Mathematics 1262 (C++ Programming) Hilary 2014

February 1, 2014

1 First assignment, due 24/1/14

Due using **submit-work** by 12 noon on Friday, January 24, 2014. Submit *either* program (A) or (B).

Program A. Write a C++ program to print out the following, using a for-loop.

```
7 divided by 7 is 1, remainder 0
8 divided by 7 is 1, remainder 1
9 divided by 7 is 1, remainder 2
and so on until
20 divided by 7 is 2, remainder 6
```

Try to get it exactly as shown, including the comma. Call your file **A01.cpp** (Conventionally, the ending `.cpp` or `.C` is used for C++ programs.)

Program B. The second version of a program in section 6 reads numbers and echoes them on the screen; the second version uses `cin.eof()` correctly. Write a program which behaves the same way as that program. If you just copy it, that's ok.

Call your file **B01.cpp**.

Use `submit-work` to submit your assignment.

Important. You only submit .cpp files, not executables. Usually it doesn't matter what they are called (except for the .cpp extension), but for this assignment it *does* matter. So please call your assignment **A01.cpp** or **B01.cpp**. If you don't, it's a nuisance.

Motivation: getting used to the system.

2 Second assignment, due 31/1/14

Write a C++ program which takes a sequence of quadruples, such as

```
1 1 0 6
24 1 14 5
25 12 13 3
31 12 13 2
29 8 5 1
20 4 10 2
7 7 11 4
20 4 14 0
5 3 14 3
```

These represent days in the 21st century, together with day-of-week (for example, 24/1/14 is a Friday). Your program should spell out these dates in full, such as

```
Friday, 24 January, 2014
```

Be careful about spacing — such as comma space 24 space January — and notice that the 2-digit date is expanded to 4 digits. You need to prefix single-digit years by 200 and double-digit years by 20.

Unlike the first assignment, the name of your program doesn't matter much.

Motivation. Using tables of character strings; reading multiple data. As previously, you should exercise care with `cin.eof()`.

3 Third assignment, due 7/2/14

There are two alternative assignments. If you choose assignment A, please submit it as **A03.cpp**, nothing else, not ass3.cpp nor a03.cpp nor a03.C nor A3.cpp nor anything else. If you choose assignment B, submit **B03.cpp**.

Assignment 3 A. Write an *efficient* routine `double pow ( double x, int n )` which applies the 'Russian peasant' strategy to the argument n (which is supposed to be nonnegative), and write a main program to run and test it.

Do not use the C/C++ math library function pow(). That is not the point.

You can just adapt the following code fragment.

```
double multiply ( double x, int n )
{
  double y;
  y = 0;
  while ( n > 0 )
```

```
  {
    if ( n % 2 == 1 )
      y += x;
    n = n/2;
    x = x + x;
  }

  return y;
}
```

```
sample run
3 4
x 3, n 4, xxx returns 81
1.5 3
x 1.5, n 3, xxx returns 3.375
2.2 4
x 2.2, n 4, xxx returns 23.4256
^D
```

Assignment 3 B. Write a routine `void do_gcd ( int m, int n )` which prints the gcd, *and also* expresses it as an integer linear combination of m and n. Here is a sample run. The output produces the linear combination and also evaluates it for purposes of comparison.

```
% a.out
1 2
gcd(1,2) = 1 = (1)(1) + (0)(2) = 1
34 89
gcd(34,89) = 1 = (-34)(34) + (13)(89) = 1
1001 11
gcd(1001,11) = 11 = (0)(1001) + (1)(11) = 11
77 1001
gcd(77,1001) = 77 = (1)(77) + (0)(1001) = 77
384 30
gcd(384,30) = 6 = (-1)(384) + (13)(30) = 6
^D
```

Here is how it is computed. Recall that the gcd calculation effectively produces a sequence

$$x_0 = m, x_1 = n, x_2, \ldots, x_{k+1} = 0$$

where $x_k \neq 0$; x_k is the gcd, and for $1 \leq j \leq k$,

$$x_{j+1} = x_{j-1} \mod x_j.$$

There are also two sequences r_j and s_j such that

$$x_j = r_j m + s_j n$$

$0 \le j \le k$ ($j = k + 1$ is of no interest). If $1 \le j \le k$, let $q = x_{j-1} \div x_j$ (integer division).

$$x_{j+1} = x_{j-1} - q x_j$$

Then

$$r_{j+1} = r_{j-1} - q r_j \quad \text{and} \quad s_{j+1} = s_{j-1} - q s_j.$$

One develops the r-sequence with three variables r_x, r_y, r_z, and likewise the s-sequence, just like the x-sequence is developed using x, y, and z. The only question is how to initialise r_x, s_x, r_y, s_y!

Include a `main()` procedure, of course, to run and test the code.

It is necesssary that the subroutine print the result, which is rather awkward; properly, the gcd, and the coefficients r and s, should be transmitted back to the calling program, but how to do so has not yet been covered in this course.

Motivation. Experimenting with functions and routines.