Obyektyönlü proqramlaşdırma

C++

Dərs №17

C++ dili ilə obyektyönlü proqramlaşdırma

Mündəricat

Funktorların istifadəsi

Sizə məlum olduğu kimi STL kitabxanasında funktorlar anlayışı mövcuddur. İndi biz bu anlayışla yaxından tanış olacağıq.

Funktorlar (başqa sözlə desək, funksional obyektlər) - bunlar yüklənmiş funksiya çağırma operatoru ehtiva edən siniflərin xüsusi formasıdır. Bir qayda olaraq, funktor funksiya tələb olunan istənilən yerdə tətbiq edilə bilər. Başqa sözlə funksiyanın çağırılması tələb olunduğunda funksiyanın çağırılması üçün yüklənmiş operator çağırılır ("dairəvi mötərizələr" operatoru).

Faktorun adi funksiyadan əsas fərqi hər hansı bir qiymətin statik dəyişənlər kimi saxlanılması imkanıdır. İş prinsipi belədir:

Funktora ilkin müraciət funktorun özünün qiymət verdiyi konstruktoru çağırır. Adi funksiyanın istifadə edilməsi zamanı ilkin qiymətləndirməni icra etmək üçün "biclik" etmək lazım gəlir. Bundan başqa, bəzi mənbələr deyir ki, funktorun göstərici vasitəsilə çağırılması adi funksiyaya nisbətən daha tez baş verir.

Funktorların təyinatını daha ətraflı anlamaq üçün nümunəyə baxaq. Vurma cədvəlini reallaşdıraq. Növbəti tələblər qoyulur: Birinci vuruq soldan saga artır, ikinci isə yuxarıdan aşağıya. Bu iterasiya alqoritminə klassik nümunədir.

Tapşırığın yerinə yetirilmə alqoritmini gözdən keçirək:

- Bir ədəd üçün xüsusi dəstin hasil edilməsi və bir neçə ədəd üçün hasiletmənin təkrarlanması.
  - Konteyner-siyahının (vektorun) qiymətləndirilməsi (inisiallaşdırılması).
  - Konteyner-siyahının (vektorun) doldurulması.
  - Konteyner-siyahının (vektorun) qiymətinin ekranda çap edilməsi.
- Verilmiş ədəddən başlayaraq və hər dəfə onu müəyyən qiymət artıran ardıcıl ədədlər hasil edən funktorun yaradılması.

```
//alqoritmlərlə işləmək üçün kitabxana
//biz ona sonra daha ətraflı baxacayıq
#include <algorithm>
#include <iostream>
//konteyner-siyahı kitabxanası
 #include <list>
using namespace std;

/*
İki sahə ehtiva edən funksional obyekt:
1. Artırma qiymətinin (delta) saxlanılması
2. Cari qiymət (current)üçün ədəd hasil edilir
*/
class addNumberFrom
{
    int delta;
        int current;
/*
Sinif konstruktoru qiymətin artımını və cari
qiymətini verir. Sonuncu olmaya da bilər, onda o
sıfır qəbul edilir.
*/
```

```
public:
    addNumberFrom(int number, int from =
                  0):delta(number), current(from) {}
/*
Funktorun əsası - cari hasil edilən ədədə artımın
qiymətini əlavə edən funksiyanın çağırılmasının
yüklənməsi operatorudur
*/

    int operator()()
    {
        return current += delta;
    }
};

//Vurma cədvəlinin baslığının ekrana verilməsi

void main()
{
    cout<<"TABLE:"<<"\n\n";
    cout << "----------" <<"\n\n";

    for(int i=1; i<=10; i++)
    {
        //Konteyner-siyahının yaradılması
        list<int> l(10);

        /*
        generate_n alqoritminin çağırılması. Aydındır
        ki, o özü heç bir şey hasil edə bilməz, lakin
        başlanğıc qiymətlərin və siyahının
        elementlərinin sayı verilmiş diapazonda
        qiymətlər hasil edir. Ədədin hər bir qiymətə
        yazılması üçün üçüncü parametrin istinad
        etdiyi funksiya çağırılır: */
```

```
        generate_n(l.begin(), l.size(), addNumberFrom(i));

        /*
        Lakin biz funksiya yerinə funksiyanın
        çağırılmasının yüklənməsi operatorunu -
        addNumberFrom obyektini qoyuruq. Əkər çağırılış
        ilk dəfə baş verirsə, onda obyektin konstruktoru
        çağırılır. O delta sahəsinə i qiymətini verir,
        current sahəsinə isə susmaya görə funksiyanın
        ikinci parametrinin qiymətini verir, yəni 0.

        Beləliklə, konteyner-siyahı i dəyişəninin
        dəyişməsi ilə və 1-dən 10-a qədər hasillərlə
        doldurulur. generate_n alqoritmində
        siyahıdakı elementlərin sayını qaytaran
        size()metodu istifadə edilir. Əgər başlanğıc
        və son iteratorlar varsa, onda ən yaxşısı
        Generate alqoritmindən istifadə etməkdir.
        */

        //Konteyner-siyahıdan ədədin göstərilməsi

        copy(l.begin(), l.end(),
                    ostream_iterator<int>(cout,"\t"));

    }
}
```

Ümüd edirik ki, nümunə funktoru daha dərindən anlamağa imkan verdi. İndi qeyd edək ki, müxtəlif hesabi və məntiqi əməliyyatlar, həmçinin müqayisə əməliyyatları standart STL vasitələri ilə, yəni artıq mövcud olan funktorlar dəsti ilə qismən reallaşdırıla bilər:

1. ***Hesabi funktorlar:***
   - **plus** toplama **x + y**;
   - **minus** çıxma **x - y**;
   - **multiplies** vurma **x · y**;
   - **divides** silmə **x / y**;
   - **modulus** qalığın alınması **x % y**;
   - **negate** işarənin tərsi **-x**;
2. ***Müqayisə funktorları:***
   - **equal_to** bərabərdir **x == y**;
   - **not_equal_to** bərabər deyil **x != y**;
   - **greater** çox **x > y**;
   - **less** az **x < y**;
   - **greater_equal** böyük və ya bərabər **x >= y**;
   - **less_equal** kiçik və ya bərabər **x <= y**;
3. ***Məntiqi funktorlar:***
   - **logical_and** mətiqə "və" **x && y**;
   - **logical_or** mətiqə "və ya" **x || y**;
   - **logical_not** mətiqə "fərqlidir" **! x.**

Predikatların istifadəsi

Və yenə də STL... Daha bir anlayışa biz sizinlə ətraflı baxmamışıq ki, bu da predikatlardır.

Situasiyadan asılı olaraq qərar qəbul edən və sizin hansısa bir proqramınızı koordinasiya edən xüsusi vasitələr sizə lazım ola bilər. Adaətən bu məqsədlə biz dilin məntiqi ifadələrini istifadə edirik. Lakin STL-də bu problem başqa cür həll etmək qərarlaşdırılıb. Məhz predikatların yaradılması ilə.

Predikat - məntiqi qiymət (true və ya false) qaytaran xüsusi funksiyadır.

Yəqin ki siz müəyyən etdiniz ki, predikat yaratmaq kifayət qədər asandır - sadəcə bool tipi qaytaran funksiya yazamaq kifayətdir. Nümunəyə baxaq. Ədədin cütlüyünü təyin edən predikat yaratmağa çalışaq. Əgər ədəd cüt olarsa, predikat true qiymətini qaytaracaq. Növbəti ardıcıl əməliyyatları istifadə edək:

- Cütlük ədədin ikiyə bölünməsindən alınan qalıqla müəyyənləşdirilir.
- Sonra, konteyner-siyahı 1-dən 10-a qədər qiymətlərlə doldurulur (bunun üçün qiyməti siyahının sonuna əlavə edən **push_back()** metodu çağırılır).
- Sonra qiymət ekrana verilir.
- Sonra axtarış və bütün cüt qiymətlərin seçilməsi həyata keçirilir.

Qeyd: Bu məsələni remove_if alqoritmi həll edir. Diqqət edin ki, _if ilə bitən əksər alqoritmlər sonuncu parametr kimi predikatlardan istifadə edirlər. remove_if alqoritmi silinən elementdən sağda gələn bütün qiymətləri konteynerin əvvəlinə doğru yerləşdirir. Bu səbəbdən biz hesab edəcəyik ki, qaytarılan iteratorun istinad etdiyi ünvanından başlayaraq konteynerin verilənlər oblastının sonuna qədər predikatın false qiymətini qaytardığı silinməmiş qiymətlər yerləşir. Burada bizim misalda verilmiş xüsusiyyəti növbəti şəkildə istifadə edirik: bütün cüt qiymətlə ekrana verilir (bunun üçün biz onları silinən qiymətlərin oblastını əks etdirən iteratorlar cütlüyünü axın iteratoruna ötürməklə çıxış axınına köçürürük).

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

//predikat
bool isEven(int num)
{
    return bool(num % 2);
}

void main()
{
    list<int> l;
        list<int>::iterator t;
```

```
    for(int i=1; i<=10; i++)
        l.push_back(i);

    copy(l.begin(), l.end(),
        ostream_iterator<int>(cout," "));
    cout<<"\n\n";
    t=remove_if(l.begin(), l.end(), isEven);
    copy(l.begin(), t, ostream_iterator<int>(cout," "));
}
```

Alqoritmlərin istifadəsi

Artıq bildiyiniz kimi STL kitabxanasında müxtəlif standart əməliyyatları yerinə yetirən funksiyalar mövcuddur. Bu əməliyyatlar, məsələn, axtarış, çevirmə, çeşidləmə, köçürmə və s. kimi əməliyyatlardır. Bu funksiyalar alqoritmlər adlanırlar. Alqoritmlər üçün parametrlər kimi iteratorlardan istifadə edilir.

Qeyd etmək lazımdır ki, alqoritm üçün ona hansı tip iteratorun ötürülməsi əhəmiyyət kəsb etmir. Əsas odur ki, bu iterator müəyyən bir qrupa düşsün. Məsələn, əgər alqoritmin parametri biristiqamətli iterator olarsa, onda otürülən iterator ya biristiqamətli, ya ikisitiqamətli, ya da ixtiyari müraciətli iterator ola bilər.

Alqoritmlər haqqında əsas məlumat

Bütün alqoritmlər iki qrupa bölünürlər: verilənləri dəyişən və dəyişməyənlər.

Hər bir alqoritm funksiya şablonu və ya funksiya şablonları dəstidir. Yəni, istənilən alqoritm tamamilə müxtəlif konteyner ilə işləyə bilər.

İterator qaytaran alqoritmlər adətən müvəffəqiyyətsizlik xəbərdarlığı üçün giriş verilənlərinin sonunu istifadə edirlər.

Alqoritmlər diapazonun yoxlanılmasını onların giriş çıxışı zamanı icra etmirlər.

Əgər alqoritm iterator qaytarırsa, bu elə alqoritmin girişindəki tipdə iterator olacaq.

Alqoritmlər <algorithm> başlıq faylında təyin edilmişdir.

STL kitabxanasının ən çox istifadə edilən alqoritmlərini nümunə göstərək.

1. ***Modifikasiya olumayan əməliyyatlar:***
   - **for_earch()** - ardıcıllığın hər bir elementi üçün əməliyyatı icra edir;
   - **find()** - ardıcıllıqda qiymətin ilk rast gəlməsini tapır;
   - **find_if()** - ardıcıllıqda predikata ilk uyğun gələni tapır;
   - **count()** - ardıcıllıqda qiymətin rast gəlmə sayını hesablayır;
   - **count_if()** - ardıcıllıqda predikatın icra olunma sayını hesablayır;
   - **search()** - ardıcıllığın altardıcıllıq kimi ilk rast gəlməsini tapır;
   - **search_n()** - ardıcıllıqda n təkrarlanması olan altardıcıllığı tapır və onun ilk rast gəlməsini qaytarır.
2. ***Modifikasiya olunan əməliyyatlar:***
   - **copy()** - birinci elementdən başlayaraq ardıcıllığı köçürür;
   - **swap()** - iki elementin yerini dəyişdirir;
   - **replace()** - elementləri göstərilən qiymətlərlə dəyişdirir;
   - **replace_if()** - predikatın icrası zamanı elementləri dəyişdirir;

- **replace_copy()** - elementləri göstərilmiş qiymətlərlə dəyişdirərək ardıcıllığı küçürür;
- **replace_copy_if()** - predikatın icrası zamanı elementləri dəyişərək ardıcıllığı köçürür;
- **fill()** - bütün elementləri verilmiş qiymətlə dəyişdirir;
- **remove()** - verilmiş qiymətli elementləri silir;
- **remove_if()** - predikatın icrası zamanı elementləri silir;
- **remove_copy()** - verilmiş qiymətli elementləri silərək ardıcıllığı köçürür;
- **remove_copy_if()** - verilmiş qiymətli elementləri silərək ardıcıllığı köçürür;
- **reverse()** - elementlərin ardıcıllığını tərs çevirir;
- **random_shuffle()** - elementlərin təsadüfi bərabər paylanmasını nəzərə alaraq yerdəyişdirir (ardıcıllığı "qarışdırır");
- **transform()** - ardıcıllığın bütün elementləri üzərində verilmiş əməliyyatı icra edir;
- **unique()** - iki qonşu bərabər elementi silir;
- **unique_copy()** - iki qonşu bərabər elementi silərək ardıcıllığı köçürür.

3. ***Çeşidləmə:***

- **sort()** - ardıcıllığı yaxşı orta səmərəliklə çeşidləyir;
- **partial_sort()** - ardıcıllığın bir hissəsini çeşidləyir;
- **stable_sort()** - bərabər elementlərin ardıcıllığını saxlayaraq ardıcıllığı çeşidləyir;

- **lower_bound()** - verilmiş qiymətdən kiçik olan ən kiçik elementi tapır;
- **upper_bound()** - verilmiş qiymətdən böyük olan ən kiçik elementi tapır;
- **binary_search()** - çeşidlənmiş ardıcılıqda verilmiş elementin olmasını təyin edir;
- **merge()** - iki çeşidlənmiş ardıcıllığı birləşdirir.

4. ***Çoxluqlarla iş:***
   - **includes()** - daxil olmanın yoxlanılması;
   - **set_union()** - çoxluqların birləşdirilməsi;
   - **set_intersection()** - çoxluqların kəsişməsi;
   - **set_difference()** - çoxluqların fərqi.
5. ***Minimum və maksimumlar:***
   - **min()** - ikisindən ən kiçiyi;
   - **max()** - ikisindən ən böyüyü;
   - **min_element()** - ardıcıllıqda ən kiçik element;
   - **max_element()** - ardıcıllıqda ən böyük element.
6. ***Yerdəyişdirmə:***
   - **next_permutation()** - leksikoqrafik ardıcıllıqda növbəti yerdəyişdirmə;
   - **prev_permutation()** - leksikoqrafik ardıcıllıqda əvvəlki yerdəyişdirmə.

«Alqoritmin mürəkkəbliliyi» anlayışı

Bu gün biz sizinlə artıq alqoritmlər anlayışı ilə tanış olduq. Çox zaman müəyyən formalı alqoritmlər dəstindən optimal alqoritmin seçilməsi lazım gəlir. Bu zaman həll edilən məsələdə cari verilənlərin ölçüsünün məhdudiyyəti və əməliyyat sisteminin parametrləri, prosesin tezliyi, yaddaşın tutumunu nəzərə almaq lazımdır.

Həqiqəti müəyyən etmək üçün nəzəriyyədə konkret deyil, kütləvi məsələlərin həlli alqoritmlərinə baxılır. Kütləvi məsələ sonsuz sayda indivudal məsələr ardıcıllığını əks etdirir. İndividual məsələ bu məsələnin həlli üçün tələb olunan giriş verilənlərinin həcmi ilə xarakterizə edilir. Bu zaman əgər individual məsələnin ölçüsü hər hansı bir natural n ədədi olarsa, onda kütləvi məsələlərin həlli alqoritminin mürəkkəbliliyi n arqumentli funksiya olur.

Başlanğıc üçün uzunluğu n olan bütün ikilik açarın sadə seçilməsi alqoritminə baxaq. Belə ki, məlumdur ki, bu cür açarların sayı 2 üstü n-dir. Başqa sözlə bu alqoritmin mürəkkəbliliyi 2 üstü n-dir. Belə alqoritm sadə seçim variantıdır və **eksponensial alqoritm** adlanır.

İndi isə iki n-rəqəmli ədədin sütunla vurulması alqoritminə baxaq. O birrəqəmli ədədlərin n üstü 2 sayda vurulmasından ibarətdir və onun mürəkkəbliliyi bu vurmaların sayı ilə təyin edilir ki, bu da təbii ki, n üstü 2-yə bərabərdir. Belə alqoritm ***polinominal alqoritm*** adlanır.

Yuxarıda verilənlərin hamısı tamamilə sadə nəzəriyyədir və savadlı yazılmış bir proqramın malik olduğu yeganə keyfiyyət olmadığını bizim başa düşməyimizə imkan verir. Müxtəlif giriş verilənləri üçün proqramın icrası müddəti ilə xarakterizə edilən səmərəlilik - bu proqramın daha bir vacib üstünlüyüdür.

Konkret proqram üçün dəqiq asıllılığının tapılması - bu çox mürəkkəb məsələdir. Bu səbəbdən konkret alqoritmin asimptotik qiymətləndirilməsi ilə kifayətlənirlər, yəni əsas parametrin böyük qiymətlərində onun təxmini davranışının təsviridir. Bəzən asimtotik qiymətləndirmə üçün iki funksiya arasında ənənəvi O (böyük O) münasibəti istifadə edilir. Məsələn, $f(n) = n^2/2 + 3n/2 + 1$ funksiyası təxminən $n^2/2$ funksiyası kimi artır (biz asta artan 3n/2+1 cəmini atırıq). 1/2 konstant vuruğu da atılır və alqoritm üçün $O(n^2)$ kimi işarə edilən asimptotik qiymətləndirmə əldə edirik.

Burada biz müxtəlif mürəkkəblilikdə olan alqoritmlərin icra olunma zamanlarının müqayisə cədvəlini verək və müəyyənləşdirək ki, nə üçün kompüterin məhsuldarlığının artırılmasında sürətli alqoritmlərin istifadəsinin əhəmiyyəti çox artmışdır.

Eyni bir məsələnin mürəkkəbliliyi log(n), n, n^2 və 2^n olan dörd müxtəlif həlli alqoritmi gözdən keçirək. Hesab edək ki, onlardan ikincisi hansısa bir kompüterdə öz həlli üçün müəyyən parametrə görə (məsələn, 10 üstü 3) bir dəqiqə zaman sərf edir. Bu halda bu 4 alqoritmin həmin kompüterdə həlli müddəti təxminən növbəti şəkildə olacaq:

Alqoritmin mürəkkəbliliyi	n=10	$n=10^3$	$n=10^6$
log n	0.2 san.	0.6 san.	1.2 san.
n	0.6 san.	1 dəq.	16.6 saat.
n^2	6 san.	16.6 saat.	1902 il
2^n	1 dəq.	10^{295} il	10^{300000} il

Beləliklə yekunlaşdıraq.

Kompüterlərin sürətinin artması ilə bu və ya digər alqoritmin işi üçün parametrlərin qiymətləri də artır. Beləliklə, kəmiyyətlərin orta qiyməti artır, asta və sürətli alqoritmlərin icra zamanına nisbəti kəmiyyəti də artır. Kompüter nə qədər sürətlidirsə, zəif alqoritmin istifadəsində nisbi qazanc bir o qədər çoxdur!

İmtahan

Tapşırıq I

OYP (siniflər, varislilik və s.) istifadə edərək sadə faylları idarəetmə sistemi hazırlayın.

Faylları idarə edmə sistemi disklərin məzmununu göstərmə, qovluq/fayllar yaratmaq, qovluqları/faylları silmək, qovluqların/faylların adlarını dəyişdirmək, qovluqları/faylları köçürmək/yerini dəyişdirmək, qovluqların/faylların ölçülərini hesablamaq, şablona görə axtarış aparmaq və s. kimi imkanlara malik olmalıdır.

Tapşırıq II

Tamfunksional test sisteminin reallaşdırılması.

Bu sistemdə iki rejim olmalıdır: adminstrator və test olunan.

Test olunan üçün iş rejiminin təsviri (gələcəkdə qonaq adlandıracağıq):

- Sistemə daxil olmaq üçün qonaq qeydiyyatdan keçməlidir, bu prosedur bir dəfə yerinə yetirilməlidir. Sistemə növbəti girişlərdə giriş loqin və parola görə olur.
- Qeydiyyat zamanı A.S.A., ev ünvanı, telefon məlumatları daxil edilməlidir.
- İstifadəçi üçün loginin unikal olması lazımdır.

■■Daxil olduqdan sonra qonaq özünün əvvəlki test nəticələrini görə bilər, növbəti testləşdirmədən keçə bilər.

Testləşdirmə müxtəlif bilik kateqoriyası üzrə həyata keçirilə bilər. Məsələn:

Riyaziyyat (böllmə) -> Diskret riyaziyyat (konkret test)
-> Riyazi analiz (konkret test)

Fizika (bölmə) -> Kvant fizikası (konkret test)
-> Mexanika (konkret test)

■■Test verildikdən sonra qonaq testin nəticəsini, düzgün cavabların sayını, döğru cavabların faizini və aldığı qiyməti görə bilir.

■■Tələbə test prosesini yarıda dayandıra bilər və onu uyğun zamanda davam etdirə bilər.

■■Qiymətləndirmə test suallarının sayına görə 12 ballı sistem ilə aparıla bilər.

■■Qonaqların loqin və parolları şifrələnmiş şəkildə saxlanılır.

Administrator (gələcəkdə admin) rejimi üçün iş rejiminin təsviri:

■■ Sistemdə yalnız bir admin ola bilər, adminin loqin və parolu proqrama ilk daxil olunduğu zaman verilir.

■■Gələcəkdə loqin və parolu dəyişdirmək olar (lakin bunu yalnız admin edə bilər).

- Parol və loqini yalnız şifrələnmiş formada saxlamaq lazımdır.
- Sistemlə işləyərkən admin növbəti imkanlara sahibdir:
  - İstifadəçilərin idarə edilməsi – yaradılması, silinməsi, dəyişdirilməsi.
  - Statistikaya baxış – test nəticələrinə ümumi kateaoriyalar üzrə, konkret testə görə, konkret istifadəçiyə görə baxış. Statistikanın baxış nəticələrini faylda saxlamaq olar.
  - Testin idarə edilməsi – admin sistemə kateqoriya, testlər, testlər üçün suallar əlavə edə bilər, doğru və yanlış cavabları verə bilər, fayldan kateqoriyalar və testlər import və eksport edə bilər.

Tapşırığın yerinə yetirilməsi zamanı tam mənada OYP və STL apparatından istifadə etmək lazımdır.

Kod oxunaqlı, yaxşı şərhlərlə meşayiət edilmiş olmalı və identifikatorların adlandırılmasının vahid sistemindən istifadə edilməlidir.