C++ STL ve Vector Referans Kılavuzu

Programlama Referansı

13 Temmuz 2025

İçindekiler

1 Giriş **2**

2 Vector Container **2**
2.1 Temel Tanımlama ve Başlatma 2
2.2 Temel Fonksiyonlar 2

3 STL Algoritmaları **3**
3.1 Sıralama Algoritmaları 3
3.2 Arama Algoritmaları 3

4 Set Container **3**
4.1 Set Temel Kullanımı 3
4.2 Multiset ve Unordered Set 4

5 Map Container **4**
5.1 Map Temel Kullanımı 4

6 Lambda Fonksiyonlar **5**
6.1 Temel Lambda Kullanımı 5

7 Struct + Vector Kombinasyonu **6**
7.1 Struct Tanımlama ve Kullanımı 6

8 Yararlı STL Fonksiyonları **6**
8.1 Numeric Algoritmalar 6
8.2 Diğer Yararlı Fonksiyonlar 7

9 Performans İpuçları **8**

10 Sık Karşılaşılan Hatalar **8**

11 Özet Referans Tablosu **8**

1 Giriş

Bu kılavuz, C++ Standard Template Library (STL) ve özellikle vector container'ının en sık kullanılan fonksiyonlarını ve özelliklerini kapsamaktadır. Programlama yarışmaları ve genel C++ geliştirme için pratik bir referans olarak hazırlanmıştır.

2 Vector Container

2.1 Temel Tanımlama ve Başlatma

```cpp
#include <vector>
#include <iostream>
using namespace std;

// Bo  vector
vector<int> v1;

// Belirli boyutta vector (varsay lan de er 0)
vector<int> v2(5);

// Belirli boyut ve de er ile
vector<int> v3(5, 10);  // 5 elemanl , hepsi 10

// Ba lang   listesi ile
vector<int> v4 = {1, 2, 3, 4, 5};

// Ba ka bir vector'dan kopyalama
vector<int> v5(v4);

// 2D vector
vector<vector<int>> matrix(3, vector<int>(4, 0));  // 3x4 matrix
```

Listing 1: Vector Tanımlama Örnekleri

2.2 Temel Fonksiyonlar

```cpp
vector<int> v = {1, 2, 3, 4, 5};

// Eleman ekleme
v.push_back(6);           // Sona ekle
v.insert(v.begin() + 2, 10);  // 2. pozisyona ekle

// Eleman silme
v.pop_back();             // Sondan sil
v.erase(v.begin() + 1);   // 1. pozisyondaki eleman  sil
v.erase(v.begin() + 1, v.begin() + 3);  // Aral k silme

// Eri im
cout << v[0];             //  ndeks  ile eri im
cout << v.at(1);          // G venli eri im
cout << v.front();        //  lk  eleman
cout << v.back();         // Son eleman

// Boyut kontrolleri
cout << v.size();         // Eleman say s
cout << v.empty();        // Bo  mu?
v.clear();               // T m elemanlar  sil

// Boyut de i tirme
v.resize(10);            // Boyutu 10 yap
v.resize(15, 5);         // Boyutu 15 yap, yeni elemanlar 5
```

Listing 2: Vector Temel İşlemleri

3 STL Algoritmaları

3.1 Sıralama Algoritmaları

```cpp
#include <algorithm>
#include <vector>
using namespace std;

vector<int> v = {5, 2, 8, 1, 9};

// K    kten b y   e s ralama
sort(v.begin(), v.end());

// B y kten k       e s ralama
sort(v.begin(), v.end(), greater<int>());

// Custom comparator ile s ralama
sort(v.begin(), v.end(), [](int a, int b) {
    return a > b;  // B y kten k       e
});

// K smi s ralama
partial_sort(v.begin(), v.begin() + 3, v.end());

// Stable sort (e it elemanlar n s ras n  korur)
stable_sort(v.begin(), v.end());
```

Listing 3: Sıralama Örnekleri

3.2 Arama Algoritmaları

```cpp
vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9};

// Linear search
auto it = find(v.begin(), v.end(), 5);
if (it != v.end()) {
    cout << "Bulundu: " << *it << endl;
}

// Binary search (s ral  vector i in)
bool found = binary_search(v.begin(), v.end(), 5);

// Lower bound (>= de er)
auto lb = lower_bound(v.begin(), v.end(), 5);
cout << "Lower bound: " << *lb << endl;

// Upper bound (> de er)
auto ub = upper_bound(v.begin(), v.end(), 5);
cout << "Upper bound: " << *ub << endl;

// Equal range
auto range = equal_range(v.begin(), v.end(), 5);
cout << "Equal range: " << *range.first << " - " << *range.second << endl;
```

Listing 4: Arama Fonksiyonları

4 Set Container

4.1 Set Temel Kullanımı

```cpp
#include <set>
#include <iostream>
```

```
using namespace std;

// Set tan mlama
set<int> s;
set<int> s2 = {3, 1, 4, 1, 5};  // Tekrar eden elemanlar otomatik silinir

// Eleman ekleme
s.insert(10);
s.insert(20);
s.insert(15);

// Eleman silme
s.erase(10);                     // De er ile silme
s.erase(s.find(20));            // Iterator ile silme

// Arama
auto it = s.find(15);
if (it != s.end()) {
    cout << "Bulundu: " << *it << endl;
}

// Boyut ve bo  kontrol
cout << s.size() << endl;
cout << s.empty() << endl;

// Lower/Upper bound
auto lb = s.lower_bound(15);
auto ub = s.upper_bound(15);
```

Listing 5: Set Kullanımı

4.2 Multiset ve Unordered Set

```
#include <set>
#include <unordered_set>
using namespace std;

// Multiset (tekrar eden elemanlar)
multiset<int> ms = {1, 2, 2, 3, 3, 3};
ms.insert(2);
cout << ms.count(2) << endl;  // 2'nin ka  tane oldu u

// Unordered set (hash tabanl )
unordered_set<int> us = {1, 2, 3, 4, 5};
us.insert(6);
auto it = us.find(3);  // O(1) ortalama
```

Listing 6: Multiset ve Unordered Set

5 Map Container

5.1 Map Temel Kullanımı

```
#include <map>
#include <string>
#include <iostream>
using namespace std;

// Map tan mlama
map<string, int> m;
map<string, int> m2 = {{"ali", 25}, {"veli", 30}};

```

```cpp
// Eleman ekleme
m["ahmet"] = 20;
m.insert({"mehmet", 35});
m.insert(make_pair("fatma", 28));

// Eleman eri imi
cout << m["ahmet"] << endl;       // 20
cout << m.at("mehmet") << endl;  // 35 (g venli eri im)

// Eleman silme
m.erase("ali");
m.erase(m.find("veli"));

// Arama
auto it = m.find("ahmet");
if (it != m.end()) {
    cout << it->first << ": " << it->second << endl;
}

// Iterasyon
for (auto& pair : m) {
    cout << pair.first << ": " << pair.second << endl;
}

// Boyut kontrolleri
cout << m.size() << endl;
cout << m.empty() << endl;
```

Listing 7: Map Kullanımı

6 Lambda Fonksiyonlar

6.1 Temel Lambda Kullanımı

```cpp
#include <vector>
#include <algorithm>
using namespace std;

vector<int> v = {5, 2, 8, 1, 9};

// Basit lambda
auto print = [](int x) { cout << x << " "; };
for_each(v.begin(), v.end(), print);

// Capture ile lambda
int threshold = 5;
auto count_greater = [threshold](const vector<int>& vec) {
    return count_if(vec.begin(), vec.end(),
                    [threshold](int x) { return x > threshold; });
};

// Custom sort lambda
sort(v.begin(), v.end(), [](int a, int b) {
    return a > b;  // B y kten k       e
});

// Mutable lambda
int counter = 0;
auto increment = [counter]() mutable { return ++counter; };
```

Listing 8: Lambda Fonksiyonlar

7 Struct + Vector Kombinasyonu

7.1 Struct Tanımlama ve Kullanımı

```cpp
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

struct Student {
    string name;
    int age;
    double grade;

    // Constructor
    Student(string n, int a, double g) : name(n), age(a), grade(g) {}

    // Kar  la t rma operat r
    bool operator<(const Student& other) const {
        return grade > other.grade;  // Nota g re azalan
    }
};

int main() {
    vector<Student> students;

    //   renci   ekleme
    students.push_back(Student("Ali", 20, 85.5));
    students.emplace_back("Veli", 21, 92.0);
    students.emplace_back("Ay e", 19, 78.5);

    // Nota g re s ralama (struct'taki < operat r )
    sort(students.begin(), students.end());

    // Custom lambda ile s ralama
    sort(students.begin(), students.end(),
         [](const Student& a, const Student& b) {
             return a.name < b.name;  //  sme  g re alfabetik
         });

    // Ya a g re s ralama
    sort(students.begin(), students.end(),
         [](const Student& a, const Student& b) {
             return a.age < b.age;
         });

    // Belirli ko ulu sa layan   rencileri   bulma
    auto high_grade = find_if(students.begin(), students.end(),
                              [](const Student& s) {
                                  return s.grade >= 90.0;
                              });

    return 0;
}
```

Listing 9: Struct ile Vector Kullanımı

8 Yararlı STL Fonksiyonları

8.1 Numeric Algoritmalar

```cpp
#include <numeric>
#include <vector>
```

```
using namespace std;

vector<int> v = {1, 2, 3, 4, 5};

// Toplam
int sum = accumulate(v.begin(), v.end(), 0);

//    arpm
int product = accumulate(v.begin(), v.end(), 1, multiplies<int>());

// K smi toplam
vector<int> partial_sums(v.size());
partial_sum(v.begin(), v.end(), partial_sums.begin());

//         arpm
vector<int> v2 = {1, 1, 1, 1, 1};
int dot_product = inner_product(v.begin(), v.end(), v2.begin(), 0);
```

Listing 10: Sayısal Algoritmalar

8.2 Diğer Yararlı Fonksiyonlar

```
#include <algorithm>
#include <vector>
using namespace std;

vector<int> v = {1, 2, 3, 4, 5};

// Min/Max elemanlar
auto min_it = min_element(v.begin(), v.end());
auto max_it = max_element(v.begin(), v.end());
auto minmax_pair = minmax_element(v.begin(), v.end());

// Ters   evirme
reverse(v.begin(), v.end());

// D nd rme
rotate(v.begin(), v.begin() + 2, v.end());

// Benzersiz elemanlar
sort(v.begin(), v.end());
v.erase(unique(v.begin(), v.end()), v.end());

// Rastgele kar   t rma
random_shuffle(v.begin(), v.end());

// Sonraki perm tasyon
next_permutation(v.begin(), v.end());

//   nceki   perm tasyon
prev_permutation(v.begin(), v.end());
```

Listing 11: Diğer STL Fonksiyonları

9 Performans İpuçları

Not

Vector için performans ipuçları:

- `reserve()` kullanarak bellek tahsis etmeyi optimize edin
- Sık eleman ekleyecekseniz `push_back()` tercih edin
- Ortaya ekleme/silme işlemleri pahalıdır
- 2D vector yerine tek boyutlu vector + hesaplama düşünün

İpucu

STL seçimi için ipuçları:

- Sıralı ve benzersiz elemanlar için `set`
- Hızlı arama için `unordered_set`
- Anahtar-değer çiftleri için `map`
- Tekrar eden elemanlar için `multiset/multimap`

10 Sık Karşılaşılan Hatalar

Dikkat

Dikkat edilmesi gereken noktalar:

- Iterator invalidation (vector resize durumunda)
- `at()` vs `[]`: `at()` bounds checking yapar
- `find()` sonucu kontrol etmeyi unutmayın
- Lambda capture'da reference vs value farkı
- `const` correctness önemlidir

11 Özet Referans Tablosu

İşlem	Fonksiyon	Karmaşıklık
Vector sona ekleme	`push_back()`	O(1) amortized
Vector ortaya ekleme	`insert()`	O(n)
Vector arama	`find()`	O(n)
Vector sıralama	`sort()`	O(n log n)
Set ekleme	`insert()`	O(log n)
Set arama	`find()`	O(log n)
Map erişim	`operator[]`	O(log n)
Binary search	`binary_search()`	O(log n)

Tablo 1: STL Fonksiyonları Karmaşıklık Tablosu