueryTemperature = Convert.ToInt32 (Console.ReadLine());
    if (QueryTemperature >= boiling)
        Console.WriteLine ("at this temperature water be
    else if (QueryTemperature <= freezing)
        Console.WriteLine ("at this temperature water be
    else
        Console.WriteLine ("at this temperature water is
}
}

```

## C# for loops

- similar to C, but with some improvements

```
for (int i=1; i<=12; i++)
{
    Console.WriteLine ("8 x {0} = {1}", i, i*8);
}
```

- notice we can declare the variable `i` to live only within the `for` loop
  - why is this sensible?

## C# do loops

```
int i = 1;
do {
    Console.WriteLine ("8 x {0} = {1}", i, i*8);
    i++;
} while (i<=12);
```

- exactly the same as C

## Selection

- `switch` and `case` statements are available in C#
- a superset of C, do not abuse the superset though
- see `chartest` example in the slide after next for the syntax

## foreach statement

```
foreach (char ch in myString)
{
    Console.WriteLine ("now ch is {0}", ch);
}
```

## Program to determine vowels or consonants

```
using System;

public class chartest
{
    static bool is_vowel (char ch)
    {
        switch (ch) {

            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':
                return true;
            default:
                return false;
        }
    }
}
```

## Program to determine vowels or consonants

```
static void ProcessLine (string line)
{
    foreach (char ch in line) {
        if (is_vowel (ch))
            System.Console.WriteLine ("char {0} is a vowel",
                ch);
        else
            System.Console.WriteLine ("char {0} is a consonant",
                ch);
    }
}
```

## Program to determine vowels or consonants

```
public static void Main (string[] args)
{
    string line;

    Console.WriteLine ("Please enter string: ");
    line = Console.ReadLine ();
    ProcessLine (line);
}
}
```

## Switch statements

- in C# you can switch on string constants!
- you can also abuse switch statements
  - do not do it!
- keep switch statements simple and obvious
- the language allows you to goto a case clause, again use sparingly if at all

## Switch statements

- for example consider the following code fragment:

## Switch statements

- ```
switch (line)
{
    case "red":
        value = 1;
        break;
    case "purple":
        value = 2;
        break;
    case "yellow":
        Console.WriteLine ("yellow is deprecated, use ora
        goto case orange;
    case "orange":
        value = 3;
        break;
    default:
        value = 4;
}
```

- used sparingly like this is probably acceptable (as a fall through)

## Switch statements

- ```
switch (line)
{
    case "orange":
        value = 3;
        break;
    case "red":
        value = 1;
        break;
    case "purple":
        value = 2;
        break;
    case "yellow":
        Console.WriteLine ("yellow is deprecated, use ora
        goto case orange;
    default:
        value = 4;
}
```

- but jumping out of sequence is considered very poor programming style

## Switch statements

- notice that a major difference between C# and C/C++ is that the fall through must be done via a `goto` if the statement is present in the fall through case
- for example:

## Switch statements

```

switch (line)
{
    case "red":
        value = 1;
        break;
    case "purple":
        value = 2;
        break;
    case "yellow":
    case "orange":
        value = 3;
        break;
    default:
        value = 4;
}

```

- notice the automatic fall through for yellow and orange

## Switch statements

```

switch (line)
{
    case "red":
        value = 1;
        break;
    case "purple":
        value = 2;
        break;
    case "yellow":
        Console.WriteLine ("yellow is deprecated, use orange");
        goto case orange;
    case "orange":
        value = 3;
        break;
    default:
        value = 4;
}

```

- whereas the presence of the `WriteLine` forces us to use a `goto` to fall through. The compiler would generate an error without the `goto`

## Console input

- can be achieved by the statement:

```

{
    string line;
    int i;

    ...
    line = Console.ReadLine ();
    ...
    if (line != null) {
        Console.WriteLine ("you wrote: " + line);
        i = Convert.ToInt32 (line);
        Console.WriteLine ("as an integer its value is {0}", i);
    }
}

```

## Basic input/output via examples

```

...
Console.Write ("what is your guess? ");
line = Console.ReadLine ();
if (line != null)
    Console.WriteLine ("you wrote: " + line);
your_guess = Convert.ToInt32 (line);
...
Console.WriteLine ("{0} is too high, try a lower number", your_guess);
...

```

## Tutorial question

- write a console/command line program in C# which chooses a number between 1 and 100 and asks you to guess the number
- it should allow multiple guesses and tell you whether you are too low or too high
- it should keep a count of the number of attempts

## Tutorial question

- hint you will need
- ```
Random rnd = new Random (1234);  
int secret = rnd.Next (100)+1;
```
- to obtain a near random number between 1 . . 100

## Tutorial question

- notice that the statement
- ```
Random rnd = new Random (1234);
```
- generates the same "random" number each time the program is run!

## Tutorial question

- implement your own function called MyRandom which returns an object of type Random and takes no arguments
  - it should use the system method Random and pass the milliseconds from the time of day as its seed
- use your version of MyRandom in your guessing number game