**Chapter 5 Multiple selection using SWITCH … CASE**

In this chapter we look at another aspect of Selection.

In a previous program we had to write code which performed different actions depending on the operator's choice of one out of four operators.

The portion of code which did this looked like the following:-

**if (oper == '+')**

**answer = first + second;**

**else**

**if (oper == '\*')**

**answer = first \* second;**

**else**

**if (oper == '-')**

**answer = first - second;**

**else**

**answer = (float)first/second;**

This works but it is cumbersome and hard to read. What if there were ten choices!

Fortunately there is better provision in C++ for such multiple choice situations. Here is an alternative to the above code, using a **switch** statement:-

**switch (oper)**

**{**

**case '+':**

**answer = first + second;**

**break;**

**case '\*':**

**answer = first \* second;**

**break;**

**case '-':**

**answer = first - second;**

**break;**

**case '/':**

**answer = (float)first / second;**

**break;**

**}**

The variable on which the choice is based is in brackets following the keyword '**switch**'. Each possible value (or **case**) of that variable is then listed separately - followed by the action(s) which are to be performed when that case arises.

Each case must be terminated by the **break** keyword - which transfers control to the end of the **switch** statement. If you do not include a **break** then execution will continue into the next **case** section and on until a **break** is encountered or the end of the **switch** statement is reached.

See if you can incorporate this change into your program. (**Chap0501.cpp**)

These two ways (nested **if**s and **switch**) of dealing with selecting on the basis of user input have until now dodged an important issue. What happens if the user, instead of hitting +, -, \*, or /, hits another character, either by accident or as a result of incompetence? In both cases no calculation is done for '**answer**'*,* which is therefore likely to have some random value stored in it. The user will still be asked for a guess, though any reply is then almost certainly judged as incorrect.

We can at least let the user know that this has happened. In a **switch** statement, we can have a '**default**' section to handle any case not covered in the '**cases**' section, like this:-

**switch (oper)**

**{**

**case '+':**

**answer = first + second;**

**break;**

**case '\*':**

**answer = first \* second;**

**break;**

**case '-':**

**answer = first - second;**

**break;**

**case '/':**

**answer = (float)first / second;**

**break;**

**default:**

**cout << "Invalid - must be + - \* / \n\n";**

**}**

Can you work out how to provide this default case in the nested **if** situation? It's a worthwhile bit of brain-twisting, though in most multiple choice situations we would use a **switch** statement.

Include the '**default**' addition in your program and try it out.

Now at least the user knows an error has been committed. Unfortunately the program will still continue in a mindless sort of way.

Can you work out how to force the user into typing in a valid character, based on things you already know about input validation? Try it.

**When is SWITCH a valid solution to a problem?**

SWITCH will only work when the 'choice' variable is of an '**ordinal**' type. Ordinal types are those types which have a limited number of discrete and predictable values. **Char**acters and **int**egers are ordinal types, but **float**s are not, as they can take on an infinite number of values. So you cannot use a **float** variable in a **switch** statement.

Also **switch** statements can only be used to differentiate between actual values of a variable - they cannot easily be used for ranges. This IF statement cannot be translated into a SWITCH statement:-

**if (x > 10)**

**cout << "…..**

However this IF statement- can be written using SWITCH:-

**if (x == 10 || x == 20) switch (x)**

**{ {**

statement**; case 10:**

statement**; case 20:**

**}** statement**;**

statement**;**

**break;**

**}**

There are no statements specifically for the case of 10, and there is no **break** so execution continues to the case 20 statements. The same statements will therefore be executed for 10 or for 20.

Here are some problems you can tackle to practice using SWITCH.

You should incorporate some input **validation** as well.

Draw up a **design** before you start – sketch out the main sequence of actions.

As always you should **test** your programs thoroughly.

1 **Numbers to words**

Write a program which will continually ask the user to type in a number between 1 and 5, and which will then output the text form of the number typed in. For example, if the user responds with '2', the program should then display the word TWO on the screen.

The input should be validated - it should check that the user has indeed typed in a number between 1 and 5. Use a sentinel value such as 0 to exit the program.

2 **Snooker break calculator**.

Use a do loop to prompt the user for the code letter of each snooker ball potted:-

**R** (**r**ed) 1, **Y** (**y**ellow) 2, **G** (**g**reen) 3, **B** (**b**rown) 4, **L** (b**l**ue) 5, **P** (**p**ink) 6, **A** (bl**a**ck) 7.

Use a switch statement to add the correct score for each ball to the total break score.

When an ‘**M**’ (for Miss) is input the break is complete and you should output the final break total score.

eg Inputs **R A R B R L M**

Output **19**

3 **Roman numeral converter**

Use a loop to receive Roman numerals (M, D, C, L, X, V, I) from the user one at a time.

Use a switch to add the correct decimal equivalent to an accumulator.

When a ‘Q’ is entered to end the sequence you should show the final decimal equivalent.

3a This problem is straightforward if the Roman numerals are all in descending sequence:-

**MMXII** - should be 2012

3b It is a much tougher problem if the sequence is disturbed:-

**MMXIV** - should be 2014

… but if you aren’t careful you end up with a value of 2015.

The basic rule which applies here is that if the preceding character was of a lower value then it should have been subtracted rather than added. But you couldn’t do that at the time because you didn’t know what the next numeral would be.

This means that you have to compare each Roman numeral with the previous entry – and if it is higher value then correct the ‘error’ created by adding that previous entry. Hint - Achieve the correction by subtracting that previous value twice!

You might do some research on the exact rules for Roman numbers before tackling this harder version of the problem:-

(e.g. [*http://en.wikipedia.org/wiki/Roman\_numerals*](http://en.wikipedia.org/wiki/Roman_numerals))