2020/09/30 - Prime Number Primer (Part 2)

2020年9月21日　23:55

SYNOPSIS

-Go over LAB 3B.

LAB 3B

-On Canvas, go to the "Lab 3.2" assignment and read the "lab3.html" file attached. All lab details will be there.

-Again... there is 4 parts. There are 2 due dates:
- ~~Sept 27 (Sun): Prime1, Prime2 (PRIME DETECTION)~~
- Oct 4 (Sun): Prime3, Prime4 (PRIMES + STL)

-ALL PARTS have given template code you need to copy to start the lab.
- Not really for later parts. Just C+P tbh...

SUBMISSION COMMANDS

LAB 3.1

```
tar -cvf lab3_1.tar Prime1.cpp Prime2.cpp
```

LAB 3.2

```
tar -cvf lab3_2.tar Prime3.cpp Prime4.cpp
```

PRIME 3

- Based on PRIME2, so C+P it on over.
- Only one change needed for isprime class:
  - Replace std::find with std::binary_search. This function returns a bool so you don't need to compare result with cache.end() (or whatever you named your vector/list).
- Most changes occur in main function.
- No longer reading in numbers via stdin.
- Generate N numbers where N ≥ 1 AND N ≤ 140.
  - (rand() % 140) + 1
- Make two vectors of size N:
  - vector<int> numbers: Store generated numbers.
  - vector<bool> prime : prime[i] is TRUE if numbers[i] is prime. FALSE otherwise.
- Your mission (should you choose to accept it) is to fill both of those vectors via STL. aka. std::generate and std::transform.
  - Use vector.resize on both. Set to N.
  - Make a function (or lambda, is OK) that returns a number between 1 & 140 (inclusively). Let's call it random_number().

- Call std::generate on numbers vector, set the generator function to the random_number() function discussed above.

- Call std::transform on prime (output) based on numbers (input) using the functor created in Prime2.

  - Wait... what? Read on.

- STL BREAKDOWN

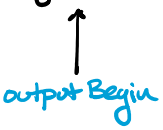
- generate(iterator begin, iterator end, function gen)
  - Will simply call gen on every element from start up to end.

- transform(iterator inputBegin, iterator inputEnd, iterator outputBegin, function op)
  - Runs op on input data (inputBegin → inputEnd) and stores data in output starting at outputBegin.

(Ex. numbers [1 2 3 4 5 6]

This is no mistake btw...

prime [ ]



```
prime[0] = op(numbers[0]);
prime[1] = op(numbers[1]);
```

is numbers[i] prime?
prime[i] is TRUE if so!

★ ANOMALIES

- Why no outputEnd? We assume input and output are same size.

- Why is inputEnd PAST the end? Because vector.end() points to element past the last. This is intentional, just like the following:

```
for (int i = 0; i < 10; i++)
```

It goes up to 10, but not including 10.

★ RESULT BTW

numbers [1 2 3 4 5 6]
prime [F T T F T F]

- count(iterator begin, iterator end, var value)
  - Trivial. Returns number of times "value" appears in a data structure ranging from "begin" to "end" iterators.
  - It's literally what the name implies...

prime [F T T F T F] = 3

just count number of "true"s.

PRIME4

- Write function to extract primes out of a vector into a new one.
  - Pass in 3 VECTORS:
    1. vector<int> &numbers
       GENERATED NUMBERS
    2. vector<bool> &is_prime
       WHETHER "NUMBERS" ARE PRIME
    3. vector<int> &primes
       ONLY PRIMES FROM "NUMBERS"

  - The first two are from Prime3. The last one is empty, and is filled via this function.
  - Only primes in "numbers" are copied into "primes". Do this by checking "is_prime" rather than via functor.
  - Thankfully, no STL tricks here. Just a naive loop.

(Ex

INIT

numbers = [1 2 3 4 5 6 7]
is_prime = [F T T F T F T]
primes = [ ]

Loop through all elements in "numbers".
If is_prime[i] is TRUE, add numbers[i] to "primes".

Thus,

numbers = [1 2 3 4 5 6 7]

isprime = [F T T F T F T]

THESE

RETURN

primes = [2 3 5 7]

- That was quite the explanation for a 3 line function...

- Write a function that will print a vector out to stdout (cout).

  - There is typo in writeup. Just do "setw(4)" to match solution executable.

- Sort vector of primes only. Then, OBLITERATE all duplicates via std::unique and vector.erase.

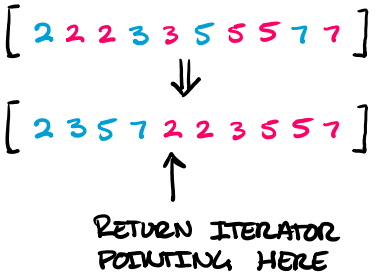
  - Sort is trivial with std::sort.

  - ANNIHILATING duplicates though...

  - std::unique will move unique elements to the front, duplicates to the back, and return an iterator to the first duplicate element.

(Ex.

BTW, FIRST IS BLUE, DUPLICATE IS RED. JUST SO YOU SEE.



- There's a few ways of fixing this, but the simplist is via vector ranged deletion:

```
vector.erase(begin, end);
```

We have the iterator to first duplicate, as well as vector.end(). Perform erasure between those.

- Print this vector via same vector printing function from earlier. Done.