Expressions and Casting

C# Programming

Data Manipulation

- We know that programs use data storage (variables) to hold values and statements to process the data
- The statements are obeyed in sequence when the program runs
- Remember that at this point we should be creating code to implement our solution

Simple Program

```
class Assignment {
      static void Main ()
      {
            int first, second, third ;
            first = 1 ;
            second = 2 ;
            second = second + first ;
      }
}
```

- This is a simple (and fairly useless) program

Variable Declaration

```
class Assignment {
      static void Main ()
      {
            int first, second, third ;
            first = 1 ;
            second = 2 ;
            second = second + first ;
      }
}
```

- This statement creates three variables

Variable Assignment

```
class Assignment {
      static void Main ()
      {
            int first, second, third ;
            first = 1 ;
            second = 2 ;
            second = second + first ;
      }
}
```

- The next statement assigns a value to one of the variables

Next Variable Assignment

```
class Assignment {
      static void Main ()
      {
            int first, second, third ;
            first = 1 ;
            second = 2 ;
            second = second + first ;
      }
}
```

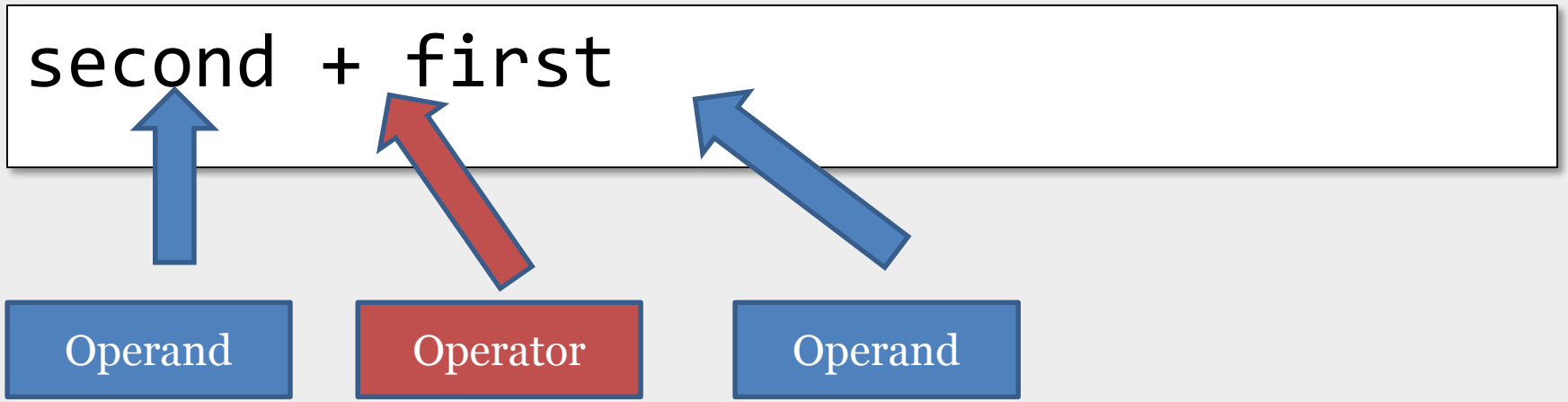
- This is another assignment

Expression Evaluation

```
class Assignment {
      static void Main ()
      {
            int first, second, third ;
            first = 1 ;
            second = 2 ;
            second = second + first ;
      }
}
```

- This assignment evaluates an expression and puts the result into the variable called second

Expressions



- An expression is made up of operators and operands

Complex Expressions

- The simplest kind of expression is a single literal value:

```
23
```

- More complicated ones involve literals, variables, operators and brackets

```
2 * ( width + height ) * 3.25
```

The Assignment Operator

```
second = second + first ;
```

- The assignment operator takes the result of an expression and puts it into a variable
- This is the fundamental means by which a program works on data

Simple Arithmetic Operators

Op.	Use
–	unary minus, the minus that C# uses in negative numbers, e.g. -1. Unary means applying to only one item.
*	multiplication, note the use of the * rather than the more mathematically correct but confusing x.
/	division, because of the difficulty of drawing one number above another on a screen we use this character instead
+	Addition.
–	subtraction. Note that we use exactly the same character as for unary minus.

Data and Type

- C# provides a range of types to store numbers
- Each type can store values in a particular range
- The compiler will not let us combine values in a way that might lose data

Dangerous Code

```
int i;
double d = 1.5;
i = d;
```

- This code will fail to compile
- The compiler is not happy to put a double precision value into an integer

Narrowing

- When you put a double value into an integer variable it won't fit:
  - The double value may have a fractional part
  - The double value may be too big to fit in the integer variable
- This is called "narrowing" and the compiler will not let a program do it

Casting

- Casting is a way that the programmer can take responsibility for a narrowing operation
- It is an explicit narrowing operation that the programmer asks to be done
- The compiler will generate code that performs the conversion

Adding a Cast

```
int i;
double d = 1.5;
i = (int) d;
```

- The cast operation is given a particular target type
- In this case we are casting the value d to an integer

Responsible Casting

- When you perform a cast you are telling the compiler that you know better than it
- You are forcing the compiler to do something it would normally not like to
- For this reason you need to be sure when you cast that it is sensible to do so
- Otherwise you will break your program

Casting Literals

```
float x;
x = (float) 3.14;
```

- You can use casting to convert literals into particular types in your program
- The cast works on the value immediately to the right of the cast type

Limited Casting Powers

```
int i;
string s;
i = 99;
s = (string) i;
```

- You can't use casting to convert from integer to string (or back)
- It only works between numeric types

Types in Expressions

- We have seen that the result produced by an operator depends on the items it is working on
  - + can add integers or concatenate strings
- Now we are going to explore how this effects the way that expressions are worked out

Integer Division

```
double d;
d = 1/2;
Console.WriteLine ( "d is : " + d ) ;
```

- This happens because the compiler uses a version of the division operator that matches the operands
- Integer values use integer division

Forcing double Division

```
double d;
d = (double) 1/2;
Console.WriteLine ( "d is : " + d ) ;
```

- The compiler will generate a double precision division if one of the operands is a double precision one
- We can do this by casting

Good Casting

- It is said that good casting makes a movie much better
- I think this is true of programs too
- I often add the casts so that it is clear what is going on, even if the compiler doesn't need them

Summary

- Expression evaluation is how data is processed by a program
- The evaluation is performed by operators
- A program will not be allowed to "narrow" data unless an explicit "cast" is given
- Casting can also be used to determine which operator is used in an expression