Week 2 Exercises : Expressions

Jan 23rd 2012

The exercises in the second week of the course focus on transforming data from one representation to another. This is a core activity in programming where having data in one particular representation can make a problem solvable. Some kinds of transformation keep all of the information, and some throw a part of the information away. The first kind is called a *bijection* in mathematics, while the second kind has many names but has been referred to as *projection*, *folding* or *sampling* in different contexts. Both kinds of transformation are used in solving this week's exercises.

```
#include<iostream>
using namespace std;
int main(int argc, char **argv)
{
int num;
  cout << "Please enter an integer:" << endl;
  cin >> num;
  cout << num << " = " << (num/49) << "x49 + " << (num%49) << endl;
}
```

Listing 1: Division and remainder example

1. Type in the code from listing 1, compile it and run it. Test the program by entering different values and observing the results. The behaviour of division and remainders (or the *modulus*) can be used as part of the solution to the following exercises.

2. Write a program that accepts a 4-digit integer as input. You can assume the input is of the correct form. Output each digit in the integer on a separate line.

3. Write a program that inputs a time as a number of minutes, and then outputs the time as a number of hours and minutes.

4. Write a program that inputs a time as a number of seconds, and then outputs the time as a number of hours, minutes and seconds.

5. Write a program that performs conversion between two arbitrary number-bases. Your program should input a source base, then input four digits, and finally an output base. The number represented by the four digits should then be converted and output in the second base. There is an example of one possible execution of your program in Figure 1. Think about how you will test your program. Obviously it has to handle the case described in the figure. What parts of the input should you vary to make sure that it is working properly?

Base conversion is an example of a *bijection* that is used heavily in Information Theory, Cyptography and Data Compression. The modulus and division operators can be used for conversion between any number bases, but in some applications there are more specific methods of conversion that only work for certain bases. These may be chosen because they are more efficient, or because they relate to a particular memory organisation that saves space for the data. One class of special cases is when the target base is a power of 2, in particular in the cases $2^1 = 2$ (binary), $2^3 = 8$ (octal), $2^4 = 16$ (hexadecimal), $2^8 = 256$ (bytes), $2^{16} = 65536$ (shorts) and 2^{32} (words), which is roughly $4 \cdot 10^9$, that is 4-billion or 4-milliard depending on which part of the world you originate form.

```
Please input source base:
2
Please input four digits:
1
1
0
1
Please input target base:
10
The input number 1, 0, 1, 1 is 13 in base 10.
```

Figure 1: Example execution for Exercise 5.

6. Write a program that converts decimal integers to binary using logic operators. You may find it helpful to start with a copy of your answer to Exercise 5 and specialise it for the input and output base. Once you have the expressions for the digits using constant values and the modulo operator, think about how you can perform the same operation using the `AND` operator and shift operators.

7. Binary Coded Decimal (BCD) is used in electronics to drive seven-segment displays. Write a program that inputs a integer between 0 and 9999 and split it into four decimal digits. For BCD encoding one digit (0-9) is encoded into the lowest four bits of an integer (0-3). This means that the bit-patterns for 10-15 are unused. The next digit is encoded into bits (4-7), again with some redundant bit-patterns. Your program should output both the separate digits and the result encoded into a 32-bit unsigned integer.

8. Take your base conversion program and test it with some integer to binary conversions. Look for a pattern between the binary representation of a number and whether it is odd or even. Write a number program that inputs a integer and uses logical operators to round it down to the closest even number.

9. Write a program that inputs an integer and rounds it up to the nearest odd number.

10. Write a program that inputs an integer and rounds it to the nearest multiple of sixteen. For example: $3 \to 0$, $7 \to 0$, $8 \to 16$, $15 \to 16$, $32 \to 32$, $1023 \to 1024$ ...