**CSCI 689** **Computer Assignment 2** **Spring 2016**
(30 points)

## Perfect Numbers

Greek mathematicians took a special interest in numbers that are equal to the sum of their **proper divisors,** which is simply any divisor less than the number itself. They called such numbers **perfect numbers.** For example, 6 is a perfect number because it is the sum of 1, 2, and 3, which are integers less than 6 that divide evenly into 6. Similarly, 28 is a perfect number because it is sum of 1, 2, 4, 7, and 14.

Write a predicate function isPerfect that takes an unsigned integer n and returns true if n is perfect, and false otherwise. Test your implementation by writing a main program in C++ that uses the isPerfect function to check for perfect numbers in the range 1 to 9999 by testing each number in turn. When a perfect number is found, your program displays it on stdout and also displays its divisors. The first two lines of the output should be 6 = 1 + 2 + 3 and 28 = 1 + 2 + 4 + 7 + 14. Your program should find two other perfect numbers in the range as well.

You can name your source/header files anything you want as far as they have proper extensions: .cc for source files and .h for header files. Guard the statements in your header file using the following format. (This is necessary because you don't want the statements in a header file are processed more than once.)

```
#ifndef CONSTANT-VALUE  // which is not defined any place else
#define CONSTANT-VALUE // same const value as for ifndef directive
// put all statements for your header file here
#endif
```

Include all system header files (that you need in your program) in your header files. For example, to gain access to the iostream library, which defines a set of simple I/O operations, insert the line #include <iostream> in your header files, and at the top of each source file, insert corresponding header files by the following statement: #include "header-file.h". Define the constant value 9999 as an unsigned integer and put its definition in the program header file. Also put the prototype of the predicate function isPerfect in the header file as well.

Each perfect number n should be displayed as $n = d_1 + d_2 + \ldots + d_m$, where $d_1, d_2, \ldots, d_m$ are the divisors of n with $d_1 = 1$. Generate such sequence as a C++ string by a divisors function. To use the strings in your program, you need to insert the line #include <string> in your header file, and to convert each divisor (an integer) to a string, you can use the conversion function to_string from the C++ library. You also need to include the prototype of divisors function in your header file.

To compile your source file and link its object file with the system library routines, you need to create a makefile. Insert the statements in this file in the following format, where the first line defines a macro that includes several options we use for the C++ compiler, and in the rest each entry consists of a line containing a colon (the dependency line), and one/more command lines beginning with a tab. To the left of the colon on the

dependency line is a target (an executable file); to the right of the colon are the target's prerequisites. For target, you can choose any valid name, and the advantages of using a makefile will be discussed in class. After creating this file, simply execute the UNIX make command without any arguments.

```
OPT = -std=c++11 -c -g -Wall -Wextra

program-executable-file.exe: program-object-file.o
	g++ -o program-executable-file.exe program-object-file.o

program-object-file.o: program-source-file.cc program-header-file.h
	g++ ${OPT} program-source-file.cc

execute:
	program-executable-file.exe | tee program-output-file

clean:
	/bin/rm -f *.o *.exe
```

For a final test of your program, execute it as: make execute. When the execution is successful, you will see the four perfect numbers on your computer screen and the output will also be stored in the output file program-output-file. You can find the correct output in file prog2.out, which is in directory: ~cs689/progs/16s/p2. After you are done, you don't need the object and executable files any more. To delete them, execute: make clean.

Submit your source and header files to your TA by executing: mail_prog program-source-file.cc program-header-file.h.