Test assignment

Thank you for participating in our recruiting test. This will be a C++ programming test!

How to prepare for this test

Install an editor and C++ compiler of your choice. You may use C++11 language features but you don't have to. You do not need any other libraries besides the standard libraries for the task, and you are not allowed to use them.

You are free in your choice of operating system and development environment. The task is a very general programming assignment testing general problem structuring and programming proficiency. The solution has to be submitted within a **9 hour time frame**. Excellent C++ specialists will of course solve the problem in less than 9 hours, and we leave it to the candidates to test their solutions thoroughly before handing them in (in case they finish early).

The programming test is not a matter of diligence, and the quality of the solution does not depend on a vast amount of code. On the contrary, we expect our candidates to solve a non-trivial problem as elegantly as possible and to implement that solution in C++. We would like to receive the best solution you can come up with!

Task Description

interval_map<K,V> is a data structure that efficiently associates intervals of keys of type K with values of type V. Your task is to implement the assign member function of this data structure, which is outlined below.

interval_map<K, V> is implemented on top of std::map. In case you are not entirely sure which functions std::map provides, what they do and which guarantees they provide, we provide an excerpt of the C++1x draft standard here: ✚ More

Each key-value-pair (k,v) in the m_map member means that the value v is associated to the interval from k (including) to the next key (excluding) in m_map.

Example: the std::map (0,'A'), (3,'B'), (5,'A') represents the mapping

0 -> 'A'
1 -> 'A'
2 -> 'A'
3 -> 'B'
4 -> 'B'
5 -> 'A'
6 -> 'A'
7 -> 'A'
... all the way to numeric_limits<key>::max()
The representation in m_map must be canonical, that is, consecutive map entries must not have the same value: ..., (0,'A'), (3,'A'), ... is not allowed. Initially, the whole range of K is associated with a given initial value, passed to the constructor.

Key type K

- besides being copyable and assignable, is less-than comparable via operator<
- is bounded below, with the lowest value being std::numeric_limits<K>::lowest()

- does not implement any other operations, in particular no equality comparison or arithmetic operators

Value type V

- besides being copyable and assignable, is equality-comparable via operator==
- does not implement any other operations

You are given the following source code:

```
#include <assert.h>
#include <map>
#include <limits>

template<class K, class V>
class interval_map {
    friend void IntervalMapTest();

private:
    std::map<K,V> m_map;

public:
    // constructor associates whole range of K with val by inserting (K_min, val)
    // into the map
    interval_map( V const& val) {
        m_map.insert(m_map.begin(),std::make_pair(std::numeric_limits<K>::lowest(),val));
    };

    // Assign value val to interval [keyBegin, keyEnd).
    // Overwrite previous values in this interval.
    // Do not change values outside this interval.
    // Conforming to the C++ Standard Library conventions, the interval
    // includes keyBegin, but excludes keyEnd.
    // If !( keyBegin < keyEnd ), this designates an empty interval,
    // and assign must do nothing.
    void assign( K const& keyBegin, K const& keyEnd, const V& val ) {
```

```
     // 1. Check for the empty interval.
      if (!(keyBegin < keyEnd )) {
         // Key provides only < operator
         return;
      }

      // 2. Incoming range should be a valid range.
      // For e.g. IF we already have something like: -min-0: A, 0-10: B, 10-max: A
      // then assign(-1, 2, D) is not valid as keyBegin and keyEnd are not falling
      // in the allowed range. But assign(0, 5, D) is valid.
      auto kIter = m_map.upper_bound(keyBegin);
      if (m_map.end() != kIter) {
         if (kIter->first < keyEnd)
            return;
      }

      // 3. Check if incoming value is canonically same as the next value
      auto nextVal = (*this)[keyEnd];
      if (val == nextVal) {
         // Cannot insert the new value. It is canonically same as the next value.
```

```
    }

    // look-up of the value associated with key
```

```
    V const& operator[]( K const& key ) const {
        return ( --m_map.upper_bound(key) )->second;
    }
};

// Many solutions we receive are incorrect. Consider using a randomized test
// to discover the cases that your implementation does not handle correctly.
// We recommend to implement a function IntervalMapTest() here that tests the
// functionality of the interval_map, for example using a map of unsigned int
// intervals to char.

int main(int argc, char* argv[]) {
    IntervalMapTest();
    return 0;
}
```

You can download this source code here: Download

Your task is to implement the function "assign". Your implementation is graded by these criteria in this order:

- Correctness (of course): In particular, pay attention to the validity of iterators. It is illegal to dereference end iterators. Consider using a checking STL implementation such as the one shipped with Visual C++.
- Simplicity: Simple code is easy to understand and maintain, which is important in large projects. To write a simple solution, you need to exploit the structure of the problem. Use functions of std::map wherever you can.
- Running time: Imagine your implementation is part of a library, so it should be big-O optimal. In addition:
  - Do not make big-O more operations on K and V than necessary, because you do not know how fast operations on K/V are; remember that constructions, destructions and assignments are operations as well.
  - Do not make more than two operations of amortized O(log N), in contrast to O(1), running time, where N is the number of elements in m_map. Any operation that needs to find a position in the map "from scratch", without being given a nearby position, is such an operation.

  Otherwise favor simplicity over minor speed improvements.
- Time to turn in the solution: You should not take longer than 9 hours, but you may of course be faster. Do not rush, we would not give you this assignment if it were trivial.

You must develop the solution yourself. You may not let others help you or search for existing solutions. Of course you may use any documentation of the C++ language or the C++ Standard Library. Do not give your solution to others or make it public. It will entice others into sending in plagiarized solutions. If you use an online compiler, make sure that the privacy settings are set to private. Publishing the task description is considered cheating and will void your application.

When you are done, please complete the form and click Submit. Please do not send your solution by email, we will be automatically notified when you submit it here. You can resubmit improved solutions as often as you like. We will take the last submission as your final solution.

We received your solution at 13:57 UTC. Please give us a couple of days to review your solution and get back to you.

Submit