C SC 397a, Spring 2010
Assignment 4
Due: Monday, February 22 at 22:00:00

Problem 1. (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.

Problem 2. (80 points) Range

Based on customer feedback, the sales department requests that the following changes be made to the Range class:

(1) Convert equal(), print(Range *ranges[ ]), and print_in_which() from global functions into static member functions.

(2) Add a static member function void set_format(const char *fmt) that establishes fmt as the printf format string to be used for all subsequent calls to print() for all ranges. Calling set_format with no arguments resets the format to the default format ("Range '%c': [%d..%d]\n"). Be sure to make a copy of the format string—the caller might reuse the buffer. The format string may be arbitrarily long.

Example:

```
Range a(1, 10, 'a');
a.print();              // Output:      Range 'a': [1..10]

Range::set_format("%c: %d-%d\n");
a.print();              // Output:      a: 1-10

Range::set_format();  // use default
a.print();              // Output:      Range 'a': [1..10]

char buffer[ ] = "The range of %c is from %d to %d\n";

Range::set_format(buffer);
strcpy(buffer, "x");    // Clobber the buffer; shouldn't affect the format
a.print();              // Output:      The range of a is from 1 to 10

Range(1,2,'t').set_format("%c %d %d\n");
a.print();              // Output:      a 1 10
```

You may assume that the format specifiers are in the order used by default format (%c, %d, %d). For example, you will not see a call like this: set_format("%d %d %c\n")

(3) Add a static member function show_counts() that prints the number of times that the contains(int), span(), name(), and print() member functions have been called thus far during execution of the program. **<u>Important twist: Calls to contains(int), span(), name(), and print() made in Range member functions are not counted!</u>**

Example:

```
int main()
{
   Range r1(1,2,'1'), r2(3,4,'2');
   r1.print();
   r2.print();

   int n = 10;
   while (n--) {
      r1.contains(n);
      r1.span();
      r1.name();
      }

   Range *rs[ ] = {&r1, &r2, 0};

   Range::print(rs);                    // Calls to contains(), span(), print()
   Range::equal(&r1, &r2);              //   and name() rising from these calls are
   Range::print_in_which(0, rs);        //   are not counted.  (See below.)

   Range::show_counts();
}
```

Output:

```
Range '1': [1..2]
Range '2': [3..4]
Range '1': [1..2]
Range '2': [3..4]
Call counts for Range:
   Contains: 10
   Span: 10
   Name: 10
   Print: 2
```

Regarding the requirement that calls from inside Range member functions are not counted, note that the calls print(rs), equal(&r1, &r2), and print_in_which(0, rs) probably made use of contains(), span(), and print(), but the call counts correspond directly to the calls appearing in the code of the example. Note also that calls to the static member function print(Range *[ ]) are <u>not</u> counted.

You may distribute code between Range.h and Range.cc as you see fit.

<u>RESTRICTION: You may not use the C++ library string class in your solution. Instead, #include <cstring> and use strlen(), strcpy(), etc., and use new and delete for memory management.</u>

Implementation notes

You may want to use an array to hold the counts and use some sort of symbolic constant to reference array elements. You can use a #define for that, but enum provides a better solution. Here is an example of using an enum to symbolically reference the first or second value in a pair of values:

```
#include <cstdio>
class Pair {
   public:
      enum Which {First = 0, Second};  // Second is 1, a third would be 2, etc.
      Pair(int a, int b) {
         itsValues[First] = a;
         itsValues[Second] = b;
         }
      int get_value(Which n) {
         return itsValues[n];
         }
   private:
      int itsValues[2];
      };

int main()
{
   Pair p(10,20);

   printf("%d\n", p.get_value(Pair::Second));
}
```

Note that the enum definition must be public to allow it to be used in main(), and that the definition must precede usage of Which as a type, such as in get_value's parameter list. A copy of the example above is in $FILES/a4/enum.cc.

The technique discussed in conjunction with slide 56—using constructors and destructors to perform operations at the beginning and end of a block of code—can be used to produce an interesting mechanism to suspend call counting at appropriate times. This approach will require introduction of another class, like Hourglass on slide 56, and that's perfectly acceptable. Further, you might find it useful for that class to be a friend of Range.

Problem 3. (15 points) get_bounds.txt

The Range User's Group (the RUG) has requested a way to query a range for its bounds. The RUG proposes this member function:

```
int *get_bounds() {
   int *bounds = new int[2];
   bounds[0] = itsLow;
   bounds[1] = itsHigh;
   return bounds;
   }
```

The VP of Engineering thinks the RUG proposal is weak. Cite a weakness in the above solution and devise a better member function to query a range for its bounds in a single call. Show the code for both your version of get_bounds and an example of its use in get_bounds.txt, a plain-text file. Note that your solution will not be a compilable file—think of it as an e-mail message proposing your idea.

Problem 4. (15 points) assign.h

Implement an inline routine named assign that mimics the assignment operator, for ints. Example:

```
"#include "assign.h"
int main()
{
   int i = 2, j = 3, k;

   assign(k, i);  // Just like k = i
   printf("i = %d, j = %d, k = %d\n", i, j, k);

   assign(i, k + j); // Like i = k + j
   printf("i = %d, j = %d, k = %d\n", i, j, k);

   assign(i, assign(j, 10)); // Like i = j = 10
   printf("i = %d, j = %d, k = %d\n", i, j, k);

   assign(j, 5) = 2;  // Like (j = 5) = 2
   printf("i = %d, j = %d, k = %d\n", i, j, k);

}
```

Output:

```
i = 2, j = 3, k = 2
i = 5, j = 3, k = 2
i = 10, j = 10, k = 2
i = 10, j = 2, k = 2
```

Note that your solution is to be a C++ source code file, assign.h, that contains the definition for assign, and nothing else.

Problem 5. (20 points) inline.txt

This problem is a hands-on exercise to explore the effect of the inline specifier. Here is $FILES/a4/inline.cc:

```
#include <cstdio>
#include <cstdlib>

class X {
   public:
      X(int a, int b);
      int getA();
      int getB();
   private:
      int itsA, itsB;
   };
```

```
IL X::X(int a, int b) { itsA = a; itsB = b; }

IL int X::getA() { return itsA; }

IL int X::getB() { return itsB; }

IL int sum(int a, int b) { return a+b; }

int main(int argc, char **argv)
{
   int N = atoi(argv[1]);
   int s;
   for (int i = 1; i <= N; i++) {
      for (int j = 1; j <= N; j++) {
         X x(i,j);
         s = sum(x.getA(), x.getB());
         }
      }
   printf("s = %d\n", s);
}
```

The string IL is intended to be replaced via the g++ -D command-line option, which is equivalent to adding a #define.

Copy inline.cc to your directory. Compile it with optimization on (-O), and without inlining (-DIL=). Then, compute the average "user" CPU time of three runs, specifying the argument 3000 for each:

```
$ g++ -O -DIL=  -o out inline.cc
$ time ./out 30000
...
```

Repeat, with optimization on, and with inlining:

```
$ g++ -O -DIL=inline -o in inline.cc
$ time ./in 30000
...
```

Remember that the -E option of g++ can be used to see the effect of things like -DIL=inline.

Use the -S option to generate assembly code, both with and without inlining:

```
$ g++ -S -O -DIL=inline -o in.s inline.cc
$ g++ -S -O -DIL= -o out.s inline.cc
```

When done, create a text file named inline.txt with answers to these questions:

(1) What was the average execution time for the inlined, and non-inlined versions of the program?

(2) Examine in.s and out.s with a text editor and search for occurrences of "call", the instruction used to call a subroutine. What do you observe about occurrences of the call instruction in the two versions?

(3) For five points of extra credit explain why the argument 30000 is passed on the command line rather than just being wired in, like this: int N = 30000;

Problem 6. (15 points) answers.txt

Create a text file named answers.txt with answers to the following questions, which assume that X is a class.

(1) Imagining a class X, consider the declaration X xs[3];. In what order are the three elements of xs constructed? In what order are they destroyed?

(2) As you know, C++ access specifiers like public: and private: apply to however many members follow them before the next access specifier. Java requires public and private to appear on a member-by-member basis. Decide which of these two design choices you like best and present a short argument in favor of that style.

(3) Why does an uninitialized reference declaration (like int& i;) simply make no sense?

Problem 7. (Extra Credit) extra.txt

Submit a plain text file named extra.txt with the following.

(a) (1 point extra credit) Estimate how long it took you to complete this assignment. Other comments about the assignment are welcome, too. I appreciate all feedback, favorable or not.

(b) (1-3 points extra credit) Cite an interesting course-related observation (or observations) that you made while working on the assignment. The observation should have at least a little bit of depth.

Think of me saying "Good!" as one point, "Interesting!" as two points, and "Wow!" as three points. I'm looking for quality, not quantity.

Test programs and grading

$FILES/a4/1000 is a reference version for 1000.cc. This problem is all or nothing—you must meet the restrictions and exactly match the reference version output to earn all twenty points; otherwise, it's a zero.

$FILES/a4 has test programs for Range (rt*.cc) and assign (assign*.cc). An executable reference version is present for each test program. Each source file has a comment of the form // Value: N points that indicates the value of that test program. To earn the specified points, your solution, when compiled together with the test program, must produce exactly the same output as the corresponding reference version. <u>**If there's any difference, even a single character, you'll lose all the points for that test program.**</u>

Set of test programs will not change after Tuesday, February 9, at 17:00 unless errors are discovered.

Here's a bash script that runs the full set of tests for Range:

```
FILES=/cs/www/classes/cs397a/spring10/files
for i in $(seq 0 6)
do
    echo Testing $i...
    rt=rt$i
```

```
        rm -f $rt
        g++ -o $rt $rt.cc Range.cc
        $rt > $rt.mine
        $FILES/a4/tests/$rt > $rt.whm
        diff -c $rt.whm $rt.mine
done
```

Note that process substitution in bash can be used to diff the output of both versions with a single command:

```
$ diff -c <(rt1) <($FILES/a4/tests/rt1)
```

Miscellaneous

Be sure to heed the restrictions in problems 1 and 2.

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 1-94.

Don't hesitate to ask the instructor for hints and/or help if you have trouble with a problem or need help with tools, like bash.

Deliverables

Use `turnin` with the tag 397a_4 to submit your solutions for grading. The deliverables for this assignment are 1000.cc, Range.h, Range.cc, get_bounds.txt, assign.h, inline.txt, answers.txt, and if you choose to submit it, extra.txt.