C SC 397a, Spring 2010
Assignment 3
Due: Monday, February 8 at 22:00:00

Problem 1. (30 points) ctest.cc

Starting with either your version or the instructor's version of ctest.java (in $FILES/a3[1]), *transliterate* ctest.java into C++. That is, turn it into C++ code that corresponds to the Java code very closely. Match the Java version in terms of methods, parameters, and control structures. For example, if "ctest 1" causes a method do_1() to be invoked in your Java version, then ctest 1 should cause a do_1() member function to be invoked in your C++ version. Use new to create all instances of Counter, to cause them to reside in the heap. Deviate slightly from Java for the ctest 1 case: to avoid having all those Counters in memory at once, immediately follow new Counter... with a delete. Use the Counter class from slide 14, incorporating it into your ctest.cc. Do not have separate Counter.h and Counter.cc files; ctest.cc should be self-contained.

Time three executions each of "ctest 1", "ctest 2", and "ctest 3" and compute average running times. Submit a plain text file timings.txt with the averages, using the same format as assignment 1.

For the timed executions, be sure to compile with optimization:

```
g++ -O -o ctest ctest.cc      # '-o ctest' names executable 'ctest' instead of 'a.out'
```

Note that handling of command line arguments in C++ is identical to C. As an example, here's a C++ program that expects two command line arguments, an integer N and a string, and prints them:

```
#include <cstdio>
#include <cstdlib>  // for atoi

int main(int argc, char **argv)
{
   if (argc != 3) {
      fprintf(stderr, "Usage: %s N S\n", argv[0]);
      exit(1);
      }

   int n = atoi(argv[1]);
   char *s = argv[2];

   printf("n = %d, s = '%s'\n", n, s);
}
```

Compilation and execution:

```
$ g++ printargs.cc
$ a.out 10 ten
n = 10, s = 'ten'
$
```

[1] $FILES represents the directory /cs/www/classes/cs397a/spring10/files

For the time being, if you wish to do string comparisons I recommend including <cstring> and using strcmp(), just like in C, rather than working with the string class, which we'll be learning about later.

Problem 2. (10 points) ctestx.cc

Based on the assignment 1 estimates, you may have found the performance of the C++ version of ctest to be disappointing. In this problem you are to produce a C++ version that has significantly better performance for the "ctest 1" case by changing the way a Counter represents its name: instead of using the library's string class, use a simple char array and assume that names will be at most ten characters in length. To be sure you don't have uses of string laying around, delete #include <string>.

To earn any points on this problem, "ctestx 1" must use less than 3.5 seconds of "user" CPU time on lectura. Just like for ctest.cc, all instances of Counter must reside in the heap.

Be sure to compile ctestx.cc with optimization (g++ -O ...).

Problem 3. (20 points) 1000.cc *[Puzzle Problem]*

Write a program that prints the integers from 1 through 1000, one per line. **RESTRICTION: Your solution may not use any control structures (if, while, for, ?:, etc.), goto statements, local variables, be recursive, or use external means such as a call to system(3) or popen(3).**

This is a "Puzzle Problem"; review the Puzzle Problem section in assignment 2 for advice and rules.

Miscellaneous

Feel free to use comments to document your source code as you see fit, but note that no comments are required.

You should be able to complete this assignment using the material on slides 1-68.

Don't hesitate to ask me for hints and/or help if you have trouble with a problem.

$FILES/a3 has reference versions of ctest, ctestx, and 1000.

Deliverables

Use `turnin` with the tag 397a_3 to submit your solutions for grading. The deliverables for this assignment are ctest.cc, timings.txt, ctestx.cc, and 1000.cc.