# T01 - Get Started In SHU Labs

**Tasks set in tutorials should be finished in your own time!**

1. Individual presentation:   
   Let's talk about your Programming experience (if any): languages used, etc. Expectations for the course and this module.
2. ***(In your own time - if you haven’t done this already in the IT induction session)***log on to the machine to have a quick look at the Student Portal system (<http://shuspace.ac.uk/>).  
   Login to My.SHU account by clicking on the **Login** button and use your login name and password to access the system.
3. Create a bookmark for safe-keeping of this 'Student Portal' web address
4. From the SHU Space login to your SHU email account using the same login name and password.  
   If you haven't already done so, create a signature for yourself showing your name, student number, course title, year (Level 4) and tutorial group. There may be more than one student with the same name!
5. From the SHU Space click on the **Login** button to login to a virtual online learning environment known to us as **Blackboard** (or Bb for short):
6. Look in the 'Module Sites' box our module.
7. Check the announcement (There should be at least one: Welcome to the module).
8. Look at the overall structure of the site and access the links and files posted there for you. *NOTE: You can also download the files, save them in your evidence folder (Q****:*** *drive – which his accessible from home), external drive or pen drive and open them*.
9. Open the **Lectures and Tutorials** section, look at the first week.
10. Look at the power point file and word documents and videos

If you are not sure ask your tutor!

1. You can start with some VERY simple programs. You will have to try and guess your way around if you have not done any programming before, but don’t worry: we are not assuming that you understand it all already. We will look at all of this in detail very soon – starting from scratch.
2. Create a new empty project in Visual Studio and open a new .cpp file. Type in (or copy and paste) the code below:

#include <iostream>

#include <string>

using namespace std;

int main()

{

cout << "Please enter your name: ";

string name;

cin >> name;

string answer;

cout << "\n" << name << ", are you ready to do some times tables? \n";

cin >> answer;

if (answer == "yes")

{ //A

cout << "There we go....\n";

cout << "TIMETABLE CALCULATIONS TO BE IMPLEMENTED SOON...\n";

} //B

else

{

cout << "Maybe some other time...\n";

}

cout << "\n\n";

system("pause");

return 0;

}

1. Run this program, trying different answers.
2. Test it with name *Tom* and answer *yes*. Then with answer *Y* or *N*. Does it behave as you would expect?
3. What happens when you type in two words like *John Smith* for the name? Why?
4. What happens if you remove the brackets on lines **A** and **B**?
5. Modify the program so that the condition for the if statement is a bit more complicated:.

if ((answer == "yes") || (answer == "y"))

1. What has changed? What do you think the ‘||’ symbol mean?
2. Replace the section between lines A and B with the following code:

{ //A

cout << "There we go....\n";

cout << "TIMETABLE CALCULATIONS\n";

int n1(5);

int n2(6);

cout << "\nWhat is " << n1 << " \* " << n2 << "? ";

int product;

cin >> product;

while (product != (n1 \* n2))

{

cout << "\nError: Try again! \n";

cout << "\nWhat is " << n1 << " \* " << n2 << "? ";

cin >> product;

}

cout << "Correct! \n";

} //B

1. Modify the program as follows:
2. Add the following line at the top of your program to include the **time** library used to create random values:

#include <ctime>

1. Add the following line of code just after line A.

srand(time(0));

1. Change the declaration of the variable n1 and n2 so that they contain different random values (between 0 and 9 each time the program runs.

int n1(rand() % 10);

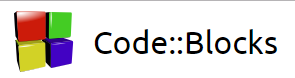
int n2(rand() % 10);

and test the new behaviour.

1. Finally, modify the program so that the user is presented with 5 multiplication exercises in one single run of the program.
2. ***(In your own time)*** Open the **VS2022C++\_ConsoleApplications** document from the “**Support Resources\Visual Studio C++ notes”** section of the module Bb site.Follow the instructions to get familiar with the Visual Studio C++ environment that we are going to use this year (and the following ones...). You will develop a couple of very small programs on the way. Don’t worry if you don’t understand all the code there. It will come.

* [](https://visualstudio.microsoft.com/vs/features/cplusplus/?rr=https://www.google.co.uk/)[MS Visual Studio 2022 C++](https://visualstudio.microsoft.com/vs/features/cplusplus/?rr=https://www.google.co.uk/)

There are other IDEs you can use to develop your C++ programs that you need to be aware of as they may be more widely accessible than MS Visual Studio and it just helps to try different tools on the same job. Try it at home.

* [repl.it](https://repl.it/)
* [Code::Blocks](https://launchpad.net/codeblocks)

Have a look in **“Support Resources\Some C++ booklets”** on the module Bb site. Do they look interesting or do you prefer books from the **“Module Home Page\reading list online”** on the module Bb site?

1. ***(Optional - for those of you with programming experience)*** Create a small C++ program that counts the number of words in a text.
2. Create a new, empty C++ project (e.g., **CountWordsProject**). Add a new source file (e.g., **Main.cpp**), copy and paste the code below in that file and save it, compile and run it. Try it and test it.

/\* This program counts the number of words in a piece of text.

We'll assume that   
(a) it can only read in and process characters in text one at a time.

(b) words are separated by a single space

(c) the last word in the text ends with a period '.'.

\*/

#include <iostream> //include input/output library

using namespace std; //using standard namespace

int numberOfWords; //declare variables

char letter;

/\* count number of words in a piece of text

\*\*/

int main()

{

cout << "\nThis program counts the number of words in a text ";

cout << "\n(ending with '.', single spaces between words) ";

cout << "\n-------------------------------------------------\n";

numberOfWords = 0; //initialise variables

cout << "\nEnter a line of text: ";

cin.get(letter); //read any character (even space)...

while (letter != '.') // ... and count words

{

if (letter == ' ')

numberOfWords = numberOfWords + 1;

cin.get(letter);

}

if (numberOfWords > 0)

numberOfWords = numberOfWords + 1; //take last word into account

cout << "\nNumber of words: " << numberOfWords << ".\n"; //result

system("pause" ); //to hold the output screen

return(0);

}

1. This solution is not correct. Can you find the bug? If yes, can you fix it? Clue – does it always count the right number of words?
2. After saving and closing your application have a look at your project folder in file explorer and see which files have been created and where. Look for the solution file (**.sln**), the project file(s) (**.vcxproj**) and the source file(s) (**.cpp**) and **Debug** folder and find the executable file (**.exe**).

* If you want to send/email a solution to someone, delete the temporary files that are not required as the solution folder gets really big. All you need is: cpp/h files, vcxproj files, sln file, NOTHING else.

Don't worry if you do not understand it all at this stage.   
The main outcome of this project is   
to learn how to design such programs and   
implement them in C++!   
So we have a few months to get there.

# T01 - A quick tour…

The **main stages of program development** that are relevant for this module are:

* **Think**: Understand the problem at hand and find a way to solve it (typically using a small example)
* **Design**: Produce an algorithm (i.e., a set of instructions) that will solve the problem and express this in one of the two design notations used (pseudocode or flowchart)
* **Code**: Translate this algorithm into C++
* **Test**: Run and test the C++ program with a set of well-chosen data to check that it works as expected.

1. A programmer has been asked to design a program that reads in a price representing the full price of an item, it then calculates and displays the price with a 10% discount.

**produceDiscountedPrice**

**output("Enter the original price: ")**

**input(originalPrice)**

**set discount to originalPrice \* 0.1**

**set discountedPrice to originalPrice - discount**

**output("The discounted price is ", discountedPrice)**

* You can assume (i.e., your program does not need to check this) that the user enters a positive value for the price.

1. What data items does this program use? Are they variables or constants, named or literal? Which type? Which scope?
2. Add the necessary data item declarations, indicating their data type as a comment.
3. Identify the various expressions used by this program. How/when are they evaluated? What is their type? Their value (when known)?
4. Identify the various types of statements used by this program: assignment, input/output and control statements.
5. Could any of the instructions be re-ordered without changing the behaviour of the program?
6. Modify this program so that it also reads in the rate of discount expressed in %, calculates the discount on the item at that rate and displays the discounted price.

* You can assume (i.e., your program does not need to check this) that the user enters a positive number for the price and a number between 0 and 100 for the discount rate.

1. What output will your new program produce for the following input values?

* originalPrice 100 discountRate 10
* originalPrice 200 discountRate 50
* originalPrice 500 discountRate 0
* originalPrice 200 discountRate 100
* originalPrice 0 discountRate 10

1. What would happen if the values entered by the user were negative? If the discount was more than 100? Do we need to test this?
2. Here is an alternative solution to the same problem. Does this one work? If yes, is it better than the previous one and why?

**produceDiscountedPrice**

**local originalPrice, discountRate, discountedPrice //floating point numbers**

**output("Enter the original price: ")**

**input(originalPrice)**

**output("Enter the discount rate: ")**

**input(discountRate)**

**set discountedPrice to originalPrice – ((originalPrice \* discountRate) / 100)**

**output("The discounted price is ", discountedPrice)**

What about this one?

**produceDiscountedPrice**

**local originalPrice, discountRate, discountedPrice //floating point numbers**

**output("Enter the original price: ")**

**input(originalPrice)**

**output("Enter the discount rate: ")**

**input(discountRate)**

**set discountedPrice to originalPrice \* (1 – (discountRate / 100))**

**output("The discounted price is ", discountedPrice)**

1. Implement your favourite solution (i.e., the one you understand best) in C++.   
   Create an empty project (called **Discount**), add a new source file (called **calculateDiscountedPrice.cpp**), attach it to your solution and type in your C++ code into it.   
   The interaction with the program would produce the following behaviour (with user's input shown in bold here):

|  |
| --- |
| Enter the original price: **100.0**  Enter the discount rate (in %): **20.0**  The discounted price is 80 pounds. |

1. Format the program's outputs so that, typically, the pound sign ‘£’ shows before a price and prices are always displayed with 2 digits after the floating point (e.g., 180 pounds should be displayed as £180.00 and 9.64534 pounds as £9.65).

|  |
| --- |
| Enter the original price: £**200**  Enter the discount rate (in %): **10**  The discounted price is £180.00. |

|  |
| --- |
| Enter the original price: £**9.99**  Enter the discount rate (in %): **3.45**  The discounted price is £9.65. |

To format the prices with a 2-digit precision you need use C++ output manipulators (in the **<iomanip>** library). *(See slides and notes for examples)*

* By fixing the precision of the output, e.g.,

cout.setf(ios::fixed);  
 cout << setprecision(2);

To display the ‘£’ sign you need to obtain its character symbol by using one of the three following methods:

* By converting its ASCII number (156) as a character directly in the output statement, e.g.,

cout << "The discounted price is " << static\_cast<char>(156)   
 << discountedPrice;

* By placing the ’£’ symbol into a const and using it whenever needed in the program, e.g.,

const char poundSign( 156); //set poundSign to '£'   
 cout << "The discounted price is " << poundSign << discountedPrice;

* By using its octal value (\234) or hexadecimal value (\x9C) directly in a constant text, e.g.,

cout << "The discounted price is \234" << discountedPrice;

1. The **div** and **mod** operators handle the integer division, i.e., **div** gives the quotient and **mod** the remainder of the division. For example, **10 div 3** is **3** and **10 mod 3** is **1**.  
   Look at the following flowchart representing a program which makes use of these operations.

* Assume the number entered is a positive integer, exactly 3-digit long.

output("Enter a 3-digit number: ")

input(number)

set hundreds to (number div 100)

end

start

set tens to ((number mod 100) div 10)

set units to (number mod 10)

output("The reverse number is ", units, tens, hundreds)

1. Describe what this program does. Does it always work as intended?
2. Rewrite it in pseudocode, including necessary data item declarations.
3. Code this solution in C++. In C++, the **div** operator is '/' and the **mod** operator is '%'.
4. Test it.   
   What would happen if the number entered was less than 100 or more than 999? Or a float number? *Look carefully at the specification given*. Do we need to test this?
5. Write a program to convert a distance in cm into metres and cm.

* Assume that the number of CMs given is valid (i.e., is a positive or zero integer value representing a distance).

The interaction with the program would produce the following behaviour:

|  |
| --- |
| Enter the distance in cm: **3723**  The distance is 37m and 23cm. |

1. Write a program to add VAT (20%) to a given price.  
   **Specification:** The program asks the user for the price without VAT and reads it in. You can assume that the user will enter the appropriate type of data (i.e., a positive floating point number or zero). It produces the price with VAT.  
   The interaction with the program would produce the following behaviour:

|  |
| --- |
| Enter the price without VAT: £**100.0**  The price with VAT is £117.50. |

* Create a design doc with a copy of the above spec. Call the word document, “Tut01 design”.
* Add a flow chart to the word doc.
* Add a pseudocode algorithm to the word doc.
* Add a test plan to the word doc.
* Keep your full C++ solution folder, but check the program runs outside VS.

Competencies: 1.1,1.2.1,1.2.3,1.4,1.6.1,1.6.3,1.7,1.9,2.7.1