Programming 1: Tutorial 5

Introduction

For this tutorial you are going to be advancing the concept of controlling the flow of the program, this time using loops to repeats blocks of code.
You can either over write previous question solutions with new ones or you can have a project for each of the exercises – don't forget to make them blank applications.

Hopefully you remember the basic layout of a C++ program?
If not here is a refresher, basic C++ programs like the ones you have been asked to do here have the following layout:

```
#include <iostream>
// include statements up here

int main()
{
  // block of code between the { and } brackets

  return 0;
}
```

It may also help if you go through a design stage for each exercise where you plan and puzzle through the logic before actually coding.

To complete these exercises you may need to reference the notes, either online or use the printed ones provided. Doing this isn't a bad thing; most programmers have to look up the syntax to make sure there code is correct.

There is sample code at the end of the tutorials, typing and running this code may help you when it comes to solving the exercises.

It is also important to understand what is happening with the questions, rather than just typing it into the compiler. If you look at it, run it and don't understand then just ask.

Errors

Again after the actual tutorial is a look at another type of error and what results it can cause.

`While/do-while` Loop Questions

1) When does the code block following `while(x<100)` execute?
A. When x is less than one hundred
B. When x is greater than one hundred
C. When x is equal to one hundred
D. While it wishes

2) How many times is a do while loop guaranteed to loop?
A. 0
B. Infinitely
C. 1
D. Variable

`for` Loop Questions

1) What value would sum have when this loop is exited?

```
int sum = 0;
for (int i = 20; i > 10; i--)
{
      sum = sum + i;
}
```

2) What does the following for loop actually do? When does it exit? Why?

```
for (int i = -5; i > 0; i++)
{
      cout << i;
}
```

3) What does the following for loop actually do? When does it exit? Why?

```
int sum = 0;
for (int i = -5; i; i++)
{
      sum = sum + i;
}
```

4) What is output by the following for loop? When does it exit?

```
int sum = 1;
int exit_loop = 1;

for (int i = 0; exit_loop; sum = sum + i)
{
      if (sum > 100)
      {
            exit_loop  = 0;
      }
      cout << sum << endl;
      i++;
}
```

Beginning `while` and `do-while` Loop Exercises

Summing Loop
Write a program using a while loop to find the sum of the integers 0 through 100 inclusive. Display the resulting sum.

User Event-controlled
Write a while loop that will only enable the user to exit if the press ‘q’

Beginning `for` Loop Exercises

ASCII Characters
Write a program that prints out the all of the ASCII character types.

Hints:
You will need to use a for loop for this
The ASCII character types are from 0 to 255
You will need to use explicit type casting

Seeing Stars
Write a program that will request one number between 1 and 9 from the user and will print the following pattern of stars (user input determines the number of rows and the number of stars in the first row).

```
* * * * *
* * * *
* * *
* *
*
```

Advanced `while` or `do-while` Loop Exercises

Circle Calculator

Implement a program that asks the user for a valid circle diameter as an input: Valid is in the range 1.75 - 13.25. If they enter a negative number quit, else repeat the loop asking for a valid input.
If they enter a correct number display the following:

```
Your Circle:
Diameter = <display number>
Radius = <display number>
Area = <display number>
Circumference = <display number>
```

Use a do-while loop to trap them until they exit or input a correct number

Circle equations are:
radius = diameter / 2;
area = (pi*radius)*radius;
circumference = pi * diameter;

Runtime (Execution) Errors

To start of with this week we are going to look at the third type of error – Runtime Errors

Runtime (Execution) Error

- Error occurs while the program is running, causing the program to 'crash' (Terminate abnormally). This usually produces an error message from the operating system
- Error is frequently an illegal operation of some sort, examples are:
  - Access violations – trying to use a resource that hasn't been allocated
  - Arithmetic errors – Divide by 0
- Program source code compiles and links without errors – detected when the program is run
- Some operating systems do not reliably detect and respond to some execution errors

The Most Common Runtime (Execution) Error

Is likely to result from using un-initialised variables

Try the following code out:

```
#include <iostream>

using namespace std;

int main ()
{
      int number; // declared not initialised

      cout << number; // print out the un-initialised

      return 0;
}
```

This is an example of a common run-time error and report.