GEBZE TECHNICAL UNIVERSITY

CSE437
Real Time System Architectures

HOMEWORK 1
REPORT

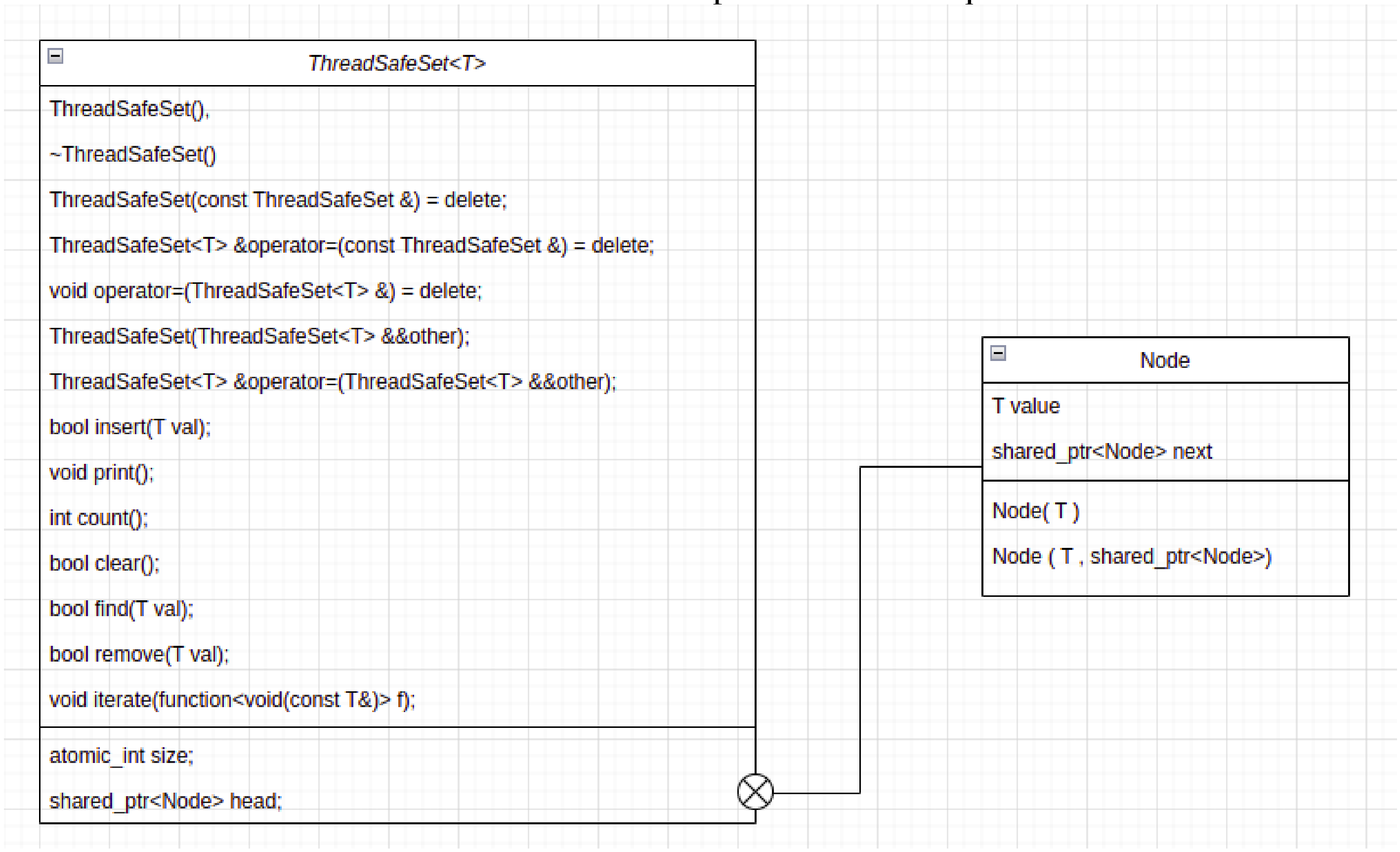
BARAN SOLMAZ
1801042601

Problem Defination:

- ThreadSafeSet class with template type,
- Class with insert,remove,find,clear,size and iterate functions,
- All functions shall be thread safe,
- Case 1: One reader thread and one writer thread,
- Case 2: Multiple reader and multiple writer threads,
- No mutex.

Problem Solution Approach:

I created a Linked List that acts like a set so that each time a new item is added, I compare the values as I move through the nodes. If it has the same value, I did not add it, otherwise I continued and added it until the value to be added is less than the current value. So I provided the sequential feature.

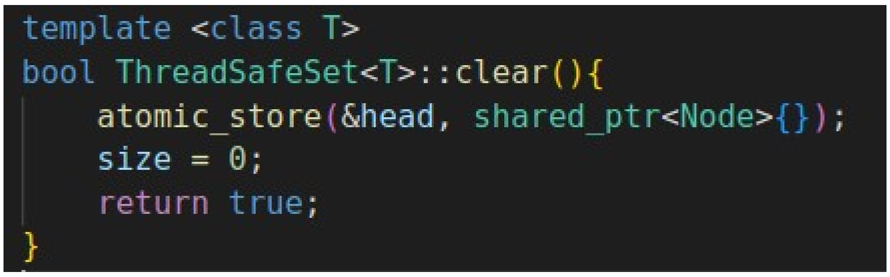

ThreadSafeSet<T>
ThreadSafeSet(),
~ThreadSafeSet()
ThreadSafeSet(const ThreadSafeSet &) = delete;
ThreadSafeSet<T> &operator=(const ThreadSafeSet &) = delete;
void operator=(ThreadSafeSet<T> &) = delete;
ThreadSafeSet(ThreadSafeSet<T> &&other);
ThreadSafeSet<T> &operator=(ThreadSafeSet<T> &&other);
bool insert(T val);
void print();
int count();
bool clear();
bool find(T val);
bool remove(T val);
void iterate(function<void(const T&)> f);
atomic_int size;
shared_ptr<Node> head;
Node
T value
shared_ptr<Node> next
Node(T)
Node (T , shared_ptr<Node>)

I kept all nodes as shared_ptr so I didn't lose memory at the end of the program.

I did all the node work atomically so that there is no interruption during the operations.

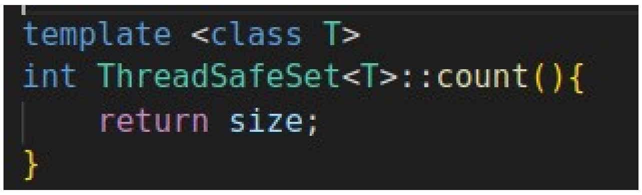
Implementations:

bool clear():



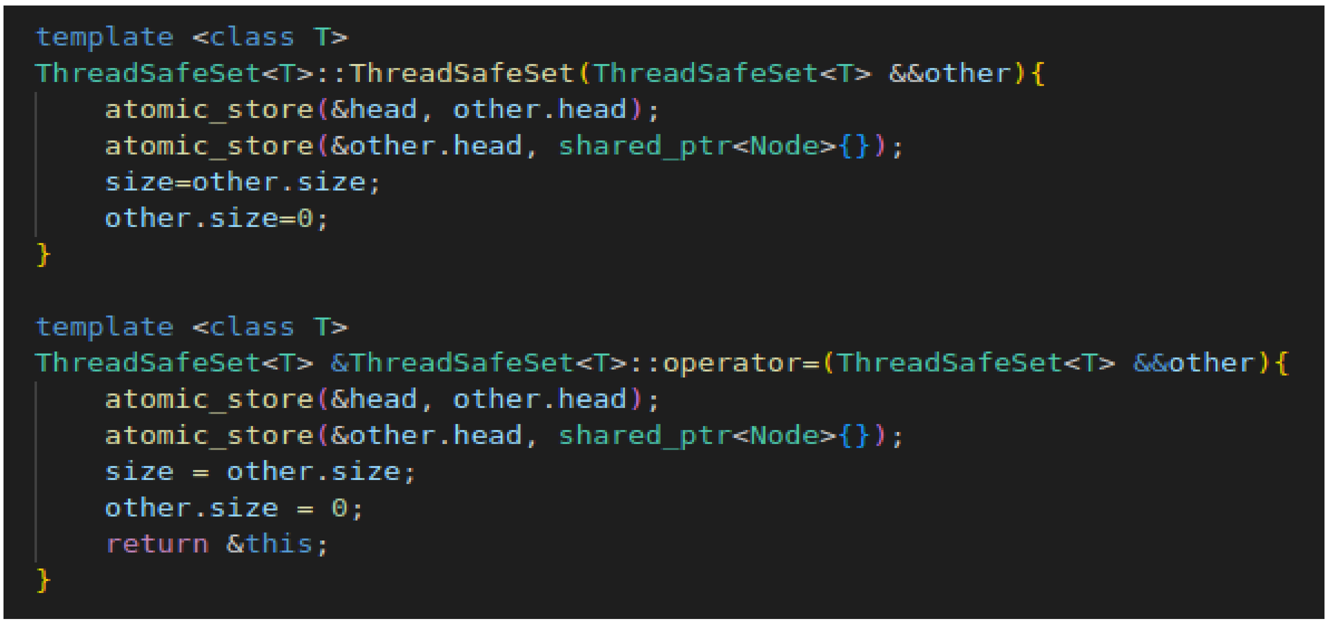
```
template <class T>
bool ThreadSafeSet<T>::clear(){
    atomic_store(&head, shared_ptr<Node>{});
    size = 0;
    return true;
}
```

int count():



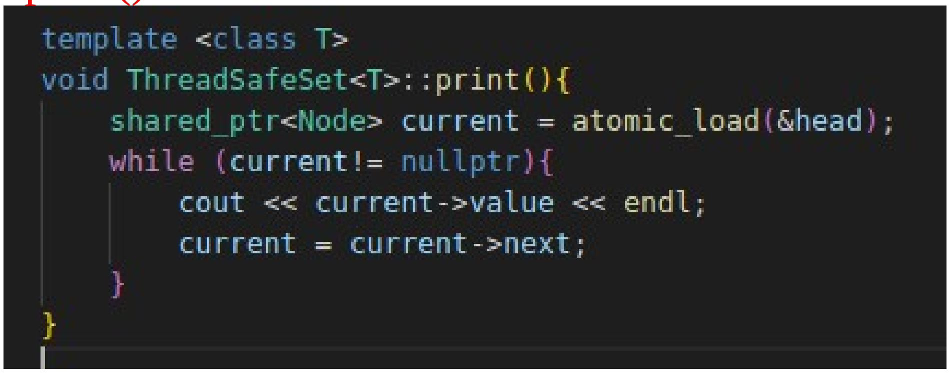
```
template <class T>
int ThreadSafeSet<T>::count(){
    return size;
}
```

Move Constructors:



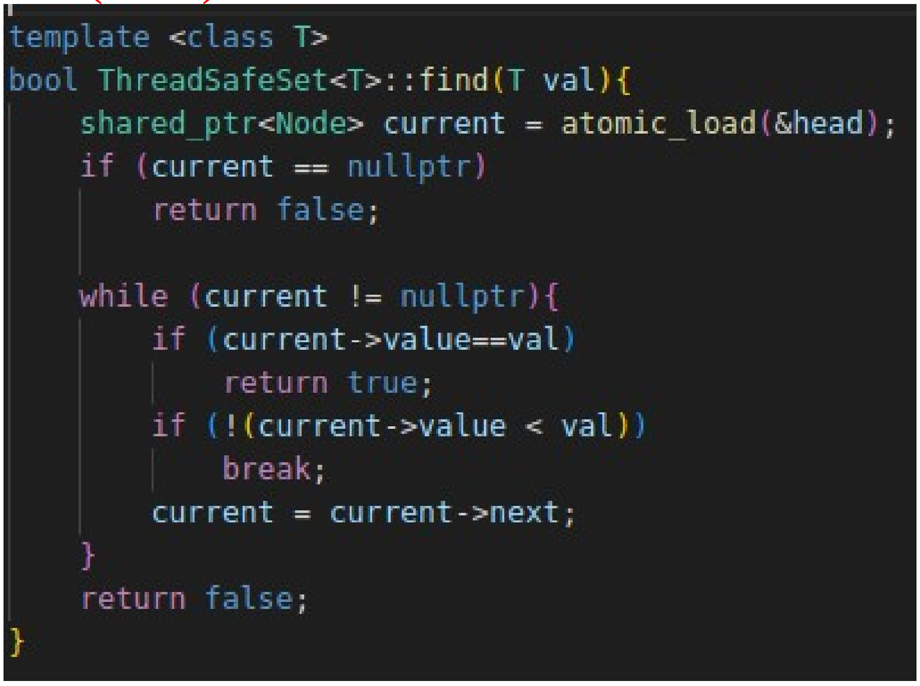
```
template <class T>
ThreadSafeSet<T>::ThreadSafeSet(ThreadSafeSet<T> &&other){
    atomic_store(&head, other.head);
    atomic_store(&other.head, shared_ptr<Node>{});
    size=other.size;
    other.size=0;
}

template <class T>
ThreadSafeSet<T> &ThreadSafeSet<T>::operator=(ThreadSafeSet<T> &&other){
    atomic_store(&head, other.head);
    atomic_store(&other.head, shared_ptr<Node>{});
    size = other.size;
    other.size = 0;
    return &this;
}
```

void print():



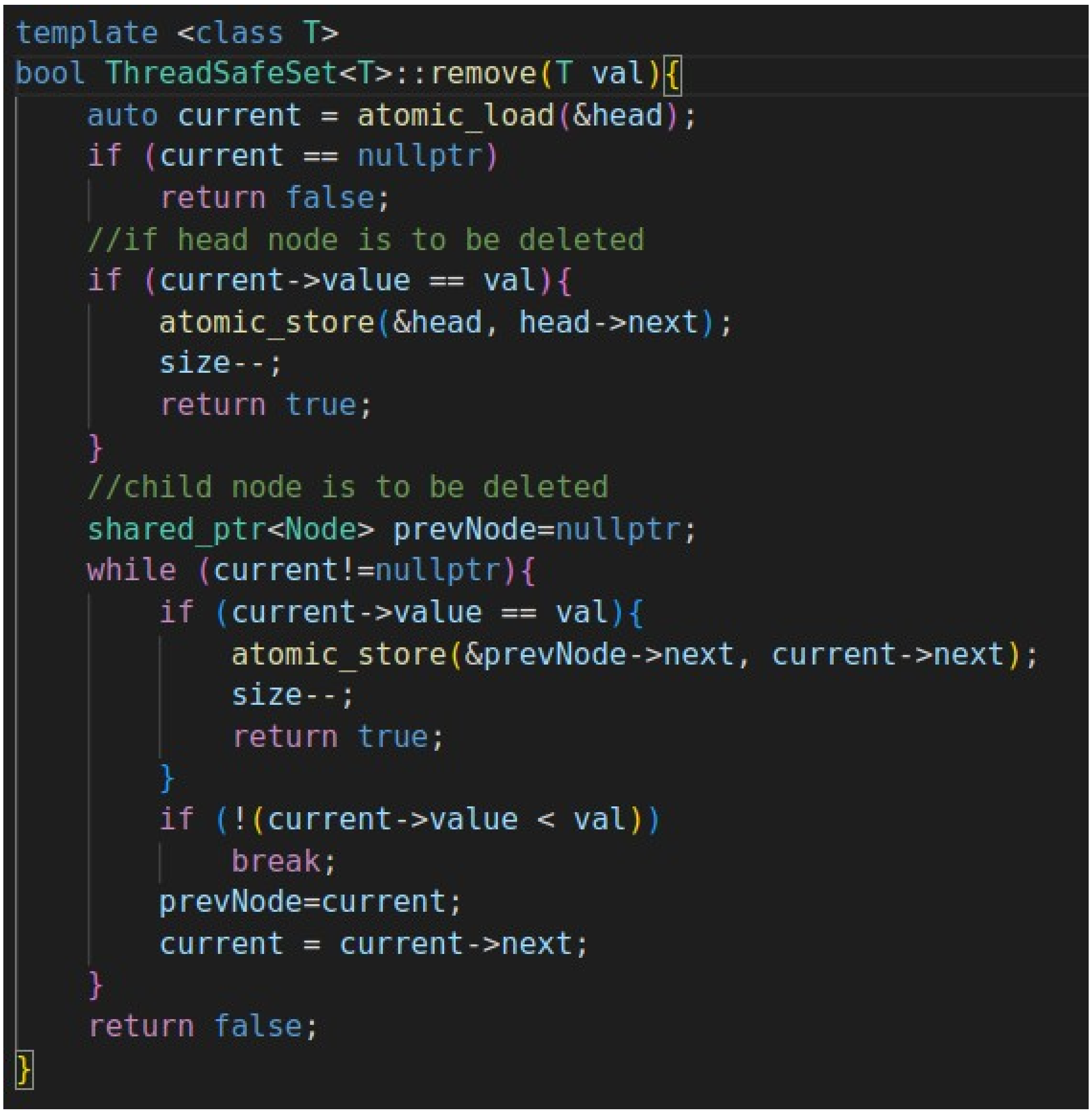
```
template <class T>
void ThreadSafeSet<T>::print(){
    shared_ptr<Node> current = atomic_load(&head);
    while (current!= nullptr){
        cout << current->value << endl;
        current = current->next;
    }
}
```

bool find(T val):



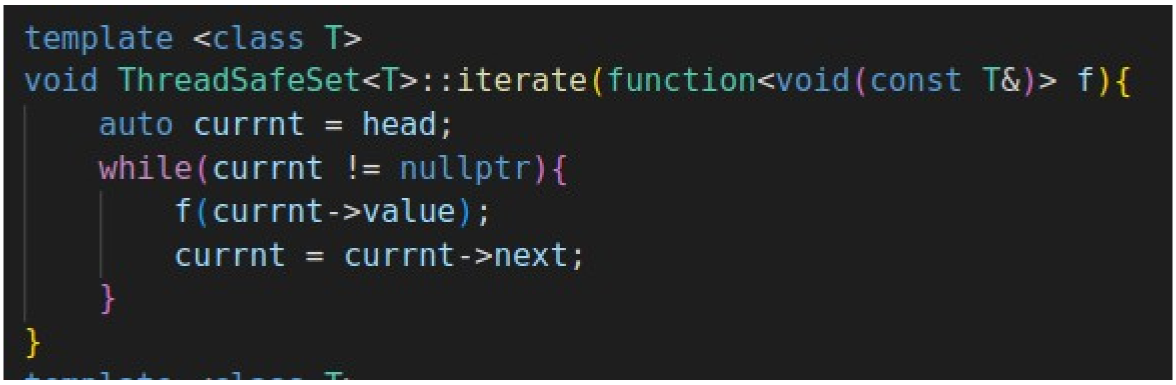
```cpp
template <class T>
bool ThreadSafeSet<T>::find(T val){
    shared_ptr<Node> current = atomic_load(&head);
    if (current == nullptr)
        return false;

    while (current != nullptr){
        if (current->value==val)
            return true;
        if (!(current->value < val))
            break;
        current = current->next;
    }
    return false;
}
```

bool remove(T val):



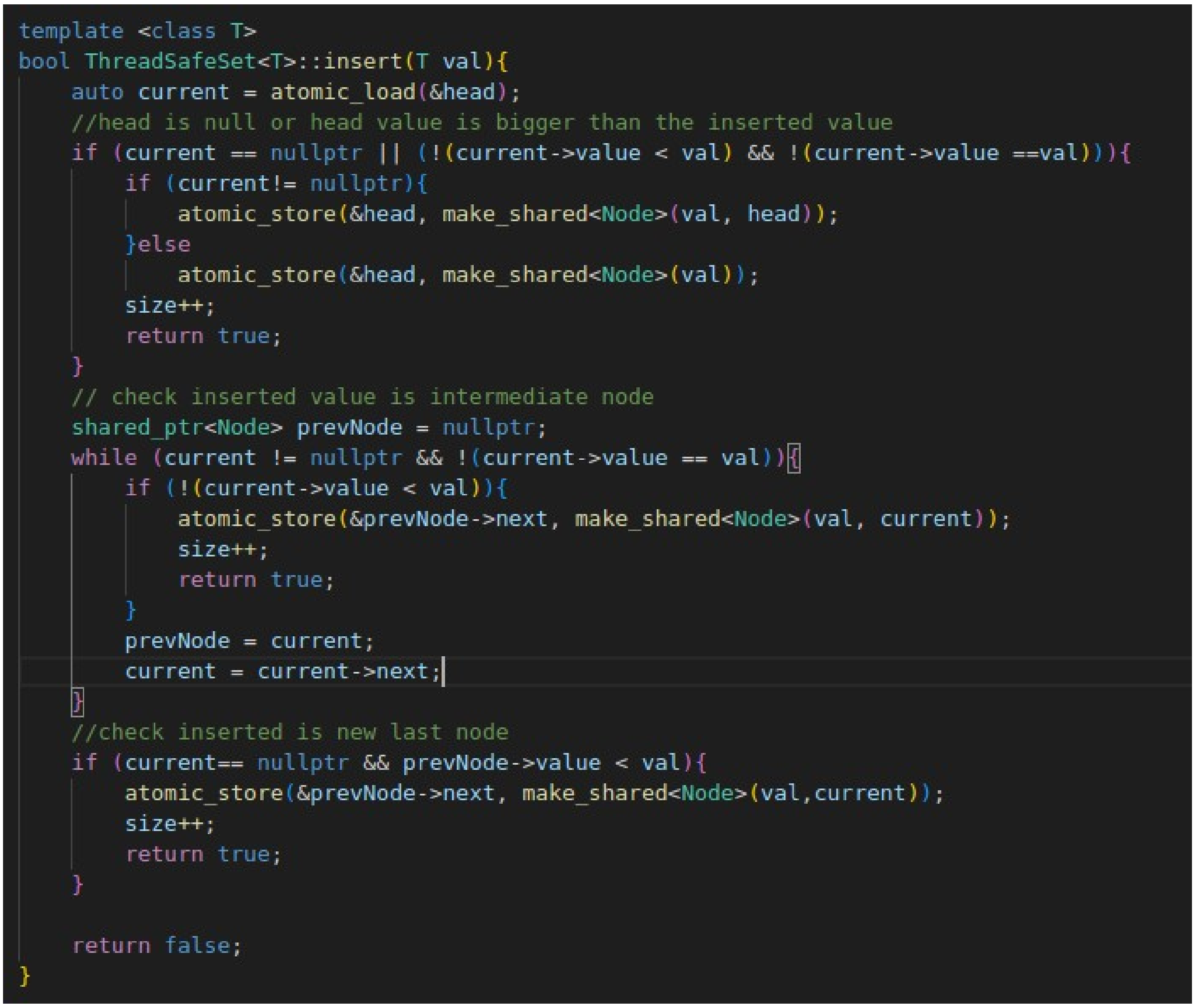
```cpp
template <class T>
bool ThreadSafeSet<T>::remove(T val){
    auto current = atomic_load(&head);
    if (current == nullptr)
        return false;
    //if head node is to be deleted
    if (current->value == val){
        atomic_store(&head, head->next);
        size--;
        return true;
    }
    //child node is to be deleted
    shared_ptr<Node> prevNode=nullptr;
    while (current!=nullptr){
        if (current->value == val){
            atomic_store(&prevNode->next, current->next);
            size--;
            return true;
        }
        if (!(current->value < val))
            break;
        prevNode=current;
        current = current->next;
    }
    return false;
}
```

void iterate([] lambda):



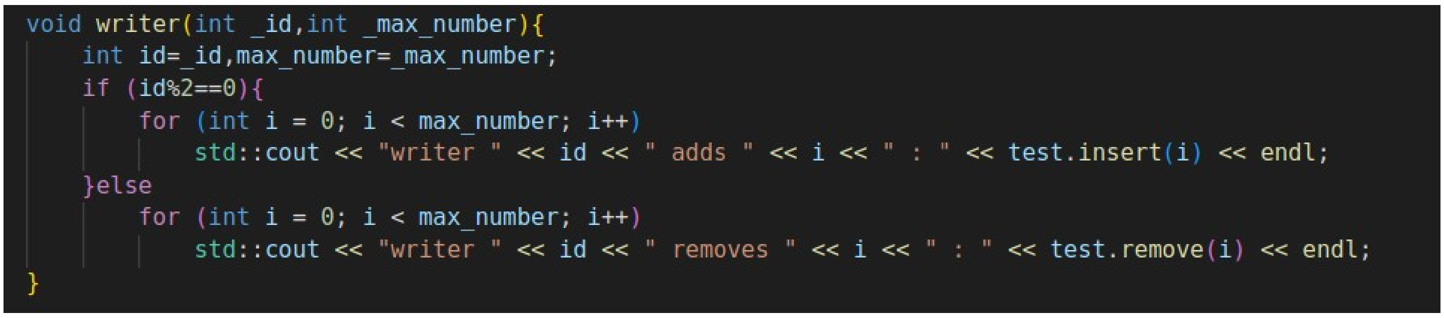
```cpp
template <class T>
void ThreadSafeSet<T>::iterate(function<void(const T&)> f){
    auto currnt = head;
    while(currnt != nullptr){
        f(currnt->value);
        currnt = currnt->next;
    }
}
```

bool insert(T val):



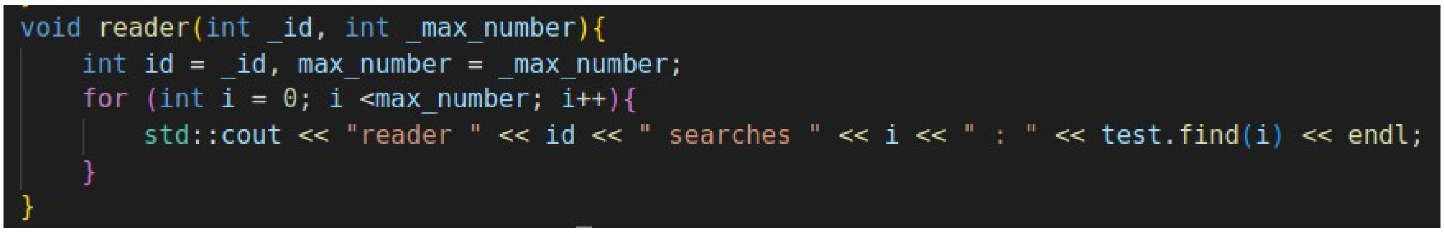
```cpp
template <class T>
bool ThreadSafeSet<T>::insert(T val){
    auto current = atomic_load(&head);
    //head is null or head value is bigger than the inserted value
    if (current == nullptr || (!(current->value < val) && !(current->value ==val))){
        if (current!= nullptr){
            atomic_store(&head, make_shared<Node>(val, head));
        }else
            atomic_store(&head, make_shared<Node>(val));
        size++;
        return true;
    }
    // check inserted value is intermediate node
    shared_ptr<Node> prevNode = nullptr;
    while (current != nullptr && !(current->value == val)){
        if (!(current->value < val)){
            atomic_store(&prevNode->next, make_shared<Node>(val, current));
            size++;
            return true;
        }
        prevNode = current;
        current = current->next;
    }
    //check inserted is new last node
    if (current== nullptr && prevNode->value < val){
        atomic_store(&prevNode->next, make_shared<Node>(val,current));
        size++;
        return true;
    }

    return false;
}
```

Writer:



```cpp
void writer(int _id,int _max_number){
    int id=_id,max_number=_max_number;
    if (id%2==0){
        for (int i = 0; i < max_number; i++)
            std::cout << "writer " << id << " adds " << i << " : " << test.insert(i) << endl;
    }else
        for (int i = 0; i < max_number; i++)
            std::cout << "writer " << id << " removes " << i << " : " << test.remove(i) << endl;
}
```

Reader:

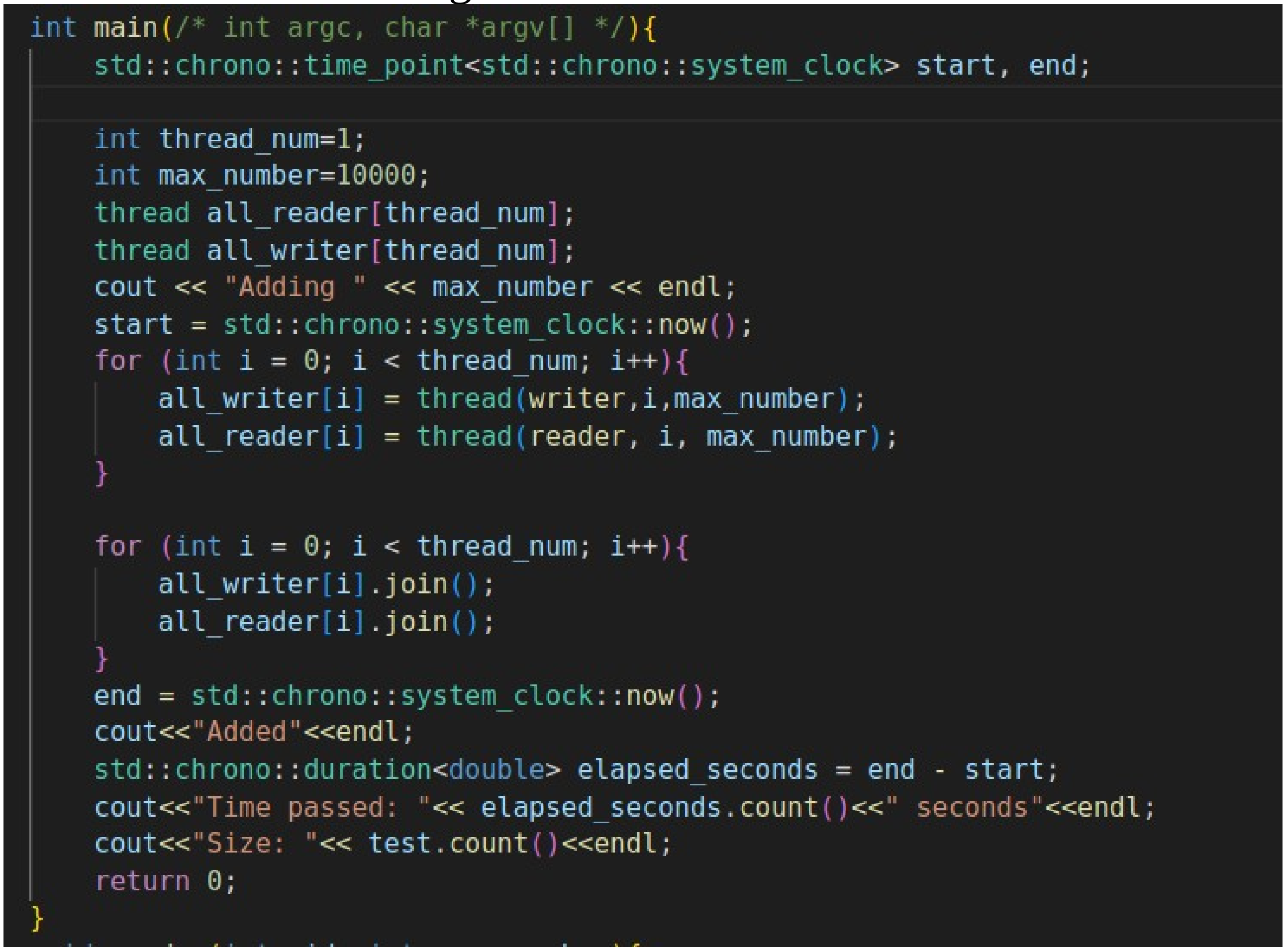


```cpp
void reader(int _id, int _max_number){
    int id = _id, max_number = _max_number;
    for (int i = 0; i <max_number; i++){
        std::cout << "reader " << id << " searches " << i << " : " << test.find(i) << endl;
    }
}
```

Tests:

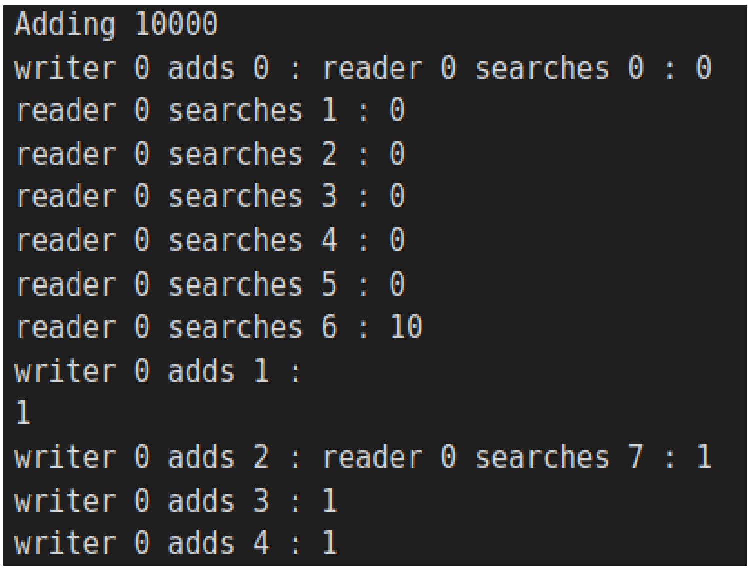
Case 1: 1 Reader – 1 Writer

Test 1: Adding 10 Thousand Numbers

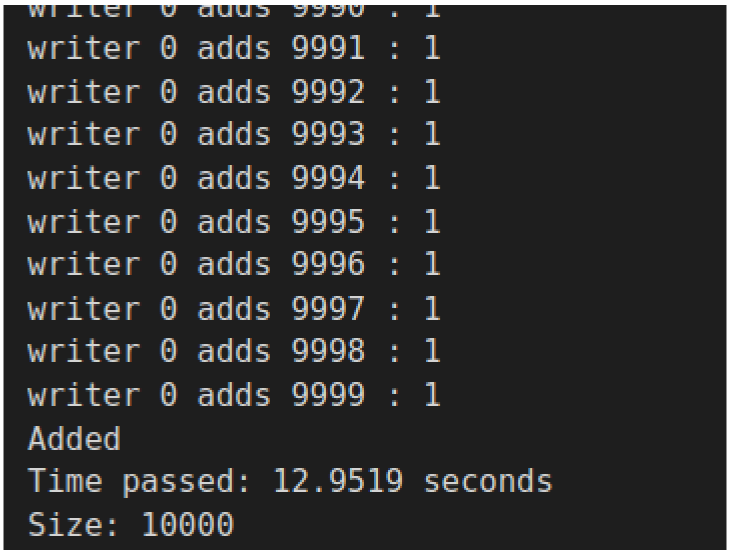


```cpp
int main(/* int argc, char *argv[] */){
    std::chrono::time_point<std::chrono::system_clock> start, end;

    int thread_num=1;
    int max_number=10000;
    thread all_reader[thread_num];
    thread all_writer[thread_num];
    cout << "Adding " << max_number << endl;
    start = std::chrono::system_clock::now();
    for (int i = 0; i < thread_num; i++){
        all_writer[i] = thread(writer,i,max_number);
        all_reader[i] = thread(reader, i, max_number);
    }

    for (int i = 0; i < thread_num; i++){
        all_writer[i].join();
        all_reader[i].join();
    }
    end = std::chrono::system_clock::now();
    cout<<"Added"<<endl;
    std::chrono::duration<double> elapsed_seconds = end - start;
    cout<<"Time passed: "<< elapsed_seconds.count()<<" seconds"<<endl;
    cout<<"Size: "<< test.count()<<endl;
    return 0;
}
```

Test 1 Results:



```
Adding 10000
writer 0 adds 0 : reader 0 searches 0 : 0
reader 0 searches 1 : 0
reader 0 searches 2 : 0
reader 0 searches 3 : 0
reader 0 searches 4 : 0
reader 0 searches 5 : 0
reader 0 searches 6 : 10
writer 0 adds 1 :
1
writer 0 adds 2 : reader 0 searches 7 : 1
writer 0 adds 3 : 1
writer 0 adds 4 : 1
```



```
writer 0 adds 9990 : 1
writer 0 adds 9991 : 1
writer 0 adds 9992 : 1
writer 0 adds 9993 : 1
writer 0 adds 9994 : 1
writer 0 adds 9995 : 1
writer 0 adds 9996 : 1
writer 0 adds 9997 : 1
writer 0 adds 9998 : 1
writer 0 adds 9999 : 1
Added
Time passed: 12.9519 seconds
Size: 10000
```

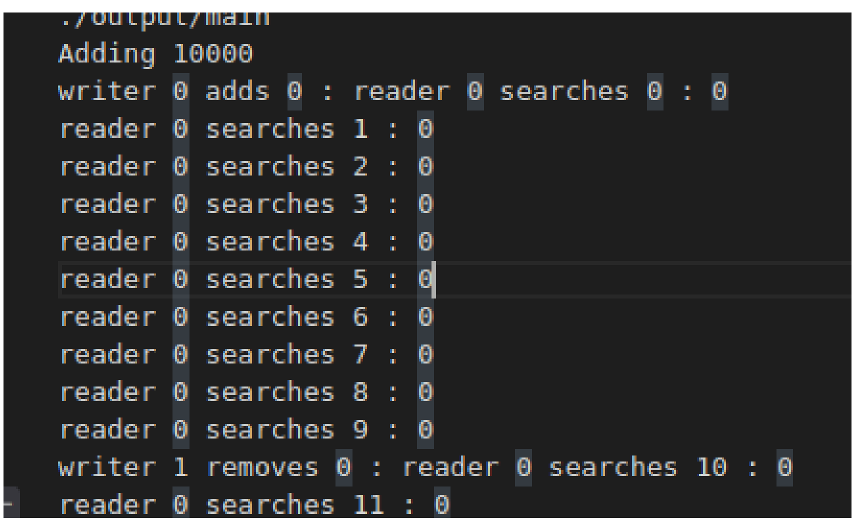
*cout is not prints atomically so my outputs are conflicting.

Case 2 : Multiple Reader- Multiple Writer

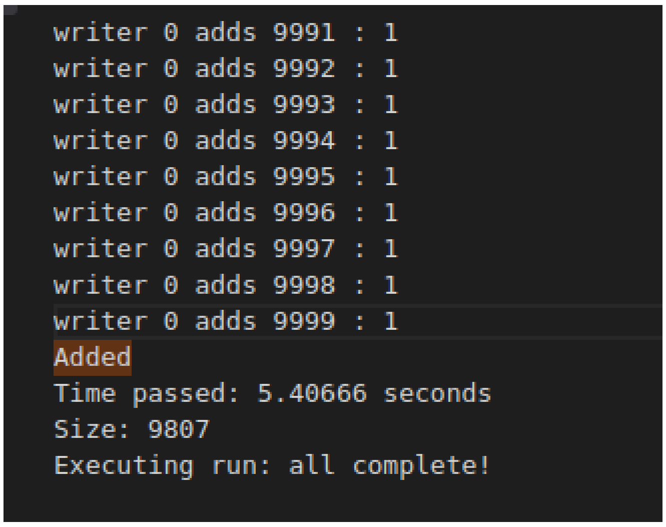
Test 2: 2 Reader – 2 Writer ; Adding 10 Thousand Numbers

My all readers just check if the number exists.

Even numbered writers add numbers and odd numbered writers remove numbers.

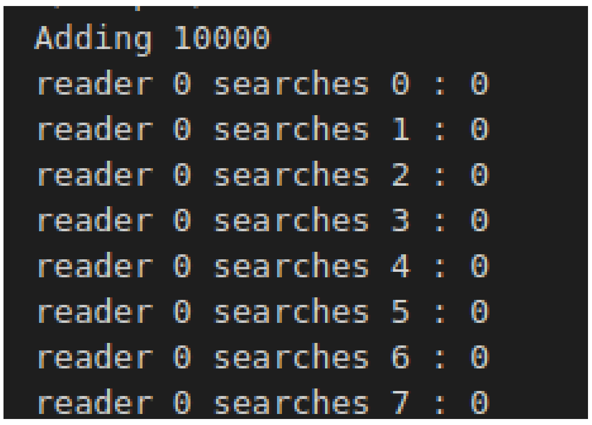


```
./output/main
Adding 10000
writer 0 adds 0 : reader 0 searches 0 : 0
reader 0 searches 1 : 0
reader 0 searches 2 : 0
reader 0 searches 3 : 0
reader 0 searches 4 : 0
reader 0 searches 5 : 0
reader 0 searches 6 : 0
reader 0 searches 7 : 0
reader 0 searches 8 : 0
reader 0 searches 9 : 0
writer 1 removes 0 : reader 0 searches 10 : 0
reader 0 searches 11 : 0
```

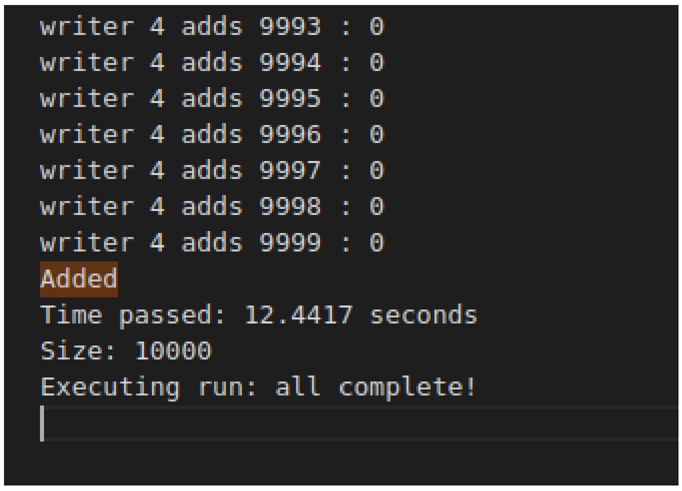


```
writer 0 adds 9991 : 1
writer 0 adds 9992 : 1
writer 0 adds 9993 : 1
writer 0 adds 9994 : 1
writer 0 adds 9995 : 1
writer 0 adds 9996 : 1
writer 0 adds 9997 : 1
writer 0 adds 9998 : 1
writer 0 adds 9999 : 1
Added
Time passed: 5.40666 seconds
Size: 9807
Executing run: all complete!
```

Test 3: 5 Reader – 5 Writer ; Adding 10 Thousand Numbers

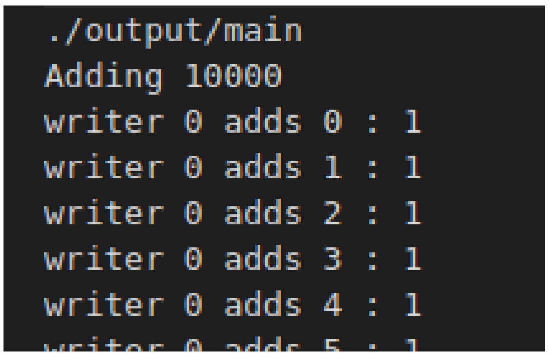


```
Adding 10000
reader 0 searches 0 : 0
reader 0 searches 1 : 0
reader 0 searches 2 : 0
reader 0 searches 3 : 0
reader 0 searches 4 : 0
reader 0 searches 5 : 0
reader 0 searches 6 : 0
reader 0 searches 7 : 0
```

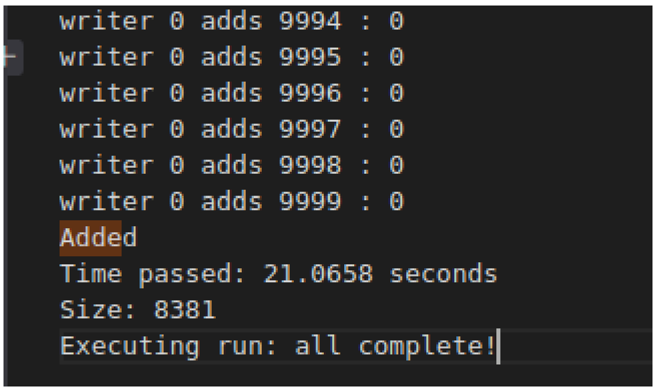


```
writer 4 adds 9993 : 0
writer 4 adds 9994 : 0
writer 4 adds 9995 : 0
writer 4 adds 9996 : 0
writer 4 adds 9997 : 0
writer 4 adds 9998 : 0
writer 4 adds 9999 : 0
Added
Time passed: 12.4417 seconds
Size: 10000
Executing run: all complete!
```

Test 4: 10 Reader – 10 Writer ; Adding 10 Thousand Numbers



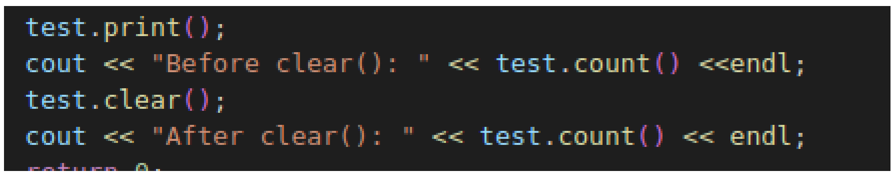
```
./output/main
Adding 10000
writer 0 adds 0 : 1
writer 0 adds 1 : 1
writer 0 adds 2 : 1
writer 0 adds 3 : 1
writer 0 adds 4 : 1
```



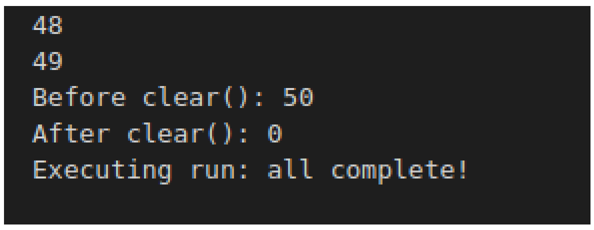
```
writer 0 adds 9994 : 0
writer 0 adds 9995 : 0
writer 0 adds 9996 : 0
writer 0 adds 9997 : 0
writer 0 adds 9998 : 0
writer 0 adds 9999 : 0
Added
Time passed: 21.0658 seconds
Size: 8381
Executing run: all complete!
```

Function Tests:

clear():

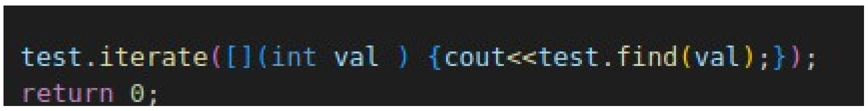


```
test.print();
cout << "Before clear(): " << test.count() <<endl;
test.clear();
cout << "After clear(): " << test.count() << endl;
```

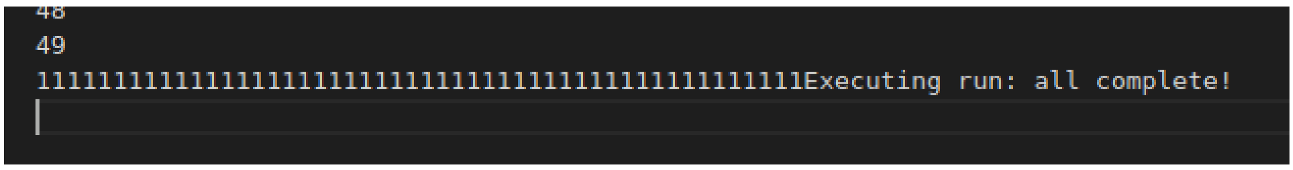


```
48
49
Before clear(): 50
After clear(): 0
Executing run: all complete!
```

iterate([] lambda):

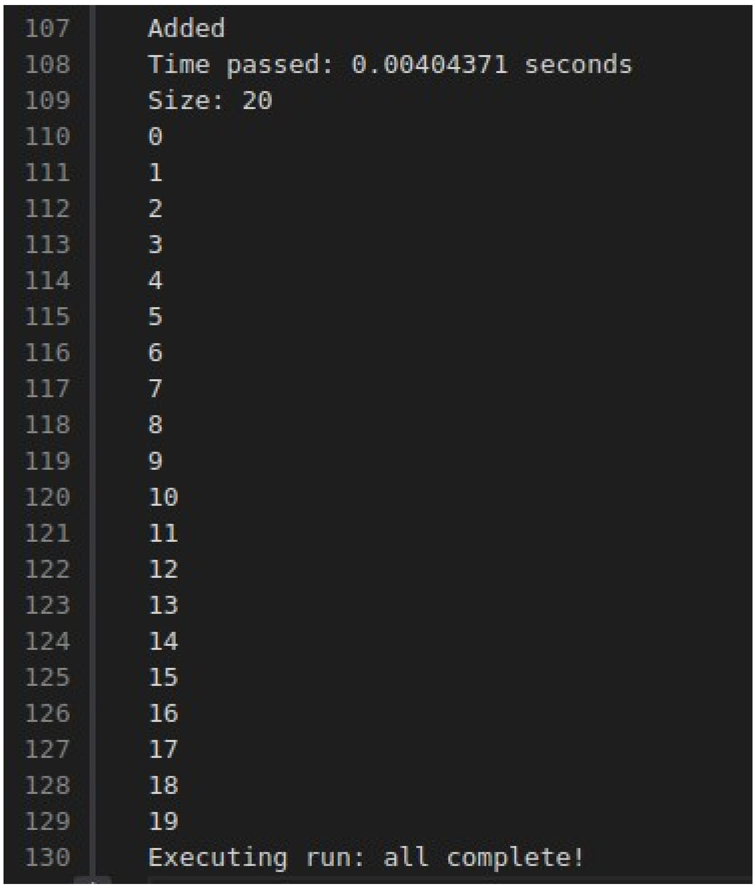


```
test.iterate([](int val ) {cout<<test.find(val);});
return 0;
```

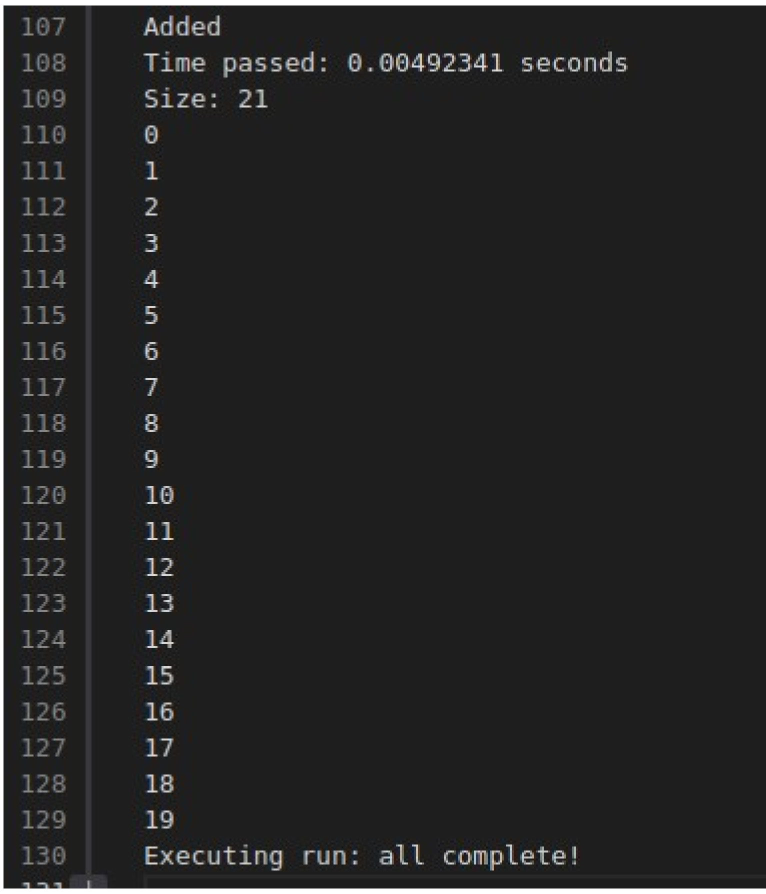


```
48
49
11111111111111111111111111111111111111111111111111Executing run: all complete!
```

Note: Simultaneous addition works correctly, but sometimes the counter is more than it should be.



```
Added
Time passed: 0.00404371 seconds
Size: 20
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
Executing run: all complete!
```



```
Added
Time passed: 0.00492341 seconds
Size: 21
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
Executing run: all complete!
```

As you can see, outputs are correct but size is wrong.