Homework 7 (due 3/13)

1. (100 pts.) Implement a basic **biguint integer** calculator using C++ and Qt. All values are of type `biguint` and overflow handling is not required. You must re-implement `biguint` using dynamic arrays to support values of any sizes.

   Your `biguint` class should support the following new member variable and functions (along with those required in homework 5):

```
private:

     size_type *data;

     size_type capacity;

     // data[0..capacity-1 is a dynamic array

     // data[i] corresponds to the coefficient of 10^i for 0 <= i < capacity

public:

// default constructor
// pre: none
// post: creates a biguint whose value is n and whose capacity is at least m
//       (default n = 0, m = 1)
// NOTE: a decimal integer n requires log(n+1.0)/log(10) (rounded UP) digits
biguint(unsigned int n = 0, size_type m = 1);

// destructor
~biguint();

// assigment operator
void operator =(const biguint & b);

// pre: none
// post: returns the number of digits (no leading 0's) in this biguint
size_type digits() const;

// division and remainder operators
// pre: b != 0
// post: returns the quotient/remainder of this biguint divided by divisor
// NOTE: also implement the corresponding nonmember operators / and %
void operator /= (const biguint & divisor);
void operator %= (const biguint & divisor);

// pre-increment/decrement operators
// pre: none
// post: this biguint has been incremented/decremented by 1
//       return value is the NEW value
biguint operator ++();
biguint operator --();

// post-increment/decrement operators
// pre: none
// post: this biguint has beeen incremented/decremented by 1
//       return value is the ORIGINAL value
```

```
biguint operator ++(int);  // there is actually no input paramenter
                           // int is used to distinguish between
                           // post-increment from pre-increment (which
                           // has no input parameters)
biguint operator --(int);
```

Your app should support the following features:

1. two editable `QLineEdit` items to display/enter the left and right operands;
2. an editable `QLineEdit` item to display/enter the memory value;
3. `QPushButton` items to support the following operations:
   a. addition;
   b. subtraction: if the left operand is smaller than the right operand, the result is 0;
   c. multiplication;
   d. division: the answer is rounded down to the nearest integer; output an error message to the app's status bar if the user attempts to divide by 0;
   e. remainder: output an error message to the app's status bar if the user attempts to take the reminder by 0;
   f. factorial: defined as $1 \times 2 \times \cdots \times$ left operand; by definition the factorial of 0 is 1;
   g. power: defined as (left operand)$^{\text{right operand}}$; by definition $0^0 = 1$;
   h. memory store: save the left operand to memory;
   i. memory recall: restore the memory value to the right operand.

   All results are saved in the left operand.
4. a `Quit` menu/toolbar item to terminate the app;
5. an `About` menu/toolbar item to display information about the app (author, usage, copyright information, etc.)

Notes:

1. The number of digits in decimal integer n is `ceil(log(n+1.0)/log(10))`; 0 has 0 digits!
2. The number of digits in `a+b` is at most `1+max(a.digits(), b.digits())`.
3. The number of digits in `a-b` is 0 if `a < b` or at most `a.digits()` if `a >= b`.
4. The number of digits in `a*b` is at most `a.digits() + b.digits()`.
5. The number of digits in `a/b` is 0 if `a < b` or at most `a.digits() - b.digits() + 1`.

   The digits of `a/b` can be computed from left to right (most to least significant):

   ```
   biguint remainder(0, divisor.digits());
   ```

```
biguint quotient(0, digits());     // too many, but can shrink later
for (int i = digits()-1; i >= 0; --i)
{
      remainder.ls();
      remainder.data[0] = data[i];
      size_type count(0);
      while (remainder >= divisor)
      {
             ++count;
             remainder -= divisor;
      }
      quotient[i] = count;
}
// at this point quotient = a/b and remainder = a % b
// shrink *this and copy quotient/remainder back to *this
```

Example:

```
376 / 26:  digits() = 3; divisor.digits() = 2

          remainder = 0
          quotient = 0
i = 2
          shift remainder left: remainder is 0
          remainder.data[0] = 3: remainder is 3
          remainder < divisor: quotient[2] = 0
i = 1
          shift remainder left: remainder is 30
          remainder.data[0] = 7:  remainder is 37
          remainder > divisor: remainder is 37-26 = 11
          remainder < divisor: quotient[1] = 1
i = 0
          shift remainder left: remainder is 110
          remainder.data[0] = 6: remainder is 116
          remainder > divisor: remainder is 116-26 = 90
          remainder > divisor: remainder = 90-26: 64
          remainder > divisor: remainder = 64-26: 38
          remainder > divisor: remainder = 38-26: 12
          remainder < divisor: quotient[0] = 4

quotient = 014
remainder = 12
```

6. The number of digits in `a%b` is at most `b.digits()`. Same algorithm as above.

7. member function pre-increment is implemented as follows:

```
biguint biguint::operator ++()
{
     *this += 1;
     return *this;
}
```

8. member function post-increment is implemented as follows:

```
biguint biguint::operator ++(int)
{
     biguint ans(*this);
     *this += 1;
     return ans;
}
```

Example:

1. 100! =

   933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000

2. 99^{99} =

   369729637649726772657187905628805440595668764281741102430259972423552570455277523421410650010128232727940978889548326540119429996769494359451621570193644014418071060667659301384999779999159200499899

3. (174^{110}) % 221=220

4. (3^{6532}) % 104513 = 91380

You must turn in by noon of the due date:

- a hard copy of your code with sample output for each example; and
- send **one** email message with subject line: `HW7 Your_last_name Your_section` to `cs60@math.scu.edu` with *all* files (`.pro, .cpp, .h, .ui`) attached.