CO452 Programming Concepts

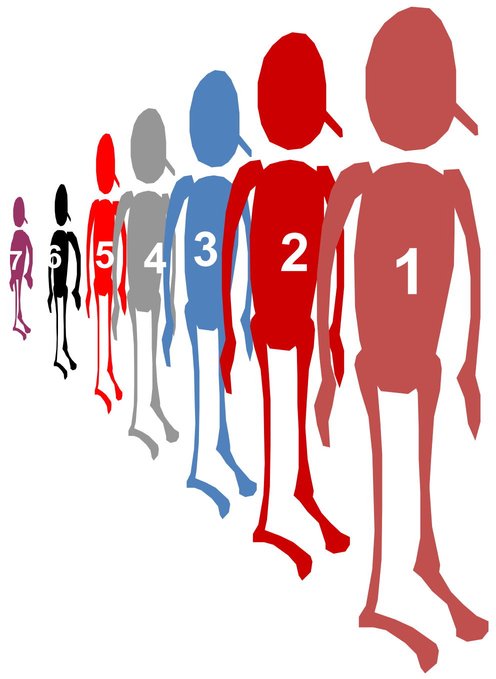
Week 11 - C# Part 2

Sequence, Selection, Iteration

Sequence

The first basic structure of all programs

Last Week: Sequence



The **sequence** is :

- a set of statements that are performed one after the other
- the order is important

In C#, we put sequence statements inside a block using { and } brackets

Indent within a block

```
{
    Console.WriteLine( "Hello\n" );
    Console.Write( "What town do you live in?");
    town = Console.ReadLine();
    Console.WriteLine( town + " is a nice place" )
    Console.WriteLine( “Enjoy programming" );
}
```

Iteration

(loops)

The second basic structure
of all programs

The 3 types of loop used in C#

while loop

```
while ( condition )
{
    // repeated instructions here

}
```

for loop

```
for ( initialise; condition; increment)
{
        // repeated instructions here
}
```

do while loop

```
do
{
    // repeated instructions here
}
while ( condition ) ;
```

The <u>break</u> statement

Give us a break

break is used to break out of a
switch statement
(or a loop)

prematurely
and continue with the rest
of the program.

(see the switch example)

The continue statement

May I continue?

<u>continue</u> can be used in <u>loop</u> statements to skip over part of the loop and continue to the next repeat

<u>Example</u>

Below is part of a program that performs calculations on a series of 100 numbers. However, <u>continue</u> is used to prevent this for the numbers 25 to 50.

```
for (   n = 1  ;  n <= 100  ;   n++   )
{
        if ( n >= 25   &&   n <= 50 )   continue ;
          // otherwise do the rest of loop

}
```

While Loop

```
//         Author :  Brian Ward
//         Date   :  7th Sep 2007

static void Main ()
{   int mark, count = 0, total = 0 ;
    double average ;
    const int MAX = 8 ;

    Console.WriteLine( "Average Mark Calculation\n" )
    Console.WriteLine( "Input Exam Scores Now " );

    while (count < MAX)
    {
       count ++ ;
       Console.Write( "Enter mark for student " + count);
       mark = Convert.ToInt32(Console.ReadLine());
       total += mark ;        // or total = total + mark
    }

    average = (double) total / MAX ;
    Console.WriteLine( "Average mark = " +
   average );
}
```

Note the use of a cast here

For Loop

```
static void Main ()
{
        int  year;                               //   year counter
        double  value, gain;                     // share value and gain
        const   int  MAX_YEARS = 4;
        const   double  INCREASE = 0.1;
        Console.WriteLine( " \t\t Shares Value Calculation\n ")
        Console.Write( "\t\t Input current Value of shares : " );
        value = Convert.ToDouble(Console.ReadLine()) ;

        Console.WriteLine( "YEAR  \t  GAIN  \t  VALUE" );

        for ( year = 1;  year <= MAX_YEARS;  year++ )
          {
               gain = value * INCREASE;
               value = value + gain;
               Console.WriteLine( year + "\t" + gain
                                              + "\t" + value );
          }

      Console.WriteLine( "\nAfter " + MAX_YEARS +
              " years, your shares will be worth £"  + value
}     ) ;
```

Selection

The third basic structure of all programs

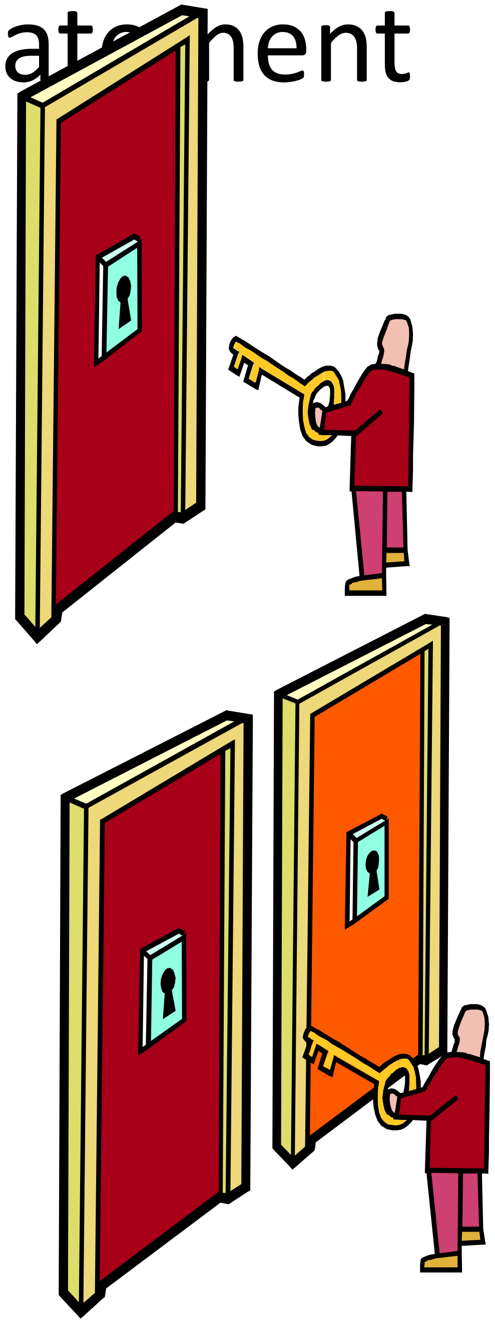
The 2 types of if() statement

if(..) statement

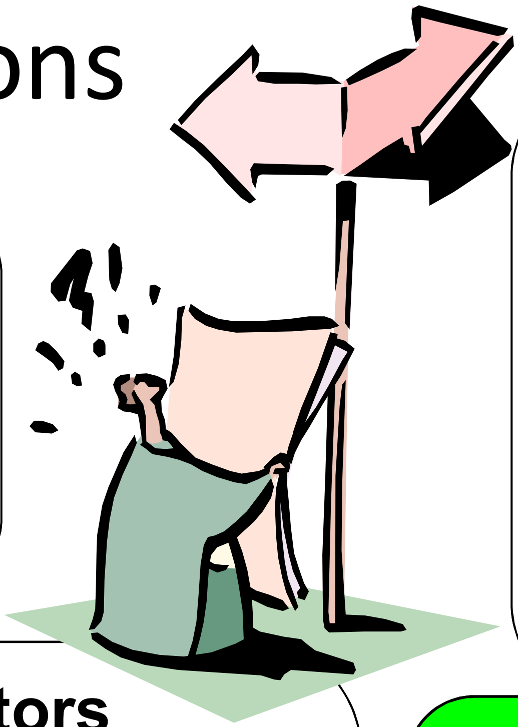
```
if ( condition )
{
        // instructions here done once
        // only if the condition is true
}
```

if (..) else statement

```
if ( condition )
{
        // done if condition TRUE
}
else
{
        // done if condition FALSE
}
```



Conditions



How does the program know which path to take?

- The path taken depends on a <u>condition</u>
- If the condition is <u>true</u>, one path is taken
- If it is <u>false</u>, another is followed

<u>Relational Operators</u>

used in true/false conditions:

<	Less than
>	Greater than
<=	Less than or equal
>=	Greater or equal
==	Equal to
!=	Not equal to

<u>Some Examples</u>

$$\begin{array}{l} \text{if } \underbrace{\begin{array}{l}\textbf{(count == 10)} \ \ldots \\ \textbf{(age >= 65)} \ \ldots \\ \textbf{(money < 5000)}\ldots\end{array}}_{\textbf{conditions}} \end{array}$$

Relational Operators

Relational operators

==   !=   <   >   <=   >=

used in conditions (see earlier slide)

Example

```
if (mark < 40)
{
        Console.Write( "You failed the exam" );
        failcount ++ ;
}
```

Logical Operators

<u>Logical (or Boolean) operators</u>

&& ***(and)*** **||** ***(or)*** **!** ***(not)***

Used to combine 2 or more conditions

Some examples :

```
if  ( price <= 50  &&  size = 38)  …
if  (choice == "A" || choice == "B") …
if  ! (mark <= 100 && mark >= 0) …
```

<u>Example</u>

```
if (age >= 13 && age <= 19)
{
     Console.Write( "You are a Teenager" );
     teencount ++ ;
}
```

The Switch Statement

for multiple selection

Example Switch Program

```
static void  Main()
{
     string grade ;                      // exam grade
     Console.WriteLine( " Exam Comments " );
     Console.Write( " Enter grade achieved " );
     grade = Console.ReadLine();
     grade = grade.ToUpper() ;            // convert to upper case
     switch  (grade)
     {
          case "A" :   Console.WriteLine( " Excellent Result " );   break ;
          case "B" :   Console.WriteLine( " Very Good " );  break ;
          case "C" :   Console.WriteLine( " Well Done " );  break ;
          case "D" :   Console.Writeline( " You Passed " ); break ;
          case "E" :   ;                  //   falls through to the next case
          case "F" :   Console.WriteLine( " Sorry You Failed ! " );  break;
          default :  Console.WriteLine( " Error .. Unrecognised grade " );
                                                              break ;
     }
}
```

```
static void Main ()
{
        int  month ;                              // number of the month
         Console.WriteLine( " The Seasons of the Year " );
         Console.Write( " Enter the month number (1 TO 12) " );
         month = Convert.ToInt32(Console.ReadLine());
            switch (month)
            {
                    case 11: case 12: case 1: case 2:
                            Console.WriteLine( " It is Winter " );        break;
                    case 3: case 4: case 5:
                            Console.WriteLine( " It is Spring " );        break;
                    case 6: case 7: case 8:
                            Console.WriteLine( " It is Summer ") ;        break;
                    case 9: case 10:
                            Console.WriteLine( " It is Autumn ") ;        break;
                    default:
                            Console.WriteLine( " Error in month number " );
                                                                  break;
            }
}
```

default is done for other cases that don't match

How to design a program (9 step process)

Designing & writing programs: Summary

The 9 Step Process

<u>Preparation Stage</u>

1. **Understand the Problem**

   In the real world:

   Analyse problem, establish requirements

<u>Design Stage</u>

2. **Input-Output Diagram** (ins and outs)
3. **Identifier List** (names of variables)
4. **Algorithm** (method of solution)
5. **Test Plan** (data for later tests + expected results)

<u>Implementation Stage</u>

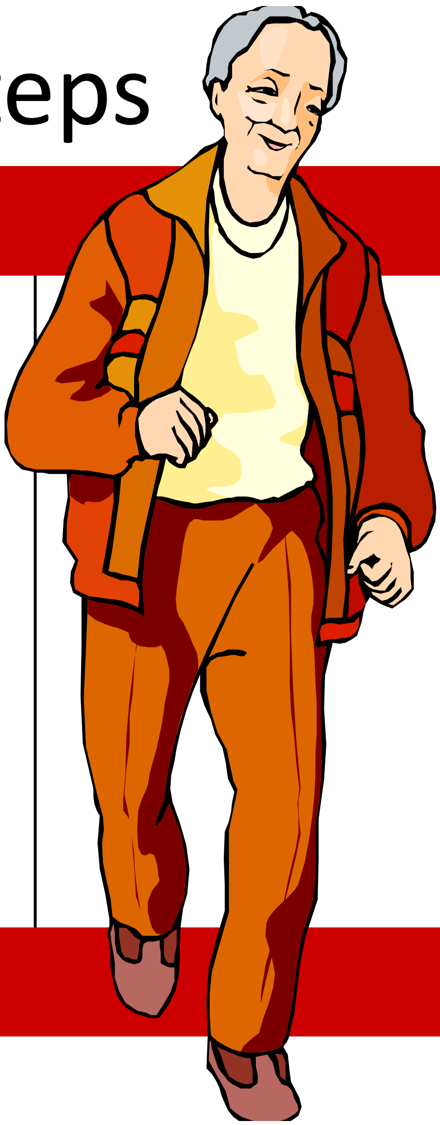
6. **Source Code** (write C#)
7. **Compile** (translate C# into machine code)

<u>Testing Stage</u>

8. **Run the program**
9. **Test using Test Plan data**

   (recording results and correcting code)

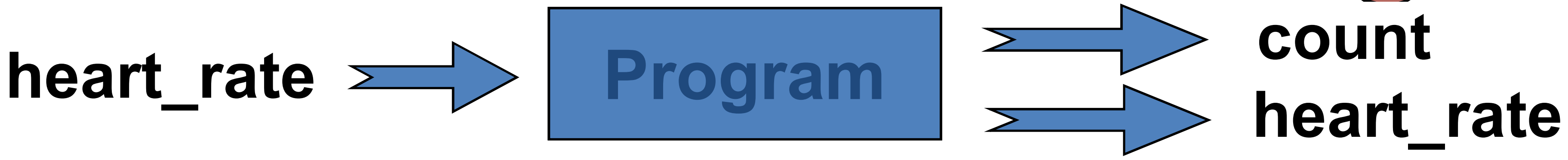
A full example, using the 9 steps



1: Understand the Problem

Brian is going to run round a track, but he is concerned about his heart. He should only run **while his heart rate is less than 130**.
We will check his heart rate **before the start**, to make sure that it is OK to begin and check again at the **end of each lap** to see if it is OK to do another. He should stop as soon as his heart_rate reaches 130. We will output how many laps he completed.

2: Input-Output Diagram



3: Identifier List

Identifier	Type	Meaning
heart_rate	int	Heart rate
count	int	Count of laps done

4: Algorithm

```
1.  set count to zero
2.  Input heart_rate
3.  while heart_rate < 130
        a.  add 1 to count
        b.  Output current count
        c.  input heart_rate
    end_while
4.  Output count
5.  Output heart_rate
```

5: Test Plan

Test No	Inputs	Expected Output	Actual Output
	heart_rate	count	count
1	80 85 100 125 140	4	
2	140	0	
3	100 120 130	2	
4	80 90 110 129 128 131	5	

Results are entered later, in step 9

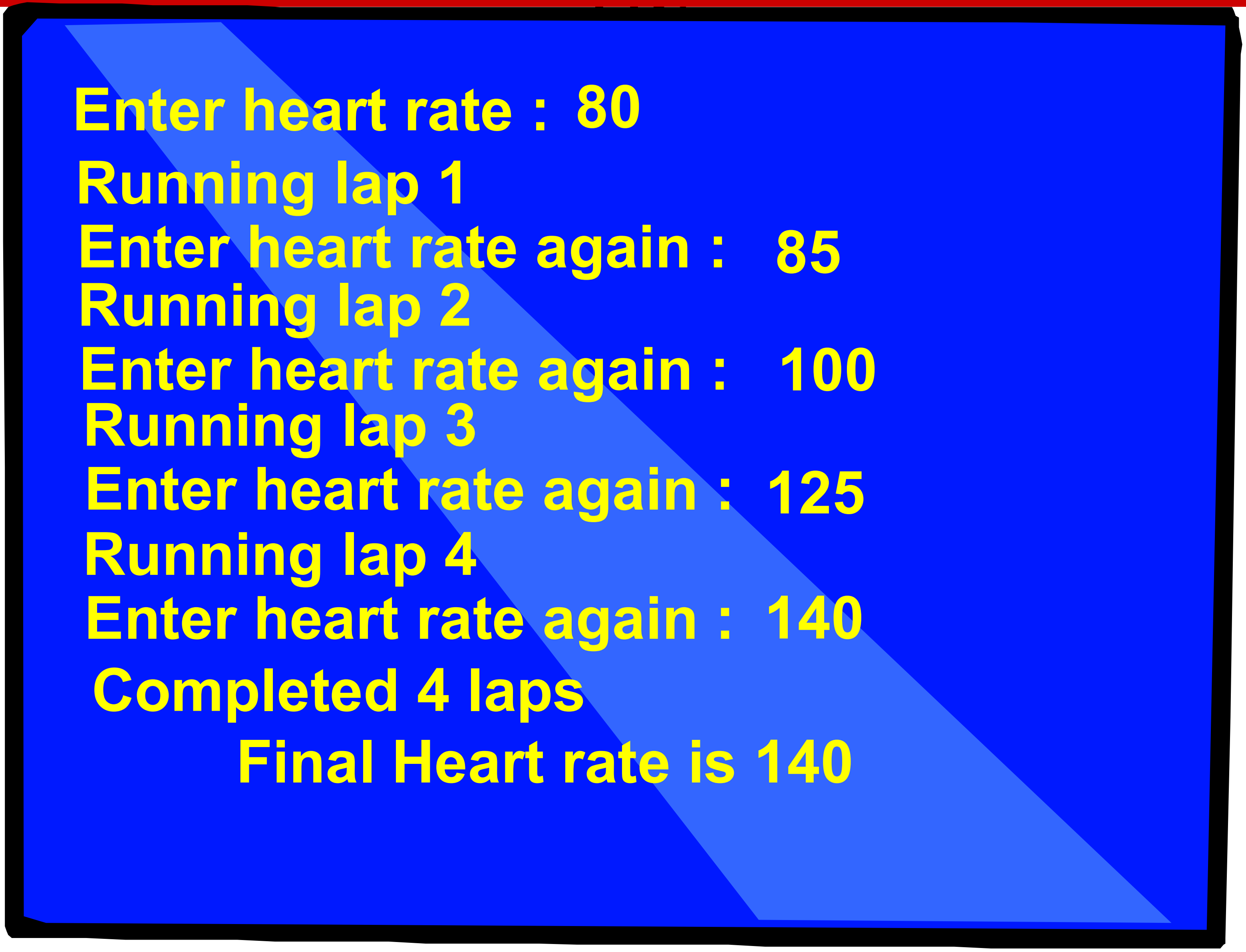
6: Source Code

```
static void Main ()
{
        int  count = 0;                 //  initialise lap counter to zero
        int  heart_rate;                 //  runner's heart rate
        string input;
        Console.Write( " Enter heart rate : " );
                input = Console.ReadLine();
                heart_rate = Convert.ToInt32(input);
        while (heart_rate < 130)
        {
                count++;
                Console.WriteLine( "Running lap " + count ) ;
                Console.Write( " Enter heart rate again : " );
                input = Console.ReadLine();
                heart_rate = Convert.ToInt32(input);

        }
        Console.WriteLine( "Completed " + count + " laps" );
        Console.WriteLine( "Final Heart rate is " + heart_rate ) ;
}
```

This is an example of the **Read-Ahead** technique

7 and 8 : Compile and Run the Program



```
Enter heart rate : 80
Running lap 1
Enter heart rate again :  85
Running lap 2
Enter heart rate again :  100
Running lap 3
Enter heart rate again : 125
Running lap 4
Enter heart rate again : 140
Completed 4 laps
        Final Heart rate is 140
```

9 : Test the Program

Test No	Inputs	Expected Output	Actual Output
	heart_rate	count	count
1	80 85 100 125 140	4	4 ✓
2	140	0	0 ✓
3	100 120 130	2	2 ✓
4	80 90 110 129 128 131	5	5 ✓

More than One Condition (using &&)

Sometimes we want more than one condition to control a loop.
In the lap-running program, we may want to check our heart_rate, but also do no more than 20 laps

The **&&** *(and)* means both conditions must be **TRUE** to keep repeating the loop.
As soon as one is **FALSE** the loop will stop.

```
Console.Write( "Enter heart rate : " );
input = Console.ReadLine();
heart_rate = Convert.ToInt32(input);
while ( heart_rate < 130  &&  count < 20 )
{
        count++;
        Console.WriteLine( "Running lap " + count);
        Console.Write( " Enter heart rate again : " );
        input = Console.ReadLine();
        heart_rate = Convert.ToInt32(input);
}
Console.WriteLine("Completed " + count + " laps");
Console.WriteLine("Final Heart rate is " + heart_rate);
```

The Last Slide

Extra Reading

Why are there 2 Equals?

Equals (=)

This is the assignment operator.
It is used to <u>change</u> the value of a variable e.g.

```
total = num1 + num2 ;
```

total is changed

Equals (==)

This is a relational operator.
It is used in conditions.
<u>Nothing is changed</u> e.g.

```
if (total == 100) ….
```

total stays the same

Initialising Variables

Variables can be given an initial value at the same time as they are declared

e.g.

```
int  count = 0 ;
float price = 7.54 ;
string name = "Joe Smith" ;
```

Multiple Assignments

Multiple variables of the same type can be assigned the same value

e.g.

```
a = b = c = 8 ;
price1 = price2 = 7.54 ;
adult_count = child_count = 0 ;
```

Operators

Relational Operators

Relational operators

== != < > <= >=

used in conditions (see earlier slide)

Example

```
if (mark < 40)
{
        Console.Write( "You failed the exam" );
        failcount ++ ;
}
```

Logical Operators

Logical (or Boolean) operators

&& *(and)* || *(or)* ! *(not)*

Used to combine 2 or more conditions

Some examples :

```
if  ( price <= 50  &&  size = 38)  …
if  (choice == "A" || choice == "B") …
if  ! (mark <= 100 && mark >= 0) …
```

Example

```
if (age >= 13 && age <= 19)
{
      Console.Write( "You are a Teenager" );
      teencount ++ ;
}
```

Arithmetic Operators

normal arithmetic operators

+ - * / ***and* %** *(used for remainder (modulo) division)*

The Increment operator (++)

Instead of :

```
count = count + 1 ;
        // add 1 to count
```

we can do :

```
count ++ ;
```

The Decrement operator (--)

Instead of :

```
count = count - 1 ;
// subtract 1 from count
```

we can do :

```
count -- ;
```

Example

```
if (mark >= 40)
{
    Console.Write( "You passed the
    exam" );
    passcount ++ ;
}
```

Arithmetic assignment operators

These are : += -= *= /= *and* %=

+=

Instead of :

```
num = num + 3 ;
        // add 3 to num
```

we can do :

```
num += 3 ;
```

-=

Instead of :

```
x = x - y ;
        // subtract y from x
```

we can do :

```
x -= y ;
```

***=**

Instead of :

```
gain = gain * 1.10 ;
        // multiply gain by 1.10
```

we can do :

```
gain *= 1.10 ;
```

/=

Instead of :

```
num = num / 2 ;
        // divide num by 2
```

we can do :

```
num /= 2 ;
```

The arithmetic and the reassignment are done in one statement

Alternative Input method

Instead of :

```
input = Console.ReadLine();
number = Convert.ToDouble(input);
```

We can use one statement :

```
number = Convert.ToDouble(Console.ReadLine());
```

Alternative input methods

```
input = Console.ReadLine();

then
number1 = int.Parse(input);
or
number2 = float.Parse(input);
or
number3 = double.Parse(input);
```