SCHOOL OF COMPUTER SCIENCE AND ELECTRONIC ENGINEERING

PRIFYSGOL
BANGOR
UNIVERSITY

Programming Fundamentals

Laboratory 4
Control Constructs

Dave Perkins

Introduction

This laboratory session explores the use of the three fundamental programming constructs covered in the lectures:

- Sequencing;
- Selection;
- Iteration (known informally as *looping*).

Attempt as many exercises as possible. Make sure that you save all your laboratory work in an appropriately named folder (e.g. **Lab4**).

Exercise 1: Doing Things in Order

Write a program to perform the following tasks in the *order* specified.

1. Read a positive integer supplied by the user into a variable called **`originalNo`**
2. Set another variable called **`magicNo`** to **`originalNo`**
3. Decrement **`magicNo`** by one
4. Multiply result by 3
5. Add 12 to **`magicNo`**
6. Divide result by 3;
7. Add 5 to **`magicNo`**
8. Subtract original number from **`magicNo`**
9. Display the original number and the magic number.

If this has been done correctly you should find that whatever original integer value is supplied, the magic number is always eight!

Exercise 2: Using Assignment to Swap Values

Write a program that swaps the values of two integer variables **`x`** and **`y`**. The variables receive their initial values from the user who enters the values via the keyboard. Thus if the user sets **`x`** equal to **1** and **`y`** equal to **99** then when the program terminates **`x`** should hold the value **99** and **`y`** the value **1**. The new state of the variables should be displayed on the console screen.

This program is not long and should contain no more than about ten lines (excluding comments).

Exercise 3: Smallest and Largest (Min and Max)

The program begins by prompting the user for two integer values. The program will detect which value is the largest input and which is the smallest and display this information on the screen. If both inputs are equal this should be detected and an appropriate message generated.

Once the program is working extend it so that three integer inputs can be handled. The largest and the smallest values should be detected and displayed on screen. In the special case of all the inputs being equal an appropriate message should be generated. Test both programs with an appropriate range of input values.

Exercise 4: Odd or Even?

Write a program to read a *positive* integer from the keyboard and determine whether it is odd or even. The program should behave as follows

```
Enter number: > 99

The number you entered is odd.
```

Once this is working for a single input place the code inside a **`while`** loop so that the user can repeatedly enter numbers for testing. To exit the program the user enters the *sentinel value* -1.

Exercise 5: Grading by Computer

In this exercise *statement selection* is used to grade a student exam on the basis of the numerical mark the student obtained. The mark is entered by the user. The grading system is specified below:

Mark	Grade
90 – 100	A
80 – 89	B
70 – 79	C
60 – 69	D
40 – 59	E
< 40	F

The mark and grade awarded should be displayed when the program terminates. If a value outside the closed interval [0,100] has been entered then an error message should be generated.

Thoroughly test your program.

Now extend the program so that it can *repeatedly* read a mark from the user and display the associated grade. Hint: use a **`while`** loop with a sentinel value of 999.

Exercise 6: Generating and Summing Sequences

Write **`while`** loops to display the following sequences

- Integers from 1 to 100 inclusive
- First fifty positive even integers
- First fifty positive multiples of five (i.e. 5, 10, 15, 20, …)
- Negative integers from -1 to -1000 inclusive
- ASCII code values for the characters A-Z inclusive
- First 100 terms in the sequence 1, 11, 16, 26, 31, 41, 46, …

Also, can you write loops to calculate the *sums* of the first three sequences above? In each case test your result by doing a little mathematics. For a discussion of sequence summation formulae visit the site below:

http://www.mathsisfun.com/algebra/sequences-sums-arithmetic.html

Exercise 7: Counting to a Million (Challenge Problem)

Create, compile and run the following program which counts up to a million. Before you actually run the program make a guess as to how many seconds it will take to complete.

```
int main()
{
      int count = 0;
      while (count < 1000000)
            count++;
      print("Counter = %d", count);
}
```

Now try amending the program so that each time the **`count`** variable is incremented its value is displayed on screen. What happens to the running time now? Finally try repeating these two steps but this time count to a billion (i.e. 1,000 million). Can you explain the results of these two experiments?

Exercise 8: Ignoring Case

Write a program which asks the user if they wish to continue. If so, the user must enter either ‘ Y’ or ‘y’ (either lower case or uppercase).

```
Do you wish to continue? [Y/N] :> Y
Continuing …
Do you wish to continue? [Y/N] :>
```

However, if the user wishes to stop they must type either ‘N’ or ‘n’ :

```
Do you wish to continue? [Y/N] :> n
Terminating!
```

In this case the program should terminate.

If the user types a character other than the set of valid inputs they should be warned and prompted again.

```
Do you wish to continue? [Y/N] :> x
Invalid input, try again
Do you wish to continue? [Y/N] :>
```

Exercise 9: Classifying Numbers (Challenge Problem)

There are many different types of numbers e.g. real numbers, rational numbers, integers and so forth and students sometimes get confused when asked to classify a particular number. Write a program which prompts the user for an integer and then automatically generates a classification based on the table below:

Set	Name	Symbol
0,1,2,3, …	Natural numbers[1]	$\mathbb{N}$
… ,-2,-1,0,1,2	Integers	$\mathbb{Z}$
1,2,3,4, …	Positive integers	$\mathbb{Z}^+$
… ,-2,-1,0	Non-positive integers	
… -3 -2,-1	Negative integers	$\mathbb{Z}^-$
0,1,2,3, …	Non-negative integers	

Figure 1 Number Sets

[1] Mathematicians cannot agree whether zero counts as a natural number! For example, John von Neumann who was both a mathematician *and* an early computer scientist classified zero as natural number but Giuseppe Peano regarded the naturals as starting from one. In the end the choice is yours but make sure you make it clear how *you* are defining the set.

Notice that these categories are not always mutually exclusive: thus a number such as 3 is a natural number, an integer, a positive integer and a non-negative integer.

One other point to note is that zero is neither positive nor negative – it is said to be unsigned. The behaviour of the program is indicated below:

```
Number Classifier
*****************
Enter a number: 3

The number you entered is a natural number. It is also an integer, a
positive integer and a non-negative integer.

Do you want to classify another number? (Y/N): N

Thanks for using Number Classifier
```

Thoroughly test your program, and in particular make sure it works for the value zero. Also check that all of your classification messages are grammatically correct.

A possible extension for this program might involve error checking: for example, what should happen if the user enters a sequence of characters which do not represent an integer?

```
Number Classifier
*****************
Enter a number: 3.5
```

Error checking is a complex topic, so be warned!

Assessed Laboratory Work

Iteration: Again and Again and Again …

Write a program containing eight **`while`** loops to generate the following sequences of integer values:

(a) 0 to 9 inclusive

(b) 99 to 50 inclusive

(c) Even numbers between 50 and 100 inclusive

(d) First 20 numbers in the arithmetic sequence 0, 8, 16, 24, . . .

(e) First 10 numbers in the sequence 1, 2, 4, 8, …

(f) First 10 numbers in the sequence 1, 2, 0, 3, -1, 4, -2, …

(g) First 10 triangle numbers

See https://www.mathsisfun.com/algebra/triangular-numbers.html

(h) First 12 numbers in the sequence 1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16 … (Appendix 2)

The solutions must be stored in a single source file called **whileTester.c**. Your program must contain screen messages indicating what each loop is trying to do. Your output must therefore closely resemble the sample output below (notice the use of commas to separate sequence elements). Also do check for each solution that you generate the correct *number* of elements.

```
WhileTester running ...
Loop to print values 0 to 9 inclusive
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Loop to print values 99 to 50 inclusive
99, 98, 97, ... 53, 52, 51
...
```

The following points should be borne in mind:

- Use a single **`while`** loop for each sequence.
- Do not create a large numbers of variables (you will be penalised).
- Keep the code as simple as possible.

The **only** file you should submit via Blackboard is w**hileTester.c**

Submission Notes and Marking Scheme

Use **Blackboard** to submit your source code files. The deadline for submission will be published on Blackboard. Late submissions will be penalised in line with School policy.

Marks for this laboratory exercise are awarded as follows:

- Generating correct sequences **80% (10% for each sequence)**
- Use of variables and variable names **10%**
- Code layout and comments **10%**

Students who submit work but have a poor understanding of what has been submitted may be heavily penalised.

When submitting work it is your responsibility to ensure that all work submitted is

- Consistent with stated requirements
- Entirely your own work
- On time

Please note that there are **severe penalties** for submitting work which is not your own. *If you have used code which you have found on the Internet* or *from any other source* then you **must** signal that fact with appropriate program comments.

Note also that to obtain a mark you must attend a laboratory session and be prepared to demonstrate your program and answer questions about the coding. Non-attendance at labs will result in your work not being marked.

Appendix 1: Factorial Function

The factorial function is well known to mathematicians and is easy to define. Thus, the factorial of the positive integer ***n*** which is written as ***n!*** is the product of all positive integers less than or equal to ***n***.

For example, 5! (pronounced “five factorial”) is equal to

5 x 4 x 3 x 2 x 1 = 120

and 3! is equal to

3 x 2 x 1 = 6

Note that there are two special cases

0! = 1

and

1! = 1

The factorial sequence therefore consists of the numbers

1, 1, 2, 6, 24, 120, ...

****

Appendix 2: A Tough Sequence

The sequence P(0), P(1), P(2), P(3) … P(n) is defined by the following rules:

P(0) = 1;

P(1) = 1;

P(2) = 1

P(n) = P(n-2) + P(n-3)

These apparently difficult rules are in fact quite simple. The first rule says that the first term in the sequence is always 1, the second rule says that the second term in the sequence is always 1 and the third rules says that the third rule in the sequence is always 1.

The last rule says that the nth term in the sequence, where n > 2, is equal to the sum of the two preceding terms P(n-2) and P(n-3).

For example, the term P(7), which is equal to 4, is equal to the sum of P(5) and P(4), namely 3 + 3.

The sequence of the first twenty terms is listed below:

1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, 37, 49, 65, 86, 114, 151

****