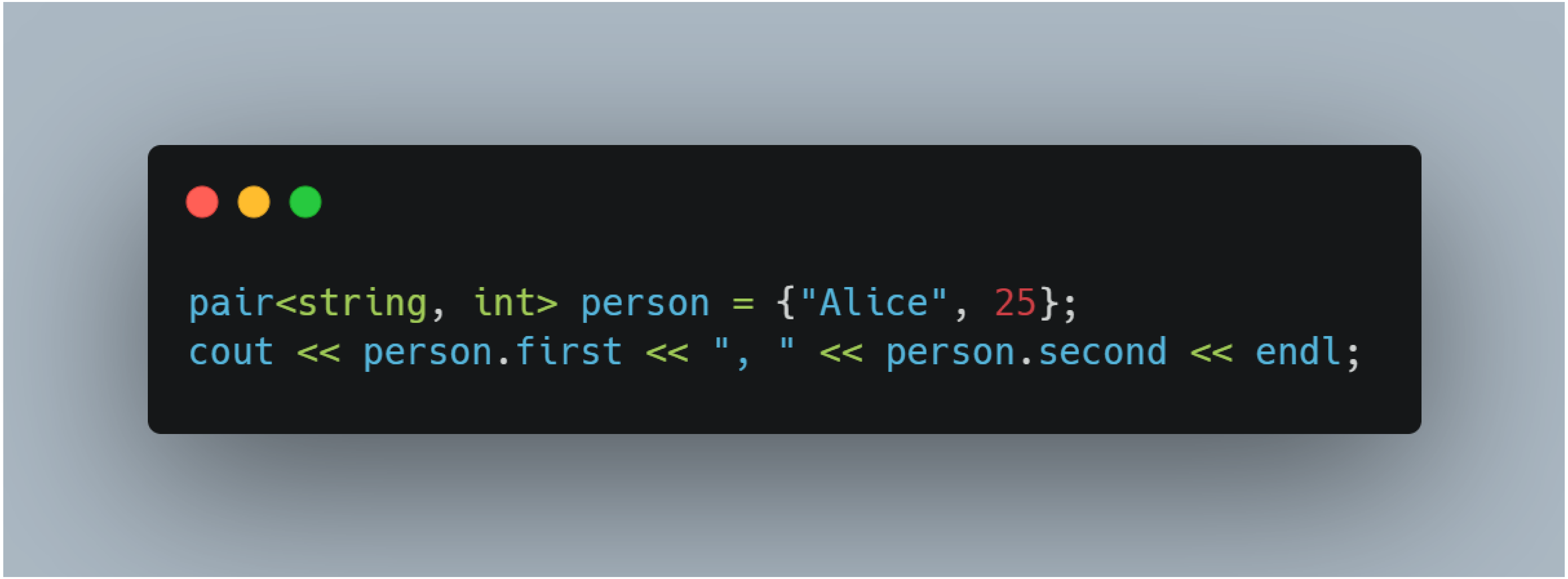
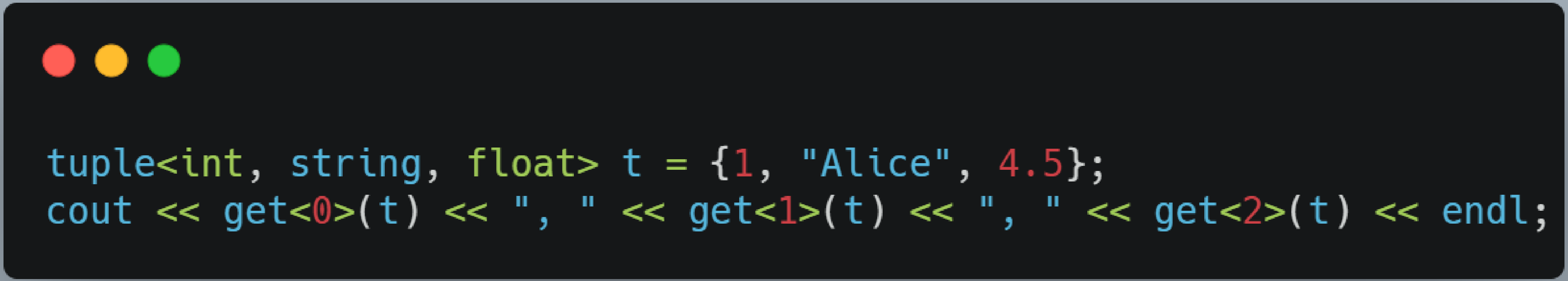
C ++ STL
Standard Template Library

pair<T1, T2>



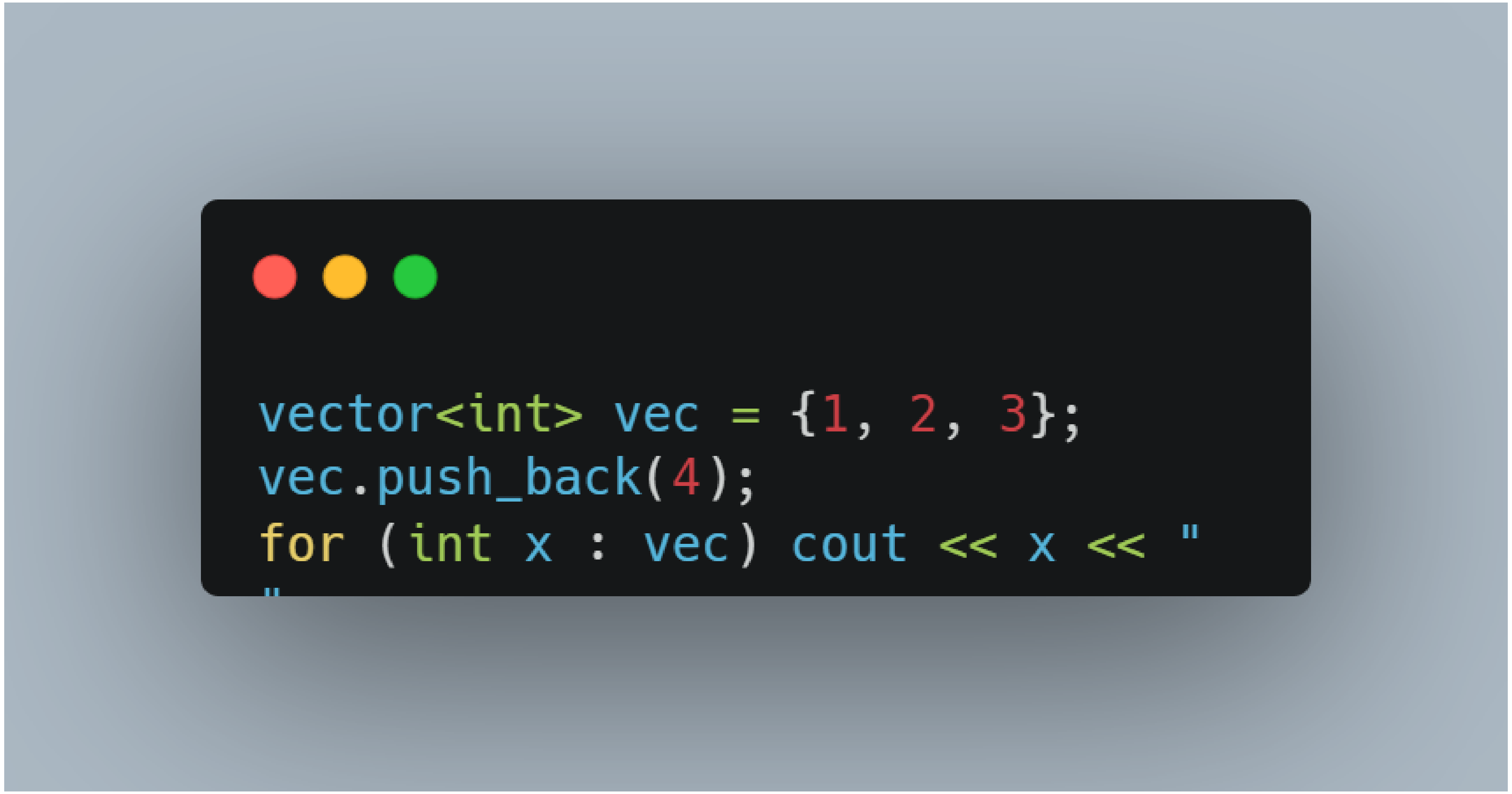
```
pair<string, int> person = {"Alice", 25};
cout << person.first << ", " << person.second << endl;
```

tuple<T1, T2, ...>



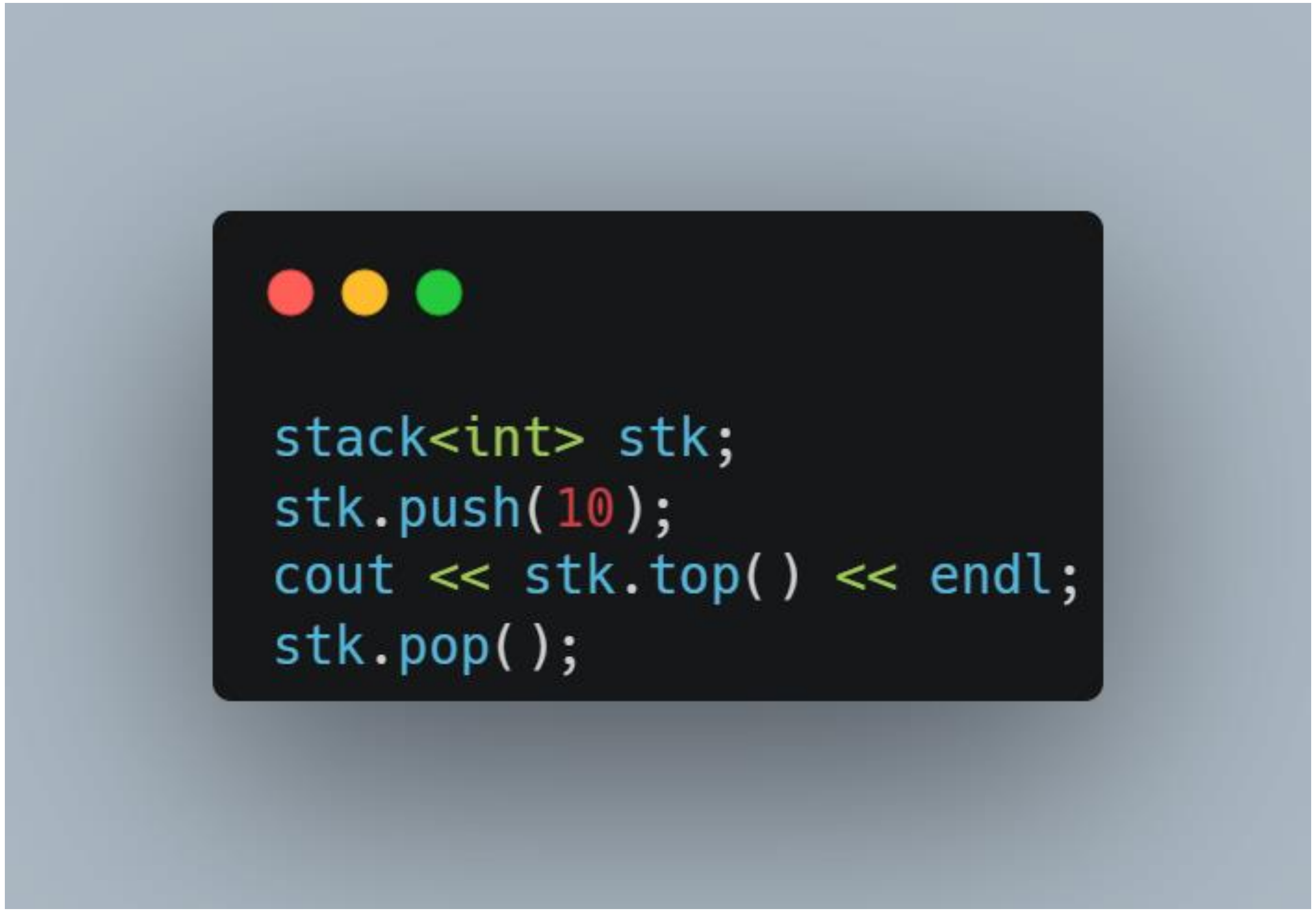
```cpp
tuple<int, string, float> t = {1, "Alice", 4.5};
cout << get<0>(t) << ", " << get<1>(t) << ", " << get<2>(t) << endl;
```

vector<T1, T2, ...>



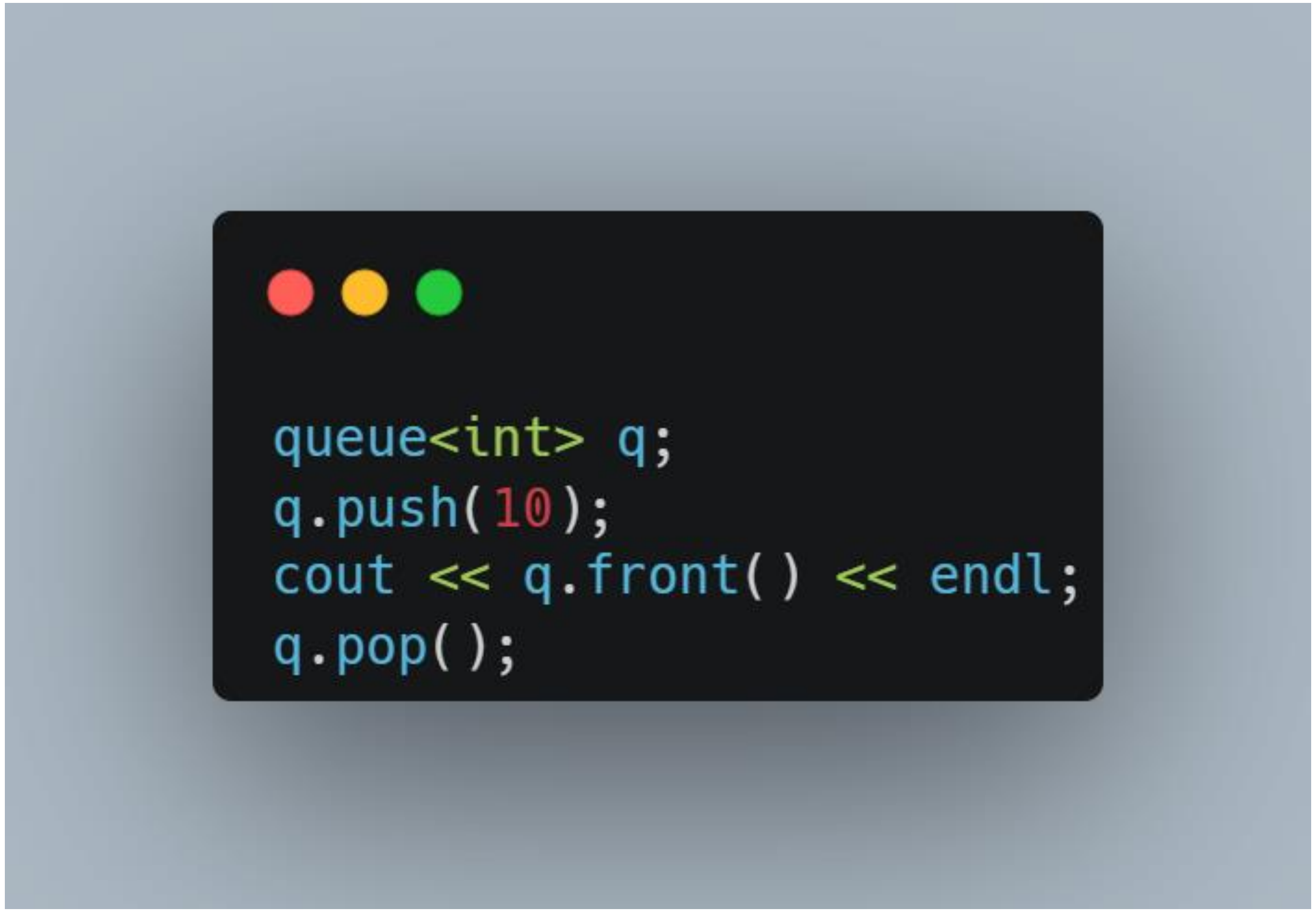
```
vector<int> vec = {1, 2, 3};
vec.push_back(4);
for (int x : vec) cout << x << "
"
```

stack<T1>



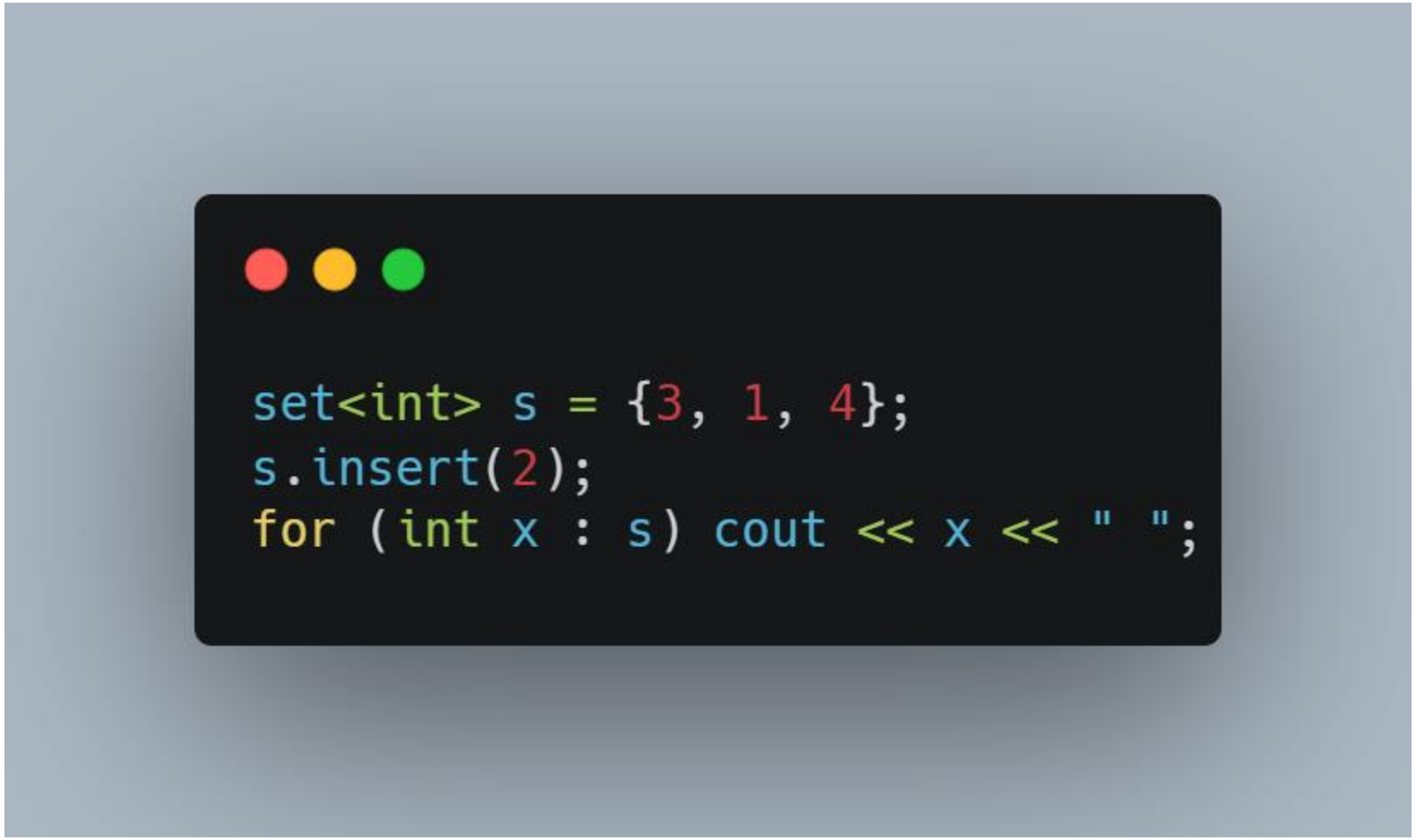
```
stack<int> stk;
stk.push(10);
cout << stk.top() << endl;
stk.pop();
```

queue<T1>



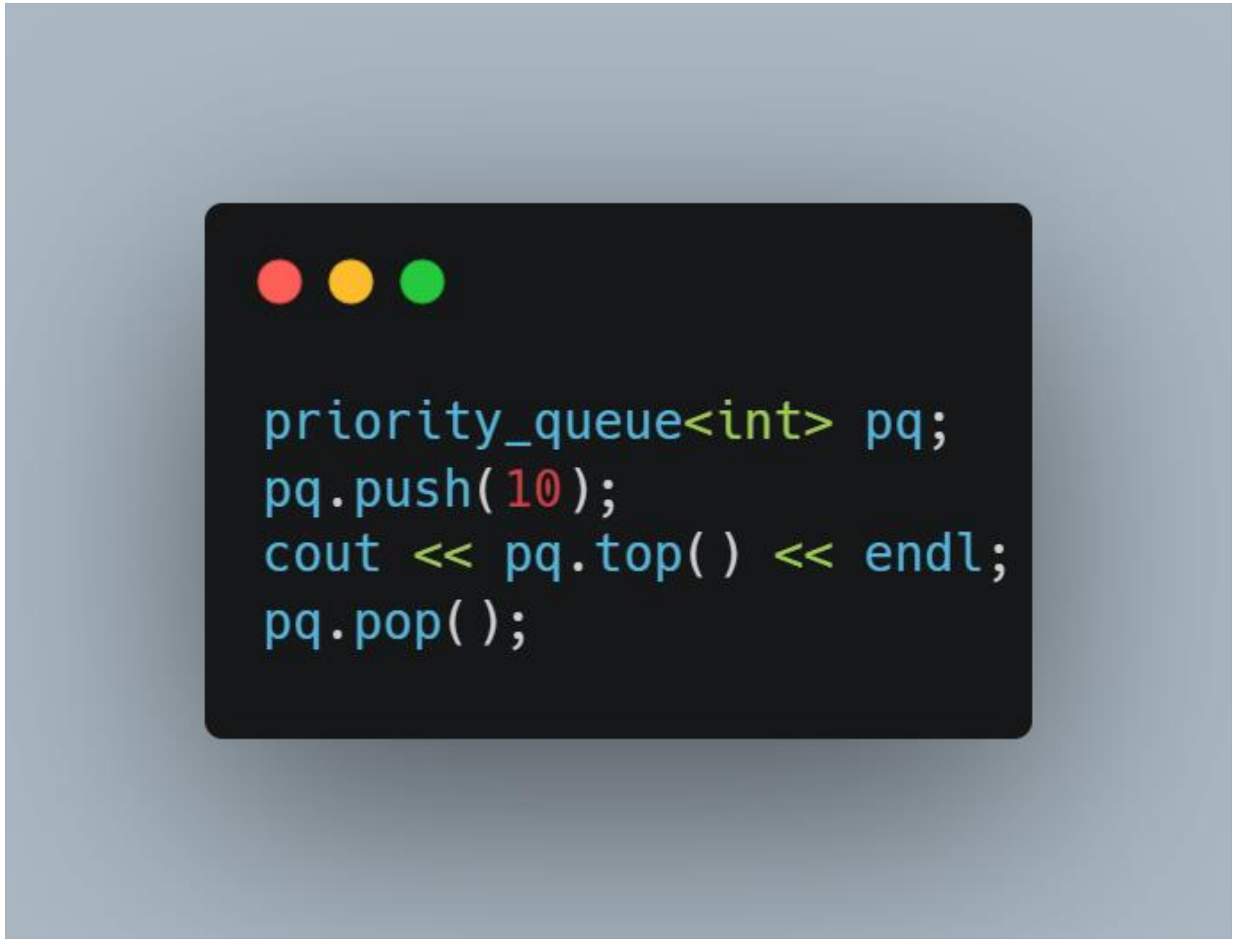
```
queue<int> q;
q.push(10);
cout << q.front() << endl;
q.pop();
```

set<T1>



```
set<int> s = {3, 1, 4};
s.insert(2);
for (int x : s) cout << x << " ";
```

priority_queue<T1>

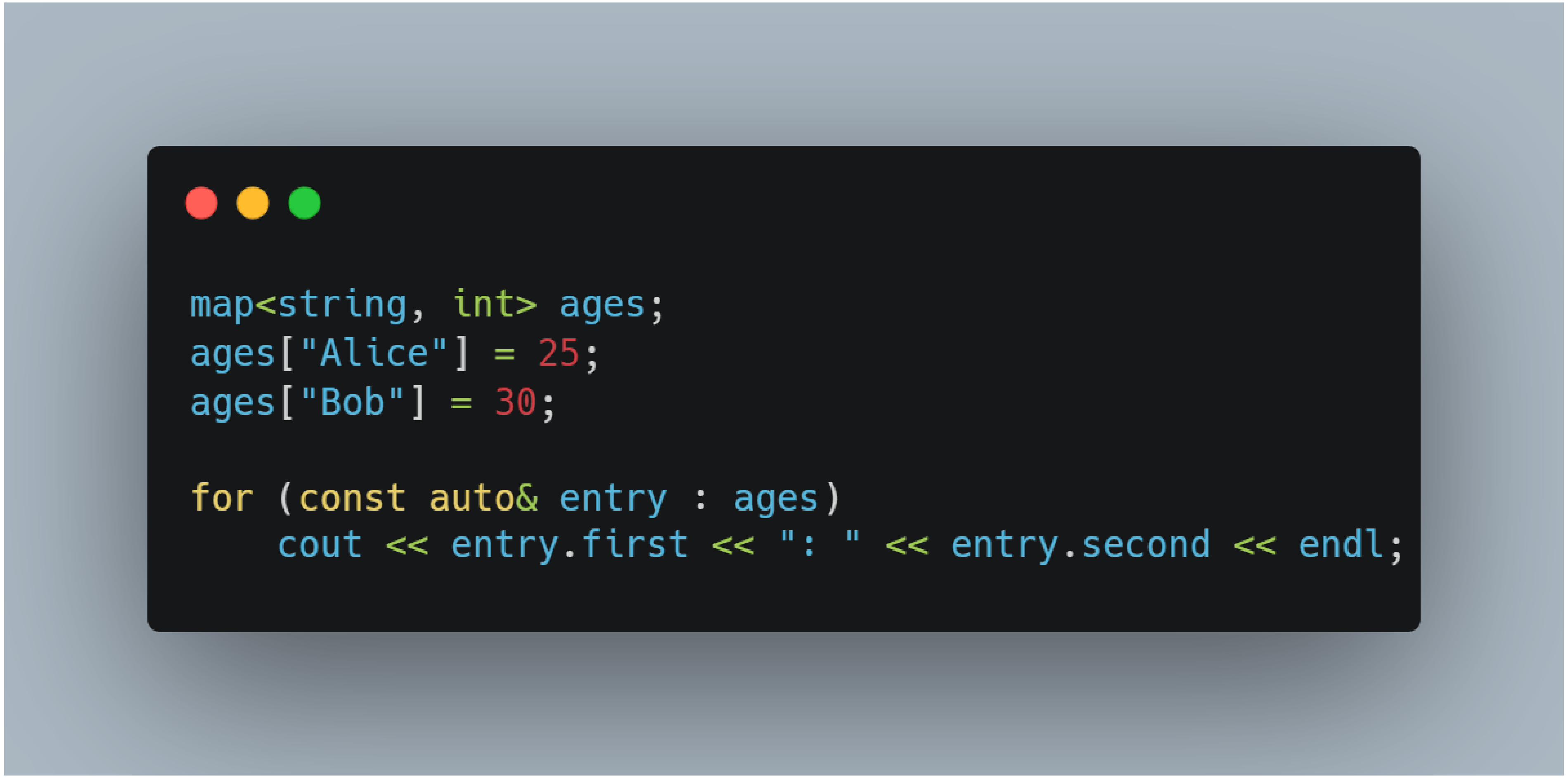


```
priority_queue<int> pq;
pq.push(10);
cout << pq.top() << endl;
pq.pop();
```

set vs priority_queue

Feature	std::set	std::priority_queue
Ordering	Sorted (ascending by default)	Not sorted; based on priority
Underlying Structure	Balanced tree (e.g., Red-Black Tree)	Heap (binary heap)
Duplicates	No duplicates	Duplicates allowed
Access	Iterate through sorted elements	Access only top element (highest/lowest priority)
Insertion Complexity	O(log n)	O(log n)
Removal Complexity	O(log n)	O(log n) (pop top element)
Use Case	Unique elements in sorted order	Efficient top-priority element access (e.g., scheduling)

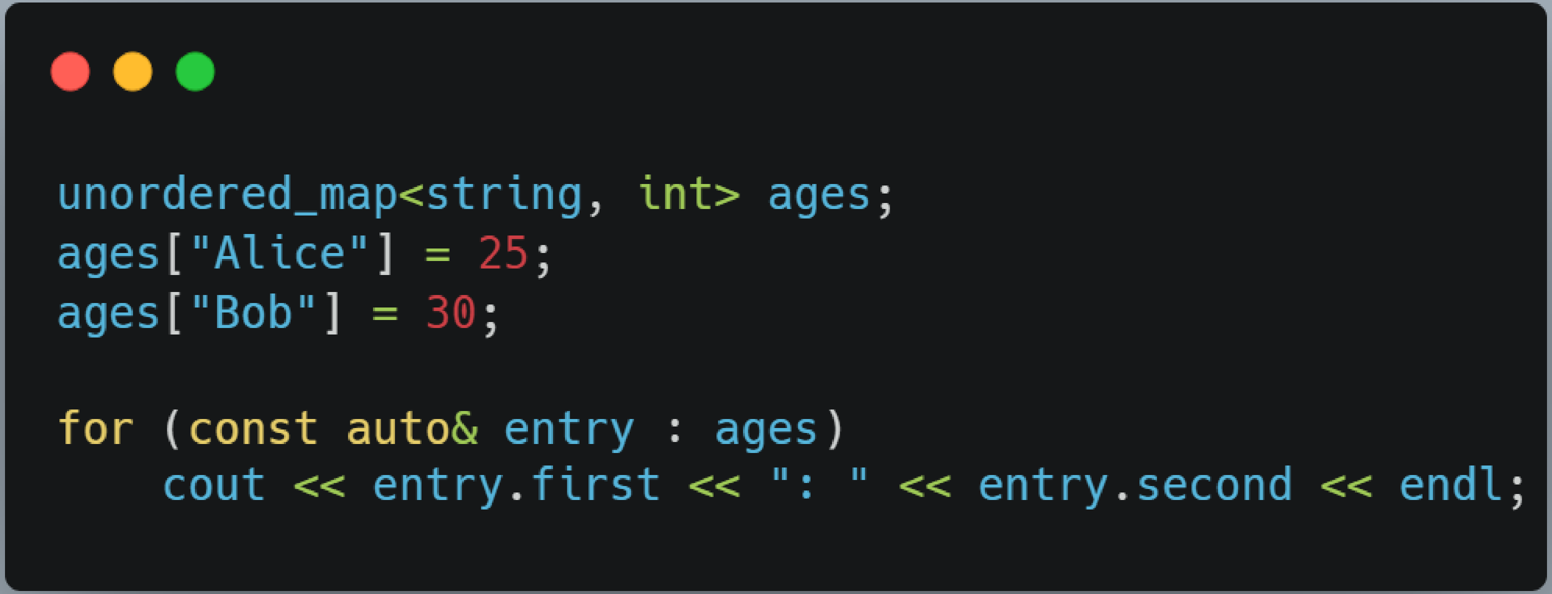
map<key, value>



```cpp
map<string, int> ages;
ages["Alice"] = 25;
ages["Bob"] = 30;

for (const auto& entry : ages)
    cout << entry.first << ": " << entry.second << endl;
```

unordered_map<key, value>



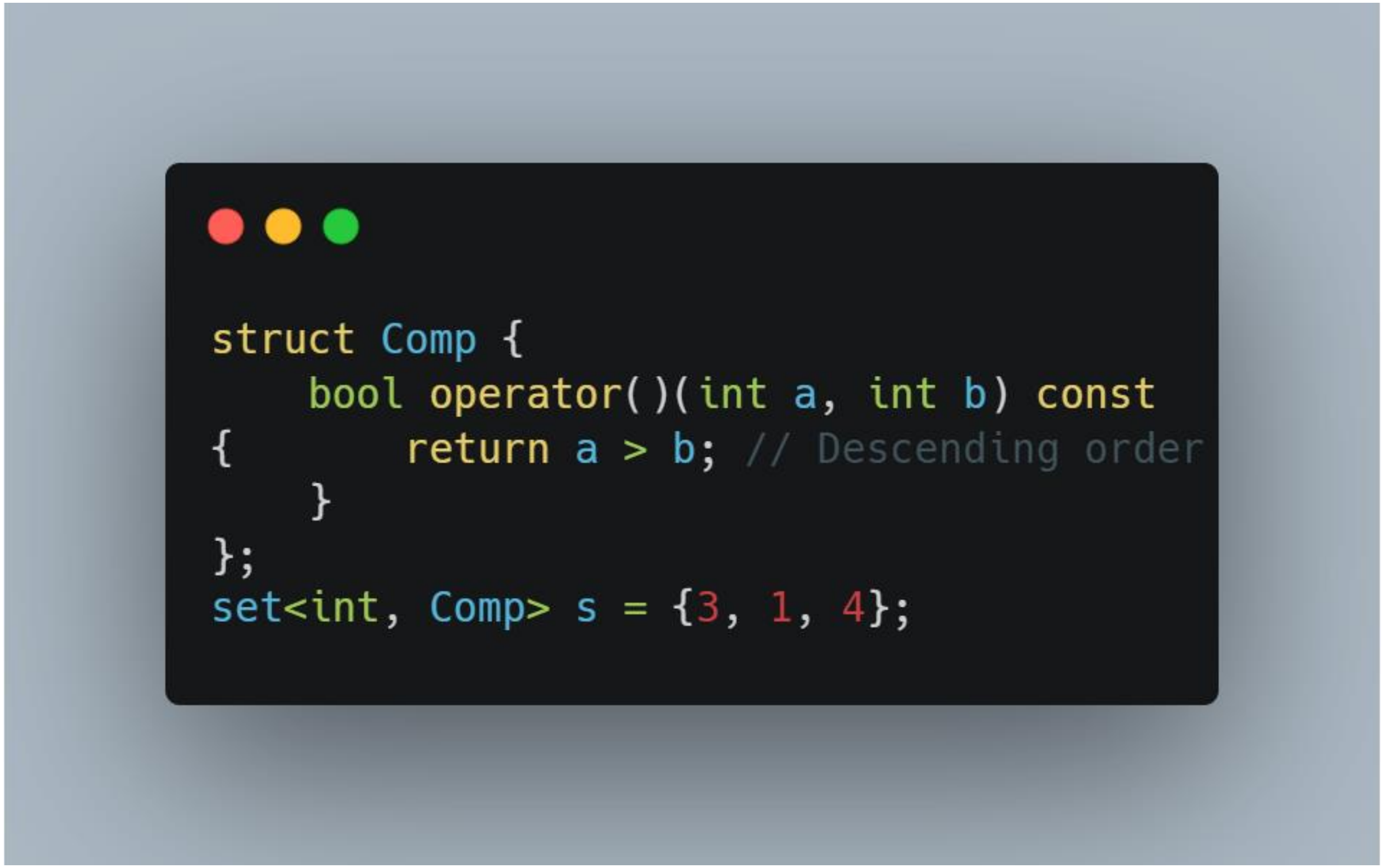
```
unordered_map<string, int> ages;
ages["Alice"] = 25;
ages["Bob"] = 30;

for (const auto& entry : ages)
    cout << entry.first << ": " << entry.second << endl;
```

map vs unordered_map

Feature	**map**	**unordered_map**
Ordering	Sorted by key	No specific order
Time Complexity	O(log n)	O(1) on average
Underlying Structure	Balanced Tree (e.g., Red-Black Tree)	Hash Table
Duplicate Keys	Not allowed	Not allowed
Insertion Order	Not preserved	Not preserved
Use Case	When sorted order is needed	When performance is critical

Custom Comparator



```
struct Comp {
    bool operator()(int a, int b) const
{       return a > b; // Descending order
    }
};
set<int, Comp> s = {3, 1, 4};
```

Resources

1. https://cplusplus.com/reference/vector/vector/
2. https://cplusplus.com/reference/utility/pair/
3. https://cplusplus.com/reference/tuple/tuple/
4. https://cplusplus.com/reference/stack/stack/
5. https://cplusplus.com/reference/queue/queue/
6. https://cplusplus.com/reference/set/set/
7. https://cplusplus.com/reference/unordered_map/unordered_map/
8. https://cplusplus.com/reference/queue/priority_queue/
9. https://en.cppreference.com/w/cpp/container

Thank You