## **Test assignment**

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

### **How to prepare for this test**

### **Task Description**

interval\_map<K,V> is a data structure that associates keys of type K with values of type V. It is designed to be used efficiently in situations where intervals of consecutive keys are associated with the same value. 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. For more information on std::map, you may refer to [cppreference.com](https://en.cppreference.com/w/cpp/container/map" \t "https://server.think-cell.com/portal/en/_blank).

Each key-value-pair (k,v) in interval\_map<K,V>::m\_map means that the value v is associated with all keys from k (including) to the next key (excluding) in m\_map. The member interval\_map<K,V>::m\_valBegin holds the value that is associated with all keys less than the first key in m\_map.

Example: Let M be an instance of interval\_map<int,char> where

* M.m\_valBegin=='A',
* M.m\_map=={ (1,'B'), (3,'A') },

Then M represents the mapping

* ...
* -2 -> 'A'
* -1 -> 'A'
* 0 -> 'A'
* 1 -> 'B'
* 2 -> 'B'
* 3 -> 'A'
* 4 -> 'A'
* 5 -> 'A'
* ...

The representation in the std::map must be canonical, that is, consecutive map entries must not contain the same value: ..., (3,'A'), (5,'A'), ... is not allowed. Likewise, the first entry in m\_map must not contain the same value as m\_valBegin. Initially, the whole range of K is associated with a given initial value, passed to the constructor of the interval\_map<K,V> data structure.

****Key type**K**

* besides being copyable and assignable, is less-than comparable via operator<, and
* 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==, and
* does not implement any other operations.

## **First Evaluation**

#include <map>

template<typename K, typename V>

class interval\_map {

friend void IntervalMapTest();

V m\_valBegin;

std::map<K,V> m\_map;

public:

// constructor associates whole range of K with val

interval\_map(V const& val)

: m\_valBegin(val)

{}

// Assign value val to interval [keyBegin, keyEnd).

// Overwrite previous values in 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, V const& val ) {

if(!(keyBegin < keyEnd)) return;

auto it = m\_map.lower\_bound(keyEnd);

const auto pvalue = it == m\_map.begin() ? m\_valBegin : (--it)->second;

m\_map[keyBegin] = val;

m\_map[keyEnd] = pvalue;

auto st = m\_map.lower\_bound(keyBegin);

auto ed = m\_map.lower\_bound(keyEnd);

// erase slot

for(++ st; st != ed; )

st = m\_map.erase(st);

}

// look-up of the value associated with key

V const& operator[]( K const& key ) const {

auto it=m\_map.upper\_bound(key);

if(it==m\_map.begin()) {

return m\_valBegin;

} else {

return (--it)->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 test function that tests the functionality of

// the interval\_map, for example using a map of int intervals to char.

Unfortunately, this program failed to meet the criterion marked in red:

* ****Type requirements are met:**** You must adhere to the specification of the key and value type given above.
* ****Correctness:**** Your program should produce a working interval\_map with the behavior described above. 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++ or GCC.
* ****Canonicity:**** The representation in m\_map must be canonical.

****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 one operation of amortized O(log N), in contrast to O(1), running time, where N is the number of elements in m\_map.
  + Otherwise favor simplicity over minor speed improvements.

## **Second Chance**

Your first try failed. But we give you one more chance to fix your code:

## **Final Evaluation**

#include <map>

template<typename K, typename V>

class interval\_map {

friend void IntervalMapTest();

V m\_valBegin;

std::map<K,V> m\_map;

public:

// constructor associates whole range of K with val

interval\_map(V const& val)

: m\_valBegin(val)

{}

// Assign value val to interval [keyBegin, keyEnd).

// Overwrite previous values in 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, V const& val ) {

if(!(keyBegin < keyEnd)) return;

typename std::map<K,V>::iterator it = m\_map.lower\_bound(keyEnd);

V pvalue = it == m\_map.begin() ? m\_valBegin : (--it)->second;

m\_map[keyBegin] = val;

m\_map[keyEnd] = pvalue;

typename std::map<K,V>::iterator st = m\_map.lower\_bound(keyBegin);

typename std::map<K,V>::iterator ed = m\_map.lower\_bound(keyEnd);

// erase slot

for(++ st; st != ed; )

st = m\_map.erase(st);

}

// look-up of the value associated with key

V const& operator[]( K const& key ) const {

auto it=m\_map.upper\_bound(key);

if(it==m\_map.begin()) {

return m\_valBegin;

} else {

return (--it)->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 test function that tests the functionality of

// the interval\_map, for example using a map of int intervals to char.

****Unfortunately, this program also did not pass.****

This time, it failed to meet this criterion:

* ****Type requirements are met:**** You must adhere to the specification of the key and value type given above. For example, many solutions we receive use operations other than those that are explicitly stated in the task description. We have to reject many solutions because they assume that V is default-constructible, e.g., by using std::map::operator[].
* ****Correctness:**** Your program should produce a working interval\_map with the behavior described above. 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++ or GCC. Many solutions we receive do not create the data structure that was asked for, e.g., some interval ends up being associated with the wrong value. Others contain a code path that will eventually dereference an invalid or end iterator.
* ****Canonicity:**** The representation in m\_map must be canonical. Some solutions we receive create consecutive map entries containing the same value, or repeat the value of m\_valBegin at the beginning of the map.

****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 one operation of amortized O(log N), in contrast to O(1), running time, where N is the number of elements in m\_map.
  + Otherwise favor simplicity over minor speed improvements.

We regret that we cannot provide you with information specific to your solution, or with a correct version of the algorithm, because if we did, then we could no longer use this challenge for our interview process. We sincerely hope for your understanding on this matter.

Since this was your final submission, we have decided not to offer you an interview.

We want to thank you for your interest in the C++ developer position at think-cell and for the time and effort you have put into taking our programing test. We know that C++ developers are a scarce resource and that you have a choice of companies you can work for. For this reason we highly value your application.

Arno, our CTO, and the HR Team want to thank you for your time and interest in our company, and wish you the very best in your future career.