```
/*
 * advanced programming
 * Alireza Akhavan Pour
 * akhavan@alirezaWeb.com
 * date: 1395/03/05
 */
int main()
{
      cout<< "In the name of God";
      Lecture_26 ();
      return 0;
}

string Lecture_26()
{
      cout << "STL" <<endl;
      return ":)"
}
```

1

مقدمه ای بر مفاهیم cohesion و coupling

تعریف **cohesion**:

دلالت دارد بر اینکه component ها یا کلاسها فقط attribute ها (فیلدها) و operationها (متدها)یی را که خیلی با هم مربوط اند یا نزدیک اند را encapsulate کند.

طراحی خوب، طراحی است که cohesion بالا داشته باشد و عملیات نامربوط را در یک کلاس قرار ندهد.

2

مقدمه ای بر مفاهیم cohesion و coupling

تعریف coupling:

مقداری کیفی است که میزان اتصال یا ارتباط بین کلاسها یا componentها یا packageها را مشخص میکند.

به طور کلی هر چه کلاس ها بیشتر به هم مربوط شوند و فیلدها و متدهای هم را استفاده کنند، coupling افزایش میابد.

طراحی خوب، طراحی است که سطح coupling نسبتا پایین داشته باشد (loosely coupled) تا منجر به پیاده سازی، تست، و نگه داری ساده تر شود.

3

Coupling and cohesion

148

Coupling

- Loose: You and the guy at the convenience store. You communicate through a well-defined protocol to achieve your respective goals - you pay money, he lets you walk out with the bag of Cheetos. Either one of you can be replaced without disrupting the system.
- Tight: You and your wife.

Cohesion

- Low: The convenience store. You go there for everything from gas to milk to ATM banking. Products and services have little in common, and the convenience of having them all in one place may not be enough to offset the resulting increase in cost and decrease in quality.
- High: The cheese store. They sell cheese. Nothing else. Can't beat 'em when it comes to cheese though.

share improve this answer

answered Sep 2 '08 at 16:37

Shog9 ♦

104k • 28 • 177 • 213

4

سوالات مهم!

- با این تفاسیر، به نظر شما قرار دادن iterator ها در درون هر container، چرا باعث میشود coupling بین الگوریتم و container کاهش بیابد؟
- آیا اضافه شدن کلاس iterator روی cohesion هم تاثیر دارد؟! چرا و چه طور؟

5

6

تابع find

در این مثال، جستجو به دنبال یک عنصر در یک بردار توسط تابع find که در کتابخانه‌ی STLتعریف شده صورت می‌گیرد. همین تابع را برای یافتن یک کلمه در یک لیست پیوندی از رشته‌ها و یک عدد اعشاری در آرایه‌ای از اعداد نیز به‌کار خواهیم برد.

01-STLFind.cpp

7

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

/*
template<typename I, typename T>
I find(I first, I last, const T& val) {
    while (first!=last && *first != val)
        ++first;
    return first;
}
*/
```

برای استفاده از list

برای استفاده از تابع find

تابع find در STL به این صورت تعریف شده

8

```
int main() {
    vector<int> v;
    v.push_back(10);
    v.push_back(123);
    v.push_back(87);

    vector<int>::iterator p = find(v.begin(), v.end(), 123);
    if (p != v.end())
        cout << "Found " << *p << endl;
    else
        cout << "Not found" << endl;
```

جستجو به دنبال ۱۲۳ در تمام اعضای بردار از v.begin() تا v.end()

9

```
list<string> words;
words.push_back("Money");
words.push_back("For");
words.push_back("Nothing");

list<string>::iterator q = find(words.begin(), words.end(), "Chicks");
if (q != words.end())
    cout << "Found " << *q << endl;
else
    cout << "Not found" << endl;
```

جستجو در کل لیست پیوندی از کلمات

10

جستجو در آرایه:
عبارت
nums + sizeof(nums)/sizeof(double)
اشاره‌گری به یکی بعد از آخرین عنصر آرایه nums خواهد بود.
طرز نوشتن این عبارت موجب می‌شود هنگام تغییر تعداد عناصر آرایه مجبور به تغییر آن نباشیم.

```
double nums[] = {1.4, 50.1, 3.5};
double *r = find(nums, nums + sizeof(nums)/sizeof(double), 50.1);
if (r != nums + sizeof(nums)/sizeof(double))
    cout << "Found " << *r << endl;
else
    cout << "Not found" << endl;
```

11

الگوی لیست پیوندی با تکرارکننده‌ی به سبک STL

در این مثال لیست پیوندی که در بخش های قبل نوشته شد را به تکرارکننده‌ای به سبک STL مجهز می‌کنیم. به این ترتیب از این لیست در بسیاری از الگوریتم‌های STL می‌شود استفاده کرد.

02-STLStyleIterator.cpp

12

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

template<typename T>
class List {
private:
    class Node {
    public:
        Node(T d, Node *n = NULL, Node *p = NULL)
            : data(d), next(n), prev(p) {}

        T data;
        Node *next;
        Node *prev;
    };
```

برای تابع find

برای تابع accumulate

```
public:
    class Iterator : public iterator<input_iterator_tag, T> {
    private:
        Iterator(Node* n) { current = n; }
        friend class List;
    public:
        Iterator& operator++() {
            if (current != NULL)
                current = current->next;
            return *this;
        }

        T& operator*() {
            return current->data;
        }

        bool operator==(const Iterator& it) {
            return current == it.current;
        }

        bool operator!=(const Iterator& it) {
            return current != it.current;
        }
    private:
        Node *current;
    };
```

هنگامی که بخواهیم تکرارکننده‌ای تعریف کنیم که با الگوریتم‌های STL قابل استفاده باشد، آن را زیرکلاسی از الگوی std::iterator تعریف می‌کنیم. پارامتر اول این الگو نوع تکرارکننده را مشخص می‌کند و پارامتر دوم آن، نوع عناصر را.

با تعریف تایپ بازگشتی این تابع به شکل ارجاع، امکان تغییر آن را هم می‌دهیم.

کد کامل را در 02-STLStyleIterator.cpp بخوانید.

14

```
int main() {
    List<int> l;

    l.push_back(86);
    l.push_front(43);
    l.push_front(12);

    cout << accumulate(l.begin(), l.end(), 0) << endl;

    List<int>::Iterator it = find(l.begin(), l.end(), 43);
    if (it != l.end()) {
        cout << "found\n";
        *it = 56;
    } else
        cout << "not found\n";

    vector<string> v;
    v.push_back("Ali");
    v.push_back("aroosi");
    v.push_back("naraft!");

    cout << accumulate(v.begin(), v.end(), string("")) << endl;

    double dd[3] = {1.2, 3.4, 67485.2};
    cout << accumulate(dd, dd + 3, 0.0) << endl;
}
```

محاسبه‌ی مجموع عناصر لیست. پارامتر سوم برای مقداردهی اولیه به متغیری است که مجموع را در خود محاسبه می‌کند.

چون عملگر * ارجاع به عنصر جاری را برمی‌گرداند می‌توان به این شکل مقدار عنصر جاری را تغییر داد.

چسباندن رشته‌های بردار به دنبال هم با به‌کاربردن متوالی عملگر +. پارامتر سوم مقدار اولیه‌ی متغیری را مشخص می‌کند که حاصل کار در آن ذخیره می‌شود.

تابع accumulate برای آرایه‌ها هم قابل استفاده است.

15

مسندها و شیء‌تابع‌ها در STL

این مثال نحوه‌ی استفاده از تابع find_if در کتابخانه‌ی STL را نشان می‌دهد. این تابع در یک مجموعه از عناصر به دنبال اولین عنصری می‌گردد که «شرط خاصی» را ارضاء کند. این شرط (که مسند یا predicate نام دارد) نامیده می‌شود، می‌تواند اشاره‌گر به تابع یا اصطلاحاً شیء‌تابع (function object) باشد که به معنی کلاسی است که عملگر پرانتز را سربارگذاری کرده است.

03-OddSelector.cpp

16

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

/*
template<class In, class Pred>
In find_if(In first, In last, Pred pred) {
    while (first!=last && !pred(*first))
        ++first;
    return first;
}
*/

bool odd(int x) { return x % 2; }

class OddSelector {
public:
    bool operator()(int x) { return x % 2; }
};

int main() {
    vector<int> v;
    v.push_back(10);
    v.push_back(39);
    v.push_back(42);

    vector<int>::iterator it = find_if(v.begin(), v.end(), odd);
    cout << *it << endl;

    vector<int>::iterator it2 = find_if(v.begin(), v.end(), OddSelector());
    cout << *it2 << endl;
}
```

برای استفاده از تابع find_if

تعریف تابع find_if شبیه به این است. پارامتر pred مسندی است که شرط موردنظر روی عناصر را مشخص می‌کند.

تابعی که فرد بودن عدد را تعیین می‌کند

یک شیء‌تابع (function object) که عملگر پرانتز را برای تشخیص فرد بودن عدد ورودی سربارگذاری کرده است.

استفاده از اشاره‌گر به تابع

استفاده از شیء‌تابع: پارامتر سوم یک شیء از کلاس OddSelector است.

استفاده‌ی پیشرفته‌تر از شیء‌تابع‌ها

این مثال، نشان می‌دهد استفاده از شیء‌تابع‌ها می‌تواند پیشرفته‌تر از مثال قبل باشد و شیء تابع مربوطه علاوه بر عملگر پرانتز اعضای دیگری هم داشته باشد. شیء‌تابع تعریف شده در این مثال، دانشجویان را بر اساس نمره cutoff فیلتر می‌کند. دقت کنید که اگر می‌خواستیم از اشاره‌گر به تابع استفاده کنیم لازم بود برای هر مقدار cutoff باید یک تابع مجزا می‌نوشتیم.

04-CutoffFuncObj.cpp

18

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

/*
template<class In, class Pred>
In find_if(In first, In last, Pred pred) {
    while (first!=last && !pred(*first))
        ++first;
    return first;
}
*/
```

تعریف تابع **find_if** شبیه به این است. پارامتر **pred** مسندی است که شرط موردنظر روی عناصر را مشخص می‌کند.

19

```
class student {
public:
    student(string n, double d) : name(n), grade(d) {}
    string get_name() const { return name; }
    double get_grade() const { return grade; }
private:
    string name;
    double grade;
};

class GradeBelow {
public:
    GradeBelow(double c) : cutoff(c) {}
    bool operator()(const student& s) {
        return s.get_grade() < cutoff;
    }
private:
    double cutoff;
};
```

کلاس نگهداری اطلاعات دانشجو

یک شیء‌تابع که دانشجویانی که نمره‌ی آنها زیر **cutoff** است را قبول می‌کند (**true**برمی‌گرداند).

20

```
int main() {
    vector<student> v;
    v.push_back(student("Madani", 8.0));
    v.push_back(student("Kasra", 17.3));
    v.push_back(student("Soheil", 4.5));

    vector<student>::iterator it;
    it = find_if(v.begin(), v.end(), GradeBelow(10));
    cout << it->get_name() << endl;

    it = find_if(v.begin(), v.end(), GradeBelow(5));
    cout << it->get_name() << endl;

    system("pause");
    return 0;
}
```

اولین کسی را پیدا کن که نمره‌اش زیر ۱۰ است

اولین کسی را پیدا کن که نمره‌اش زیر ۵ است

21

منابع

- http://ramtung.ir/apnotes/html/17_STL.html

22