Week 4 Exercises : Conditions and Branching

Feb 5th 2012

Until now we have focused upon data manipulation using only simple expressions. You have seen how to combine these expressions to solve problems in the context of simple sequential programs; combinations of input, output and assignment statements. This week we will look at more general decisions that cannot be made using simple techniques. The lecture has introduced the concept of control flow - different sequences of statements within the program that are selected by control-flow *constructs*.

Sometimes when information is being converted from one form to another every case is *regular* - the same operation (as an expression) will translate every input value to every output value. But sometimes the conversion that you need is irregular, and each case behaves differently. Understanding the difference between the two cases is an important programming skill that requires experience.

All of these exercises can be solved using if-statements in some way. They are collected into three groups that are similar in some way. When you work through each of the exercises within a single question think about which previous parts you can reuse - cut and paste code where possible to minimise the amount of work you must do and focus changing your previous solutions in just the places that differ.

To keep up with the pace of the course you must complete all of the non-starred exercises in the week that they are set. The starred exercises are optional for those of you that are aiming for a higher grade on the course.

1 Using if statements

We start the week with some numeric / bit-logic problems. This means that some part of the problem is familiar from last week and the new part involves using an if-statement.

1. a) Input a number into a `float` variable and display whether the value is an integer or not.
   b) Input an integer and display whether or not it is a multiple of two.

2. Input a pair of 2-vectors $[x_1, y_1]$ and $[x_2, y_2]$ and calculate the distance between them $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Output a message to say if the distance between them is larger than one.

3. Input an integer and display whether or not it is a power of two. Note: all powers of two have a Hamming Weight of exactly one.

4. * Input a 2-vector (x, y) and decide whether which quadrant of the plane it lies within. Output one of "TL", "TR", "BL", "BR" for top/bottom left/right. How does the structure of your program change if you change between a balanced nest of if-statements and a chain of if-else statements?

For the mid-week we are going to look at problems involving text stored in strings. You have already seen how to declare string variables, and how to input values into them / produce output from them. The new aspects of the problems involve using if-statements to decide how to operate on the data stored within them.

5. a) Input a pair of number and display one message from "First number is larger", "Second number is larger", or "Both numbers are the same".
   b) Input a single character and display if it is a digit, lower-case letter, upper-case letter or punctuation.
   c) Input a letter, if it is upper-case then convert it to lower-case, if it is lower-case then convert it to upper.

For the two starred exercises that follow, and the final exercise, you will require a method of taking individual characters from strings, and overwriting positions in strings with character values. Try the following program to see the two techniques that you need. The new syntax uses the square brackets `[` and `]` to access individual characters in a string. The dot character `.` accesses names within the string class as you saw in Exercise 7 from the first week. This time the name that is being accessed is a function that is built into the string class.

```
#include<iostream>
using namespace std;
int main(int argc, char **argv)
{
string x("Hello World!");
  cout << x[3] << endl;
char y;
  y = x[6];
  cout << y << endl;
  x[6] = 'X';
  cout << x << endl;
  cout << x.length() << endl;
}
```

Listing 1: String operations

6. * Input a string of five letters. If the string is a different length, or any of the characters are not letters then output an error message. If the string starts with a lower-case letter then convert any upper-case letters to lower-case. If the string starts with an upper-case letter then convert any lower-case letters to upper-case. For example, if the user inputs `HeLLo` then the program should output `HELLO`. If the user inputs `zORro` then the program should output `zorro`.

7. * Input a base in the range 0-35, input a 4-character string. Check that the string is a valid number in that base. If it is not then output an error message, otherwise output the decimal value of the number. Use the letters a for 10, b for 11 up to z for 35. For example, if the base is input as 17 and the string is `6g2h` then the number is invalid (as the `h` would be a digit 17 and the base only allows 0-16). If the base is 17 and the string is `6gg2` then the value would be 34376.

The final task is not split into separate pieces. There are several problems within this task that you will need to identify, and then work out how to solve them based on your previous work. We want to see some evidence that you have tried to break this into smaller problems before Thursday's lab session. We will want to see some notes, diagrams or other approach that you have tried to split this into simpler problems using what you have seen in lectures and learned in class. We will then discuss how to break this problem into simpler pieces during the Thursday laboratory.

8. Input a time as a 5-character string , e.g. "02:45" or "14:10". Output a textual description of the time. In the first example you will output "15 minutes to 3 in the morning". In the second example you will output "10 minutes past 2 in the afternoon". If the string is not a valid time then you must output "invalid time". In your design you will need to work output how much variation there is in the strings produced, and what conditions in the input produce them.