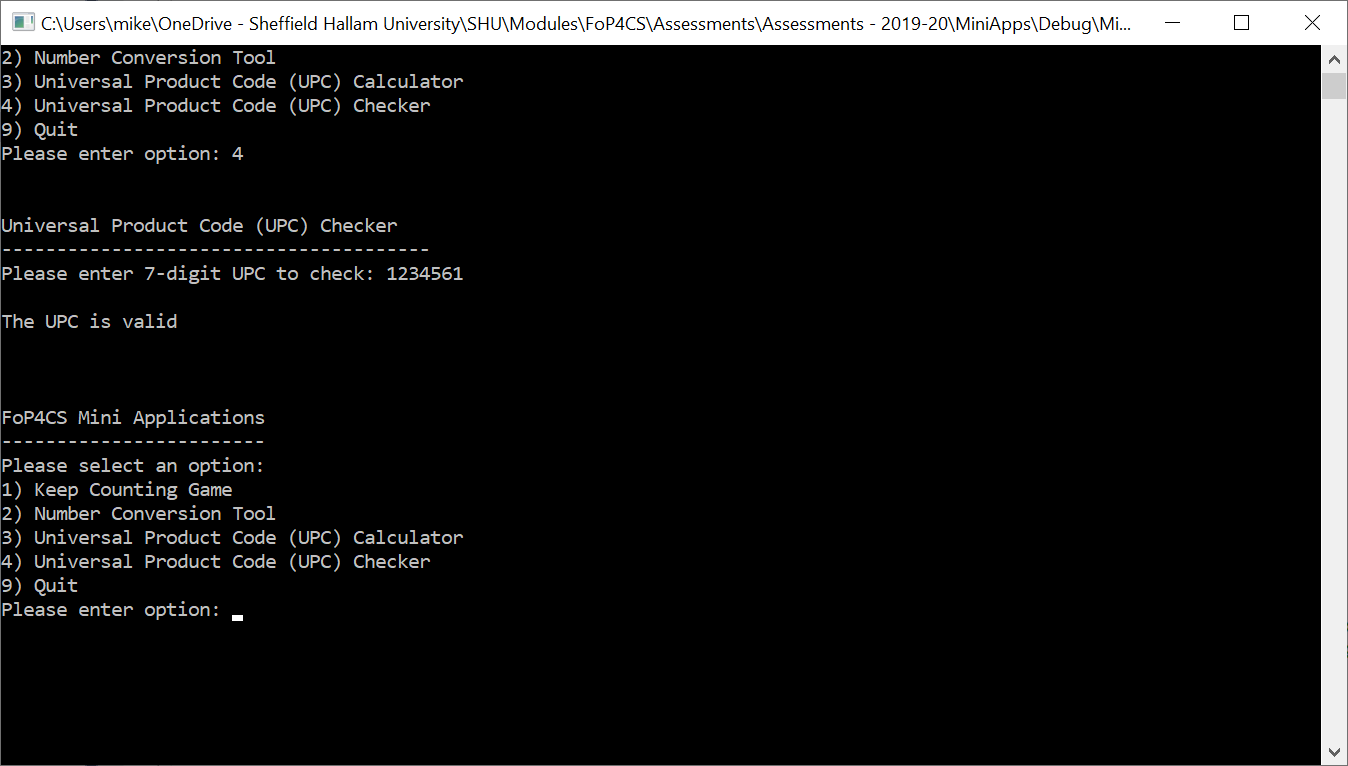
In this task you will develop a series of mini applications that are designed to get you thinking logically about constructing code in C/C++. The concepts required to implement these applications form only a small and introductory selection of features found in the language, although you will need to carefully consider how to put those building blocks together to create the mini applications. This is an individual piece of work and in-module retrieval is available for this assessment.

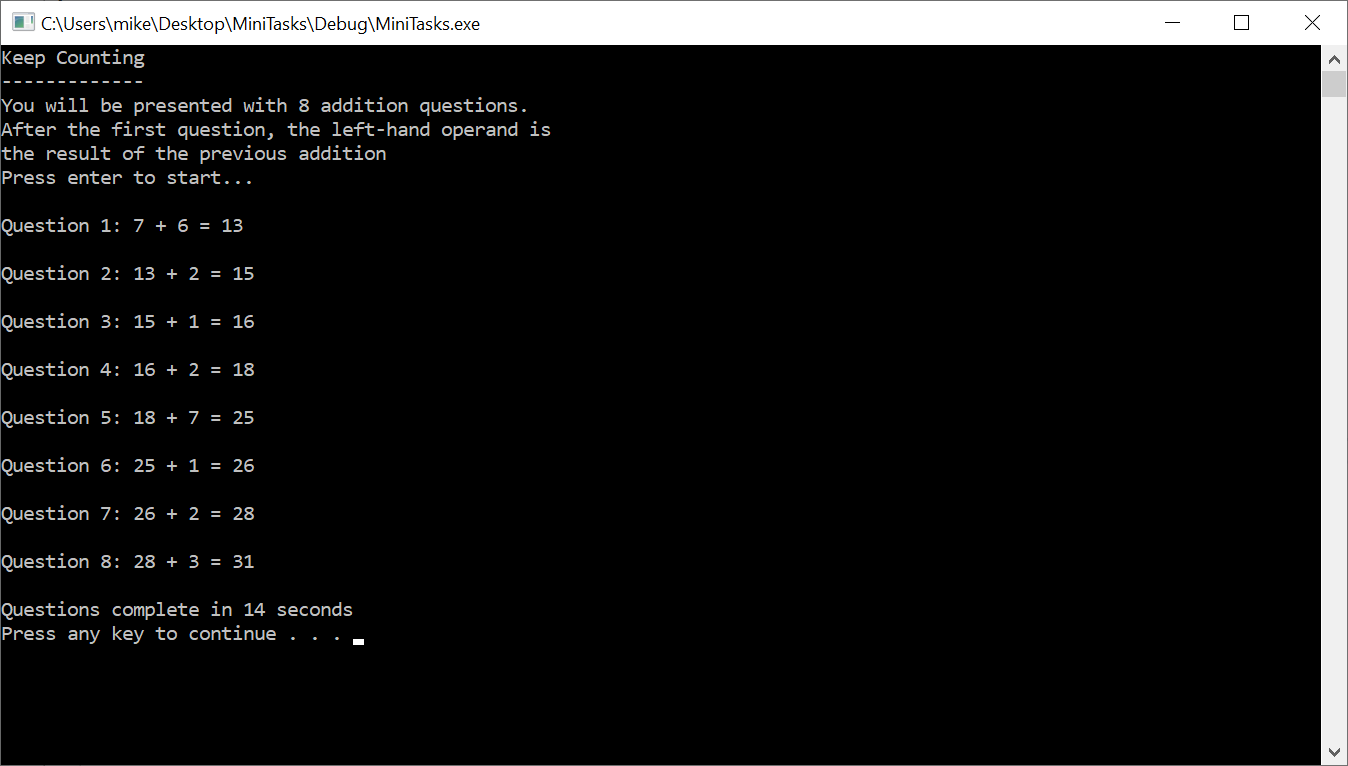
**Introduction**

All the applications given in this brief can be created using the techniques covered in class, however some can be better encoded with more advanced features such, as arrays (which we will look at in due course). If you choose to research and deploy those advanced features of the language, then you will receive the very top marks (please refer to the marking grid and ask if you wish clarification).

There are three mini applications to create and all joined together by a fourth menu system. An example of the core features is available on the module Blackboard site to download and try. The following sections outlines the core details and requirements for each of the applications.

**0) Menu System**

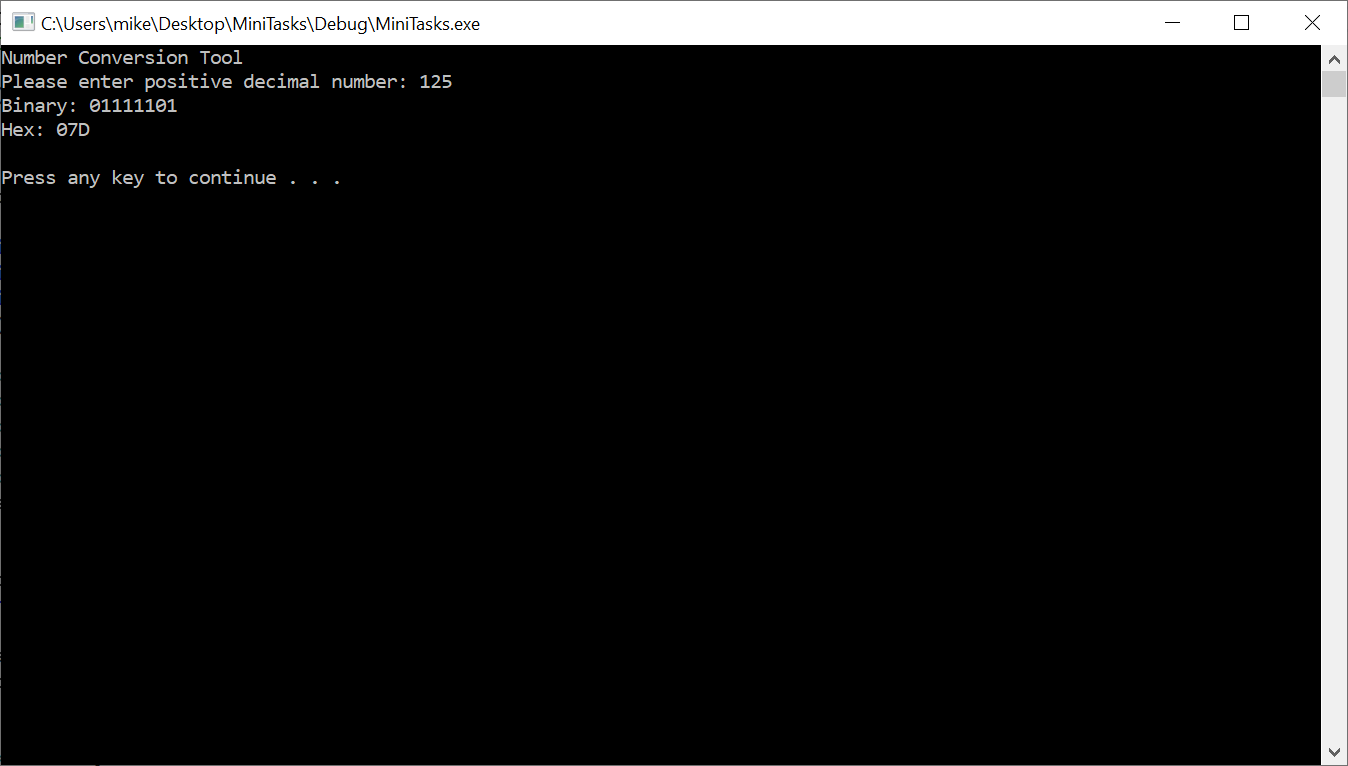
* ****Create a top-level menu system that allows the user to select which of the mini applications to run
* If the user presses a number not in the list of options, display a message and show the menu again
* Once the selected application has finished, the menu should be displayed again and the user asked which app to run (or exit)

****

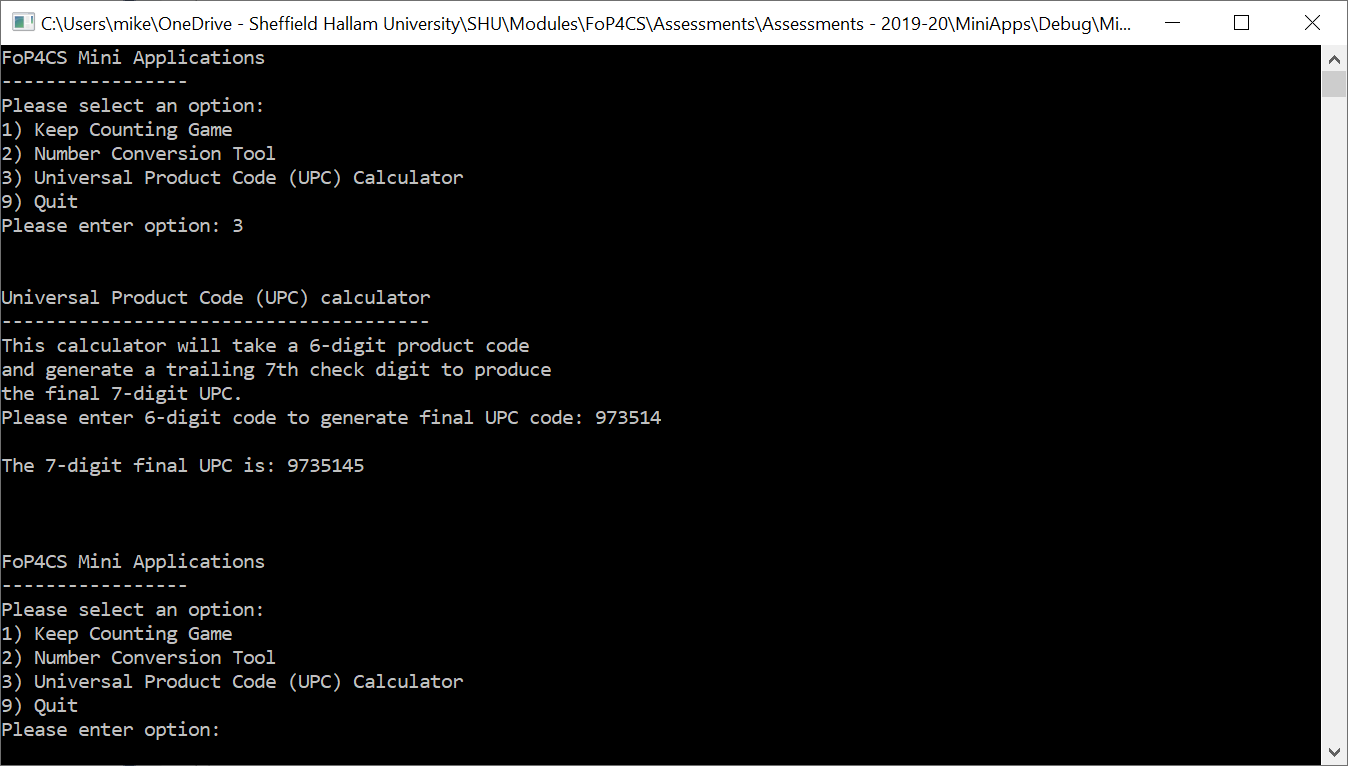
**1) Keep Counting**

* Add the same number to the previous result 7 times (starting from a pair of random numbers)
* You can assume the user enters valid input for the data type
* The game is timed, although the player needs to correctly answer all questions successfully for their time to be displayed at the end
* As soon as the player gets a wrong answer, the correct answer is displayed and the game finishes
* A sensible range of values for the operands is between 1 and 10 inclusive, but you can decide to choose your own range
* Can be extended by allowing subtraction

**2) Number Conversion Tool**

* ****Input a positive decimal number and convert to binary and hexadecimal
* Perform the conversion in code using a suitable algorithm (don’t just use a built-in function)
* You can assume the user enters a valid input
* Can be extended by inputting other bases (binary and/or hexadecimal) for conversion

**3) Universal Product Code (UPC) Generator and Checker**

* ****Input a single 6-digit number from the user, which will create a 7-digit number when the check digit is appended to the end. For example, the user input “123456” number will become “123456?” with the ‘?’ indicating the check digit.
* Check the user hasn’t entered more than 6 digits (fewer digits will simply result in zero padding to the left). For example:
  + “1234567” is not a valid user input from the user and needs to caught and the user notified
  + “123” will be interpreted as “000123”
* Use the following rules to determine the check digit:
  + Add the digits in the odd-numbered positions together (working left to right where the 1st position is the left-most digit and excluding the check digit) and multiply by three
  + Add the digits in the even-numbered positions to the result.
  + Take the remainder of the result when divided by 10 (modulo).
    - If the remainder is equal to 0 then use 0 as the check digit
    - If the remainder is not equal to 0 then subtract the remainder from 10 to derive the check digit
* Display the 7-digit number that includes the check digit. For example:
  + Input “123456” -> (3 \* (1 + 3 + 5)) + 2 + 4 + 6) = 39
  + Remainder of 39 when divided by 10 is 9
  + 10 – 9 = 1
  + Check digit is 1 and thus the UPC is 1234561
* Provide a UPC checker that allows the user to enter a 7-digit code and validate the check digit for correctness (not shown in screen shots but provided in the example application on Blackboard)
* UPC is a real thing (although we are reducing the number of digits in our example) – see https://en.wikipedia.org/wiki/Universal\_Product\_Code

**General principles**

* “Can be extended” suggestions indicate areas you can go beyond the code specification and will demonstrate creativity beyond the core for the higher grades
* Assume the user enters input valid for the data type expected (i.e. no need to capture for the user pressing a letter when a number is expected) but you should check for range correctness (e.g. the menu system should give an error when “123” is entered)
* Each of the mini applications can be written in a single function (although feel free to further subdivide into smaller functions where suitable – this will further demonstrate a deeper knowledge and understanding)
* Even if you don’t get a fully working application, you will still get credit for what you have achieved
* While it is expected that everything will go into a single .cpp file, make sure you comment your code to an appropriate level, use sensible naming and make the code readable using a consistent and sensible layout.
* Briefly describe the test data you used as you developed the code within a comment section immediately above each method you write.

**Code Development and Hints**

* There is no single correct solution and you will have to make many decisions about the implementation but do so logically and thoughtfully.
* Use methods/functions to help segment your code into manageable and meaningful modules that can aid reuse and readability
* There is no single right solution, but some code is better than others… think about what you are doing, what you want to achieve and implement it in a sensible, robust and efficient way
* Evolve your code and regularly test what you have written; when I put together applications, I didn’t sit down and write it in one go and expect it to run… I build the application in small stages, progressively adding in functionality and testing and debugging as I go (yes, everyone needs to test and debug code they write no matter how long they have been coding for so get practiced at doing so)

**Grading Guidelines**

Marks will be awarded as illustrated below:

|  |  |
| --- | --- |
| **Application** | **Percentage Available** |
| Menu System | 10% |
| Keep Counting | 25% |
| Number Conversion | 30% |
| Universal Product Code Calculator | 35% |

The University common grading descriptor will be used to determine marks for each application (provided at the end of this document). The marking scheme embeds the concept of extended work by rewarding only the highest marks to those who demonstrate evidence of independent investigation, learning, critical thought and problem analysis (via good code solutions).

**Submission Process**

Your assignment should be submitted electronically through the module Blackboard site as either:

* A single, uncompressed CPP file if all your code is in a single source file – use this option if you only have a single CPP source file as it will ensure you can preview the format and submission within the Blackboard preview window.
* Or a single ZIP file that contains your entire project and source code when using multiple CPP files (minus build and intellisence files and folders). If you upload a project ZIP file, the source code must be in the form of a Visual Studio 2017 project within the compressed ZIP file and contain all files that allow the project to be opened, built and run on a campus computer.

Your last on-time attempt will be viewed and graded (as per university regulations).

Make sure that you upload the correct files by checking once you have submitted - mistakes discovered after the deadline cannot be corrected; it is your responsibility to ensure that you submit the correct files by the deadline. You may be asked to provide a walkthrough of your code during which you will need to discuss all aspects of the work you submitted with your grade being subject to a successful walkthrough and discussions of your work.

**Remember to include all source code and check your submission once uploaded.**

The submission deadline is given at the top of this document.

**Learning Outcomes**

This task assesses the following learning outcomes from the module descriptor

* Describe, recognise and deploy key concepts that relate to designing small imperative applications
* Select and apply appropriate software tools and program testing techniques on small program