# COMP07027 Introduction to Programming

Practice Programming Project – 2014/15

**Banner ID** **Name Yu-Ching Ho**

**Your program should address the following points:**

| **Point** | **Description (the facilities you need to implement)** | **Achieved** |
|---|---|---|
| 1 | The program follows the above specification in providing a description of the problem, the initial state (the three disks are all initially on pole A) and in prompting the user to choose which disk to move and to which pole. | Yes. |
| 2 | The program defines a suitable set of variables to keep track of the location of the three disks. | The 3 int variables would have been fine for this, but only one of them is ever used and you seem to have got confused about which of the **`int`** variables are **`final`** (i.e. constants) and which can change in value. |
| 3 | The program gets the user input and, if it is valid, updates the program state so that the disk is moved to its new pole. This will need you to identify some more variables to keep track of inputs and conditions. If the input is not valid, the program displays an appropriate message and the program state is not updated. | Things go badly wrong here. You declare a **`boolean`** variable, `valid`, that is intended to indicate, in its first usage, whether the disk chosen can move, and in its second usage, whether the disk can move to the chosen pole. However, in validating the disk choice `valid` is always set to **`false`** unless the user chooses the small disk. Here is your switch statement: `switch (option) { case small_disk: valid = true; TextIO.put( "You have selected the Small disk."); break; case medium_disk: valid = medium_disk <= small_disk; TextIO.putln( "You have selected the Medium disk."); break; case big_disk:  valid = big_disk <= medium_disk; TextIO.putln( "You have selected the Big disk.");` |

```
break;
default: valid = false; }
```

small_disk is a constant equal to 1, medium_disk is a constant equal to 2, and large_disk is a constant equal to 3. So your code says that choosing the medium disk is valid if 2 <= 1 and choosing the large disk is valid if 3 <= 2. So, choosing these disks is never valid according to your program’s logic.
 Things do not get better thereafter. If the disk choice is valid (i.e. the user has chosen the small disk) the program logic continues:

```
if (Tower_small_disk == LEFT) {
 Tower_small_disk = MIDDLE; } else {

      Tower_small_disk = LEFT; }
```

This moves the small disk to the middle tower as its first move – but at this point the user has not yet indicated what tower they want to move the disk too! The rest of the code does not use the disks position again but, as in your first switch statement, continues to compare the values of constants – so the comparison outcomes are always the same as nothing the program does can change the values of these constants. Note also that although the code goes on and asks the user to choose a pole to move to, the **`switch`** statement does not use the constants that represent the poles in its cases but the constants that represent the disks!

| | | |
|---|---|---|
| 4 | After each move, the program displays where the three disks are now located. | No – the program exits after executing the do-while loop just the once. Also, there is a mistake in the code that displays where the disks are – only the position of the small disk is used:<br><br>`TextIO.putln(`<br>`"The Medium Disk is on Tower " +`<br>`     Tower_small_disk + ".");`<br>`TextIO.putln(`<br>`"The Big Disk is on Tower " +`<br>`     Tower_small_disk + ".");`<br><br>Note that these should use Tower_medium_disk and Tower_large_disk, not the small disk variable. |
| 5 | After each move, the program checks if the puzzle has been solved | No. The condition in the do-while loop compares constants and the tests are meaningless. |
| 6 | If the puzzle is not solved, the program repeats step 3. | No, the program never repeats step 3. |
| 7 | The program counts the number of moves and displays a message at the end of the program if it has taken more than the seven moves required for an optimal solution. | No. |

| | | |
|---|---|---|
| 8 | Program structure. The solution to last year’s practice project had all the code in one subroutine. Consider using subroutines to do such things as validate the input, to check the program state, and to update the program state. Document your program with comments in the code. At the very least, there should be comments against each variable declaration to explain what the variable is for and, if you have subroutines, above each one explaining what the subroutine does. Include a comment at the beginning of the program explaining what the program does. | All the code was in the main() method. There was a comment above the declarations of the constants, and another above the variables – there is only one other comment in the code (apart from the automatically generated TODO comment that you should have deleted). As already noted, the program logic does not work at all and it appears that you have not understood the problem that has been set.<br>Note that another student submitted an identical program to you – did you work on this with someone else? |