Chapter 3 – Iteration

By the end of this chapter you should:

- Understand the syntax of the 3 different loop structures
- Be able to choose which one is more suitable for a given program

Key definitions for the end of lesson:

- **Iteration**- a line/ section of code that repeats
- **Condition controlled** – both WHILE loops; they stop when a certain condition is met/ is no longer met.
- **Count controlled** – a FOR loop – stops when a set value is reached. Used when you know how many times a loop will run/ it will run a fixed number of times. Have a counter built in.
- **Infinite loop** – when you write a loop that can't end!

Condition controlled loops - WHILE

```
//This program will only stop looping when one
suitable age has been typed in

main()
{
   int  error=0, age=0;

   while (error != 1)
   {
        fflush(stdin);
        printf("Please enter your age");
        error = scanf("%d",&age);
    }
    printf("You have %d years til 100", 100-age);
}
```

- Anything inside the curly brackets belongs to the loop and will happen each loop
- The condition **while (error!= 1)** means that the loop should continue *while the variable error is not equal to* 1
- The condition is checked **at the top** of the loop
- The first time into this loop error is 0 (as it was set to that at the top) so the loop will be carried out and each statement inside will be executed
- We flush the input buffer at the top of each loop otherwise it might have old data in it
- The **error** variable stores the return value from the **scanf** function – which will only return 1 if one correct number has been typed & read in
- If a correct age was input, the program loops, and checks the WHILE condition. In this case, it is met & it exits the loop and continues with the rest of the code

Task 1

- To get an idea of how this works, copy the code from example 1 into a program of your own and compile it and run it.
- Try entering a letter or punctuation and see how it loops until you enter a suitable digit

- If you do end up in an infinite loop that won't close, press **CTRL + Break**

- Now – edit your number guessing program from Chapter 2 (Exercise 1) so that it uses a WHILE loop to do the following:
  Requirements:
  - If they guess correctly the loop should end and a the "Well done" message is output
  - If they haven't guessed correctly the program should output a message telling the if their guess was too high or too low, and ask them to try again
  - The user should get a maximum of 5 guesses at the number – if they have not guessed correctly after 5 tries, they get a "Sorry, you lose!" message

Condition controlled loops - DO WHILE

```
//This program will only continue after the correct
passcode has been typed in and matches 3880

main()
{
  int  correct = 3880, passcode=0;

  do {
      fflush(stdin);
      printf("Please enter passcode");
      scanf("%d",&passcode);

   } while (correct !=  passcode);

   printf("You entered the correct code. \n");
}
```

- In a DO loop, the condition comes **at the end** of the loop code
- Notice that the condition line, **DOES** have a semi-colon at the end
- We use a DO loop when we always want to execute a loop at least once. If we used a WHILE loop, the condition is checked at the top and if the condition is true to begin with, the loop will never run
- These loops are used **when you always want to make sure that they run at least once**
- Mostly, it doesn't matter which WHILE loop you choose

Counting iterations

```
int num = 0, x = 0, y = 0;

do
{
   printf("Enter a number, 1 to 10:");
   fflush(stdin);
   scanf("%d",&num);
   x ++;
   y = y + 5;
 } while (x!=5 && y!=25);

printf("You entered %d \n", num);
printf("X = %d \n", x);
printf("Y =  %d \n", y);
```

- This program makes the user enter a number 5 times, until the values of X and Y are equal to 5 .
- There are 2 different ways you can write a count of how many times your WHILE or DO loops actually iterate:
  - **X++ adds one to the x variable on each time through this loop**
  - **Y= Y+5, adds 5 to itself each time the loop runs**
- Each time the loop runs, either the 1 or 5 value is added to the total of X and y
- In reality you would only use one of these methods: X is fine and quicker as a counter, Y is the way you could do it if you want to **increment** by more than 1 each time
- You can also use X - - or y = y -1 to **decrement**

Task 2

- Write a program that:
    - Asks the user how many items of data they want to enter
    - Enters a DO…WHILE loop to ask the user to enter their numbers
    - The loop should stop when they have entered the correct amount of numbers
    - The program above should work out the average value of the numbers they entered and display this to the user after the loop has finished:
        - You need to keep a running total of the numbers entered
        - Think about what values you have to calculate the average
    - Outputs a message telling the user the average of the numbers they typed in

NB: you do not need to store the individual data values

Task 3

A 'magic number' is one which can be divided by 7 or 3 and if then multiplied by 5 the result is below 100.

- Create a program that:
    - Given an input number, calculates and outputs whether a number is magic or not
    - Repeats itself, allowing the user to enter 1 number at a time and see the output. It should keep running until the number 100 is entered
    - Validate the program so that it only accepts integer values between 1 and 100

Task 4

An ‘Armstrong number’ has 3 digits and is an integer where the sum of the cubes of each of its digits is equal to the number itself e.g.:

371 is an Armstrong number - $3^3 + 7^3 + 1^3 = 371$ (27 + 343 + 1)

- Write a program that gives users two options:
  1. **Check number** – the user can enter a number and it will output whether it is an Armstrong number or not
  2. **Show range** – the user enters a start and an end value and the program outputs all Armstrong numbers within that range, inclusive of the values entered

Count-Controlled Loops - FOR

Start counter = this starts counting from 1

Stop condition = this loop will continue **UNTIL** X is 10 or more

Increment counter - this will add one to x each time the FOR line runs

```
for( x=1; x<10; x++)
{
        printf("x is now %d \n", x);
}
```

The program would loop through and print:
X is now 1
X is now 2 etc….

Task 1

Write a program to display the times tables to the screen.

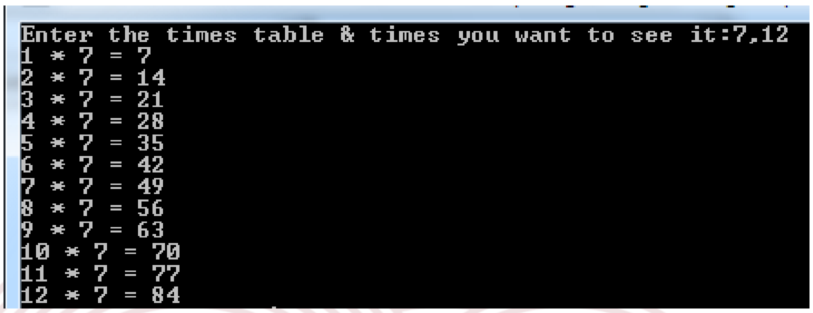
Requirements:

- The user should input:
  - which times table they would like to see
  - how many times that number they want to see
- The program should then display that times table from 1 times up to their chosen number of times
- Each multiplication should be on a separate line e.g.

  1 x 9 = 9
  2 x 9 = 18
  .
  .
  24 x 9 = 216

Output example on next slide….

Example output – Task 1



```
Enter the times table & times you want to see it:7,12
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
4 * 7 = 28
5 * 7 = 35
6 * 7 = 42
7 * 7 = 49
8 * 7 = 56
9 * 7 = 63
10 * 7 = 70
11 * 7 = 77
12 * 7 = 84
```

Task 2

Produce a program that makes the XY pattern output below:

- You can only print one character at a time
- You can only use a **maximum of 2** printf statements

```
XXXXXXXXXX
XXXXXXXXXY
XXXXXXXXYY
XXXXXXXYYY
XXXXXXYYYY
XXXXXYYYYY
XXXXYYYYYY
XXXYYYYYYY
XXYYYYYYYY
XYYYYYYYYY
YYYYYYYYYY
```

Task 3

Create a game that follow these requirements:

- A random number between 0 and 30 is generated and stored
- The program allows a user to guess a number
- If the guess == the random number then the user wins and gets a pay-out based on the following rules:
  - a multiple of 10 they get 3x their credits back
  - a prime number they get 5x their credits back
  - A number below 5 they get a 2x credit bonus
- The user should begin the game with 10 credits and it should end when the choose to exit OR if they have 0 credits
- The user can choose the amount of credit they want to bet (Within the range they have – they should not be able to bet negative values)
- Combinations of the win scenarios should be catered for.. e.g. if the guessed value was 3 this wins 7x their credits (It's prime and <5)