CS 7A - Spring 2016 - Converting between base 2, base 10, base 90 and using encryption. Due 3/1/16

Background Theory

To protect your identity when your grades are posted to the internet, I've added a level of encryption that converts your student number (base 10) and converts it to ascii characters with numbers between 33 and 122 (these include 90 recognizable characters).

Suppose your student id number (in base 10, the way you get it) is 0123456 or, as a base 10 number, without the leading zero, $N = 123,456$. One way to find the base-10 digits is to repeatedly divide by 10, each time recording the remainder and reducing the dividend by replacing it with the quotient. For $N = 123,456$ the iterative (repeating) process produces these results:

Dividend	*Divisor*	*Quotient*	*Remainder*
123456	10	12345	6
12345	10	1234	5
1234	10	123	4
123	10	12	3
12	10	1	2
1	10	0	1

The coefficients of the powers of 10 that make up 123456 appear as the remainders with the least significant digit coming out first.

$$123456 = 10^5 + 2 \cdot 10^4 + 3 \cdot 10^3 + 4 \cdot 10^2 + 5 \cdot 10 + 6 = 10\,(10(10(10(1 \cdot 10 + 2) + 3) + 4) + 5) + 6$$

Note that the left hand form (123456) is a succinct representation that is computable by either the middle or the right formula, but the right formula is more efficient (involves many fewer multiplications).

This all seems fairly strait-forward and obvious, but the beauty is in the way this algorithm to extends to bases other than 10. For example, to convert from base 10 to base 2, simply record the remainders (least significant first) when the number is divided by 2.

Dividend	*Divisor*	*Quotient*	*Remainder*
123456	2	61728	0
61728	2	30864	0
30864	2	15432	0
15432	2	7716	0
7716	2	3858	0
3858	2	1929	0
1929	2	964	1
964	2	482	0
482	2	241	0
241	2	120	1
120	2	60	0
60	2	30	0
30	2	15	0
15	2	7	1
7	2	3	1
3	2	1	1
1	2	0	1

Thus $123456_{10} = 2^{16} + 2^{15} + 2^{14} + 2^{13} + 2^9 + 2^6 = 2^6(2^3(2^4(2(2(2(2+1)+1)+1)+1)+1)+1) = 11110001001000000_2$
Here is pseudocode for accomplishing the conversion from base 10 to base 2:

```
number = positive integer

while (number > 0 )
{
  bit    = number mod 2 ;
  number = number div 2 ;
  put bit to the left of any previous bits
}
```

In general, the binary form of a number can written in this form:

$$N = d_0 + d_1 \cdot 2 + d_2 \cdot 2^2 + \cdots + d_n \cdot 2^n$$

where the ith division by 2 and produces the remainder d_i in turn.
For instance, $237 = 1 + 0 \cdot 2 + 1 \cdot 4 + 1 \cdot 8 + 0 \cdot 16 + 1 \cdot 32 + 1 \cdot 64 + 1 \cdot 128 = 11101101_2$
Here is one way we could start writing some C++ code for this algorithm:

```
/** Geoff Hagopian
Converting from base 10 to base 20 using A=0,B=1,...,S=19 as place values. **/

#include "std_lib_facilities.h"

int main() {
    int x;
    cout << "\nEnter␣and␣integer␣to␣convert␣to␣base␣2\n";
    cin >> x;
    cout << x%2; //the remainder after division by 2
    x /= 2; //
    cout << x%2;
    x /= 2;
    cout << x%2;
    x /= 2;
    //...keep repeating this until it's done...how do we know?
}
```

After working a number of these conversions by hand and taking note of the what happens to `x` when it's repeatedly divided by 2 (recording the remainder at each step) we observe that we are done when the integer part of half of `x` is zero. This suggests using a `while` loop and condition the loop on `x` not equal to zero:

```
//beginning comments and preprocessor commands omitted for brevity

int main() {
    int x;
    cout << "\nEnter␣and␣integer␣to␣convert␣to␣base␣2\n";
    cin >> x;
    while(x!=0) { // note that "!" is the negation operator.
        cout << x%2; //the remainder after division by 2
        x /= 2; // integer division discards remainder, so 3/2 is 1.
    }
}
//--------------------------------------------------------------------------------
```

Entering 237 here yields the following

```
Enter and integer to convert to base 2
237
10110111
```

This is fine except the the digits are in reverse order. We fixed this by storing the digits as `char`acters in a `string`. To do this learned how to **cast** an integer as a character. The ASCII character `'0'` has the integer value 48, so `char(48);` will print `'0'` by 'casting' the integer 48 as the `char`acter `'0'`. Starting with an empty string, `binary`, we can use concatenation to append either a `'1'` or a `'0'` with the command

`binary = char(48+x%2)+binary`,

which adds to the left side of the string at each iteration:

```
int main() {
    int x;
    string binary;
    cout << "\nEnter␣and␣integer␣to␣convert␣to␣base␣2\n";
    cin >> x;
    while(x!=0) { // note that "!" is the negation operator.
        binary = char(48+x%2)+binary; // can't use += here, why not?
        x = x/2;
    }
}
//-----------------------------------------------------------------------------
```

Questions:

1. Adapt this code to convert from base 10 to base 90 and use this to convert *the square* of your student number to a base 90 "number" (string of characters) corresponding to ascii values between 33 and 122 (inclusive). All of these characters are recognizable (printable), so that the remainders after division by 90 are translated as follows: `0` $\rightarrow$ `char(33) = '!'`, `1` $\rightarrow$ `char(34) = '"'`, `2` $\rightarrow$ `char(35) = '#'`, ... , `89` $\rightarrow$ `char(122) = 'z'`.

   For example, suppose your student number is 123456. Then the square is
   $$\begin{aligned} 15241383936 &= 169348710 \cdot 90 + 36 \\ &= (1881652 \cdot 90 + 30) \cdot 90 + 36 \\ &= ((20907 \cdot 90 + 22) \cdot 90 + 30) \cdot 90 + 36 \\ &= (((232 \cdot 90 + 27) \cdot 90 + 22) \cdot 90 + 30) \cdot 90 + 36 \\ &= ((((2 \cdot 90 + 52) \cdot 90 + 27) \cdot 90 + 22) \cdot 90 + 30) \cdot 90 + 36 \\ &= (((((0 \cdot 90 + 2) \cdot 90 + 52) \cdot 90 + 27) \cdot 90 + 22) \cdot 90 + 30) \cdot 90 + 36 \end{aligned}$$
   Now the remainders are, in reverse order, 2,52,27,22,30 and 36. These need to be increased by 33, to get our ascii values: 35,85,60,55,63,69, which correspond to the string of ascii characters `#U<7?E`. So that would be the encrypted student number.

   Hint: You'll want to simplify matters by using a `while` loop something like this:

```
long long unsigned int N; //to hold the square of your student number
string codeWord; //to hold the encrypted number
while(N>0) {
    //cast the number 33 more than the remainder when N is divided by 90 as a cha
    //and add (concatenate) that number to codeWord
    //Replace N by the integer quotient when N is divided by 90
}
```

2. Now write code to check your result by reversing it, so we can recover the student number in base 10 from the encrypted version of its square. (Not a very secure encryption, is it?)

   Submit your code as `<your initials>_encryptSID.cpp` by 3/1/16.