National Chiao Tung University　　　　　　　　　　PRO_HW #1
Department of Electrical Computer and Engineering　　　　April 11 2015

UEE 1303(1067/1069): Object-Oriented Programming

Programming HW #1

Due: 2015/4/26 23:59

1. (50%) Of the many techniques for compressing the contents of a file, one of the simplest and fastest is known as *run-length encoding*. This technique compress a file by replacing sequences of identical bytes by a pair of bytes: a repetition count followed by a byte to be repeated. For example, suppose that the file to be compressed begins with the following sequence of bytes (shown in hexadecimal)
   46 6F 6F 20 62 61 72 21 21 21 20 20 20 20 20
   The compressed file contain the following bytes:
   01 46 02 6F 01 20 01 62 01 61 01 72 03 21 05 20
   Run-length encoding works well if the original file contains many long sequences of identical bytes. In the worst case (a file with no repeated bytes), run-length encoding can actually double the length of the file.
   a) Write a program named `compress_file` that uses run-length encoding to compress a file. To run `compress_file`, we'd use a command of the
      `compress_file` *original-file*
      `compress_file` will write the compressed version of *original-file* to *original-file*`.rle`.
      For example, the command
      `compress_file foo.txt`
      will cause `compress_file` to write a compressed version of `foo.txt` to a file named `foo.txt.rle`.
   b) Write a program named `uncompress_file` that reverses the compression performed by the `compress_file` program. The `uncompress_file` command will have the form
      `uncompress_file` *compressed-file*
      compressed-file should have the extension `.rle`. For example, the command
      `uncompress_file foo.txt.rle`
      will cause `uncompress_file` to open the file `foo.txt.rle` and write an uncompressed version of its contents to `foo.txt`. Program `uncompress_file` should display an error message if its command-line argument doesn't write the `.rle` extension.

2. (50%) In C++, the largest `int` value is 2147483647. So, an integer larger than this cannot be stored and processed as an integer. Similarly, if the sum or product of two positive integers is greater than 2147483647, the result will be incorrect. One way to store and manipulate large integers is to store each individual digit of the number in an array. Write a program that inputs two positive integers of, at most, 20 digits and outputs the sum of the numbers. You have to process input file "`bignumber.txt`" to obtain two big numbers and output the results of arithmetic operations to "`result.txt`.".

   Type the following command to execute the program:

```
> ./bigN bignumber.txt result.txt
```

   The content of the "`bignumber.txt`" sample file is shown as follows. The first line and the second line indicate the big number A and B, respectively.

```
5467755
12443
```

   The content of the "`result.txt`" file is shown as follows:

```
A + B = 5480198
A – B = 5455312
```

   Requirement

   You have to finish this exercise using the data structure `BIGNUMBER` defined in `bignum.h` and write three more functions of your own: `readNumber()`, `bigNumberOperation()` and `writeResults()` in `bignum.cpp`.

```
// bignum.h
typedef struct
{
   int *data;
   int length;
   bool sign;
}BIGNUMBER;
```

```
// hw1-2.cpp
int main(int argc, char *argv[])
{
   BIGNUMBER a, b;
   readFile(argv[1], a, b);
   BIGNUMBER results[2];
   results[0] = bigNumberOperation(a,b,'+');
```

```
    results[1] = bigNumberOperation(a,b,'-');
    writeResults(argv[2], results);
    return 0;
}
```