ШАГ компьютерная академия

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

Dərs №15

Çoxölçülü dinamik massivlər

Mündəricat

1. Çoxölçülü dinamik massivlər

Yenə, döyüşə! Biz sizinlə artıq dinamik massivlərlə rastlaşmışıq, lakin bu mövzuya yenidən toxunmaq və çoxölçülü massivlərin yaradılması haqqında sizə müəyyən məlumatlar vermək istəyərdik.

C dilində çoxölçülü massiv əslində birölçülüdür. new və delete operatorları sanki, ixtiyari ölçünü dəstəkləyən dinamik massiv yaratmağa və silməyə imkan verir. Dinamik massiv yaratma əməliyyatı əsas üstünlük təşkil edən əlavə diqqət tələb edir, belə ki, massivin xarakteristikası (new əməliyyatının operandları) sabit ifadə olmaya bilər. Bu ixtiyari konfiqurasiyada çoxölçülü dinamik massiv yaratmağa imkan verir.

```
#include  <iostream>
using      namespace
std; void main()
{
    int i, j;

    // Massivlərin xarakteristikalarını təyin edən
    dəyişənlər.
    int m1 = 5, m2 = 5;
    /* İkiölçülü dinamik massivin təşkili iki
       mərhələdə aparılır.
```

```
        Əvvəlcə birölçülü göstəricilər
        massivi yaradılır, sonra  isə bu
        massivin hər bir elementinə birölçülü
        massivin ünvanı mənimsədilir.
        Massivlərin ölçüləri üçün sabit
        ifadələr tələb edilmir.
     */
    int **pArr = new int*[m1];
    for (i = 0; i < m1; i++)
    pArr[i] = new int[m2];

    pArr[3][3] = 100;
    cout << pArr[3][3] << "\n";

    // İkiölçülü massivin ardicil məhv edilməsi…
    for (i = 0; i < m1; i++)
        delete[]pArr[i];
    delete[]pArr;
}
```

2. Çoxölçülü dinamik massivlərə aid nümunələr

Nümunə 1. İkiölçülü "üçbucaqlı" dinamik massivin təşkili

Əvvəlcə birölçülü göstəricilər massivi yaradılır, sonra bu massivin hər bir elementinə birölçülü massivin ünvanı mənimsədilir. Bu zaman hər bir yeni masivin ölçüsü (elementlərinin sayı) əvvəlkindən bir vahid azdır. new əməliyyatının operandı olan kvadrat mötərizələr arasındakı dəyişən bunu asanlıqla etməyə imkan verir.

```
#include <iostream>

using namespace std;

void main()
{
    int i, j;
    // Massivlərin xarakteristikalarını təyin edən
    dəyişənlər.
     int m1 = 5, wm = 5;
    int **pXArr = new int*[m1];

    for (i = 0; i < m1; i++, wm--)

        pXArr[i] = new int[m1];

    // Massivin sıfırlarla doldurulması və ekran
    verilməsi
     for (i = m1 - 1; i >= 0; i--, wm++) {
        for (j = 0; j < wm; j++){

            pXArr[i][j]=0;
```

```
            cout<<pXArr[i][j]<<"\t";
        }
        cout<<"\n\n";
    }

    /* Üçbucaqlı konfiqurasiyalı ikiölçülü massivin
    ardıcıl ləğv edilməsi
        */

    for (i = 0; i < m1; i++)
        delete[]pXArr[i];
    delete[]pXArr;
}
```

Misal 2. Üçölçülü dinamik massivin təşkili

Üçölçülü massivin yaradılması və ləğv edilməsi əlavə iterasiya tələb edir. Lakin burada həmçinin prinsipcə yeni bir şey yoxdur.

```
#include  <iostream>
using      namespace
std; void main()
{
    int i, j;

    // Massivlərin xarakteristikalarını təyin edən
    dəyişənlər.
    int m1 = 5, m2 = 5, m3 = 2;

    // Göstəricinin göstəricisinin göstəricisi:)
    int ***ppArr;
    // Massivin yaradılması
    ppArr = new int**[m1];
    for (i = 0; i <m1; i++)
        ppArr[i] = new int*[m2];
```

```
        for (i = 0; i < m1; i++) for
            (j = 0; j < m2; j++)
                ppArr[i][j] = new int[m3];

    ppArr[1][2][3] = 750;
    cout << ppArr[1][2][3] << "\n";

    // əks yaradılmışı ardıcil ləğvetmə
    for (i = 0; i < m1; i++) for
        (j = 0; j < m2; j++)
            delete[]ppArr[i][j];

    for (i = 0; i < m1; i++)
        delete[]ppArr[i];
    delete[] ppArr;
}
```

3. Sadalanan tiplər

Sadalanan tip enum açarsözü ilə verilir və istifadəçi tərəfindən təyin edilmiş qiymətlər dəstini verir. Qiymətlər dəsti fiqurlu mötərizələr ilə əhatə olunur və öz adları ilə verilən tam adlandırılmış dəstdir. Elana baxaq:

```
enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES};
```

Onun köməyi ilə tamqiymətli sabitlərlə adlandırılan dörd add dəsti ehtiva edən tam qiymətli tip yaradılır. Sadalanan sabitlər – qiymətləri uyğun olaraq 0, 1, 2 və 3 olan CLUBS, DIAMONDS, HEARTS и SPADES adlarıdır. Bu qiymətlər susmaya görə mənimsədilmişdir. İlk sadalanan sabitə tam qiymətli 0 mənimsədilir. Siyahının hər bir növbəti üzvü ondan solda yerləşən qonşusundan bir vahid böyükdür. İstifadəçi tərəfindən təyin edilmiş Suit dəyişəninə siyahidakı qiymətlərdən biri mənimsədilə bilər.

Sadalanan tipə aid da bir geniş yayılmış nümunə:

```
enum Months {JAN = 1, FEB, MAR, APR, MAY, YUN,
             YUL AUG, SEP, OCT, NOV, DEC};
```

Bu elan nəticəsində istifadəçi tərəfindən təyin edilmiş Months tipi yaradılır ki, bu tip də ayların adlarından ibarət sabitlər ehtiva edir.

Belə ki, sadalanan ilk qiymət 1-ə bərabər olduğu üçün növbəti qiymətlər 1-dən 12-yə qədər artır.

Sadalanan tip elan edərkən sadalanan istənilən sabitə tam qiymət mənimsədilə bilər.

***Qeyd: Ənənəvi səhv**. Sadalanmanın sabiti təyin edildikdən sonra ona başqa qiymətin mənimsədilməsinə cəhd sintaksis səhv sayılır.*

Sadalamanın istifadəsi zamanı əsas hallar:

1. Tam sabitlərin yerinə sadalanmadan istifadə etmək proqramın oxunaqlığını artırır.
2. enum adları bircinsli olmalıdırlar, lakin sadalanmanın ayrı sabitləri eyni tam qiymətlər ola bilər.
3. Sadalanan tipin adlar dəsti digər tam tiplərdən fərqli xüsusi bircinsli tipdir. Sadalanan sabitlər ixtiyari tamqiymətli sabitlər, həmçinin sabit ifadələr ilə təyin edilə və qiymətləndirilə bilərlər:

```
enum ages (milton = 47, ira, harold =
           56, philip = harold + 7};
```

***Qeyd:** Diqqət edin ki, aşkar qiymətləndirmə yoxdursa, susmaya görə qayda tətbiq edilir, beləliklə - ira = 48. Bundan başqa, sadalanan sabitlərin qiyməti bircinsli olmaya bilər.*

5. Hərbir sadalanan ayrı tipdir. Sadalama elementinin tipi sadalamanın özüdür. Məsələn,

```
enum Keyword {ASM, AUTO, BREAK};
```

AUTO Keyword tipindədir.

6. Sadalanan sabit anonim elan edilə bilər, yəni tipin adı olmadan.

```
enum {FALSE, TRUE};
enum {lazy, hazy, crazy} why;
```

İlk elanetmə — mnemonik tamqiymətli sabitlərin elan edilməsinin geniş yayılmış üsuludur. İkinci elanetmə qiymətləri lazy, hazy və crazy qiymətlərini ala bilən sadalanan why tipində dəyişən elan edir.

7. Sadalanma qeyri-aşkar adi tamqiymətli tipə çevrilə bilər, lakin əksinə yox.

```
enum boolean {FALSE, TRUE} q;
enum signal {off, on} a = on; // a on qiymətini alır.

enum answer {no, yes, maybe = -1} b;

int i, j = true; // doğru true 1-ə çevrilir
a = off;// doğru
i = a;  // doğru i 1 olur
q = a;  // iki müxtəlif tip doğru deyil

q = (boolean)a; // doğru aşkar çevrilməsi
```

4. Funksiya göstəricisi

Göstərici funksiya daxil etməzdən əvvəl, xatırladaq ki, hər bir funksiya qaytarma tipi, adı və arqumentləri ilə təyin edilir. Xatırladaq ki, arqumentlər ardicilliğa və tipə görə fərqləndirilən parametrlərdir. Bəzən deyirlər ki, funksiyanın arqumenti onun parametrlərinin tipləri cədvəlidir.

İndi isə, ardıcıl müddəa ilə dərsin bu bölməsinin mövzusunun müzakirəsinə keçək.

1. Funksiyanın adının mötərizələrsiz və parametrlərsiz istifadə edilməsi, funksiyanın adı onun göstəricisi kimi çıxış edir və onun qiyməti funksiyanın yaddaşda yerləşdirilməsi ünvanıdır.
2. Ünvanın bu qiyməti hər hansı bir göstəriciyə mənimsədilə bilər və bu yeni göstərici funksiyanın çağırılması üçün tətbiq edilə bilər.
3. Yeni göstəricinin təyin edilməsində funksiyanın qaytardığı qiymətin tipinə uyğun tip və parametrlər verilməlidir.
4. Funksiya göstəricisi növbəti şəkildə təyin edilir:

```
funksiya_tipi (*göstərici_adı)(parametrlərin
xüsusiyyətləri);
```

Misal: int (*func1Ptr) (char); — func1Ptr adında funksiya göstəricisinin char parametri və int qiymət qaytarma tipində təyin edilməsi.

Qeyd: Diqqətli olun!!! *Əgər verilmiş sintaksis struktur ilk dairəvi mötərizələrsiz yazılarsa, yəni, int *fun (char); kimi olarsa, onda kompilyator onu adı fun, parametri char tipi, göstəricinin qaytarma tipi int* olan funksiya kimi qəbul edəcəkdi.*

İkinci misal: char * (*func2Ptr) (char * ,int); - char göstərici tipi və int tipli parametrləri qaytarma char göstərici tipi, adı func2Ptr olan funksiya təyin edilir.

Təcrübədə nömayiş etdirək

Funksiya göstəricisinin təyinində qaytarılan qiymətin tipi və arqumenti (parametrlərin sayı və ardıcıllığı), ünvanı daxil edilən göstəriciyə mənimsədilməsi nəzərdə tutulan funksiyanın tipi və arqumenti ilə üst-üstə düşməlidir. Deyilənləri əyani nümayiş etdirmək üçün funksiya göstəricisi olan proqramı verək:

```
#include <iostream>

using namespace std;

void f1(void)  // f1 funksiyasının təyini.
{
    cout << "Load f1()\n";
}
void f2(void)  // f2 funksiyasının təyini.
{
    cout << "Load f1()\n";
}
```

```
void main()
{
    void (*ptr)(void); // ptr – funksiya göstəricisi.
    ptr = f2;          // f2() ünvanı mənimsədilir.
                       // f2()-nin ünvana görə
    (*ptr)();          çağırılması
    ptr = f1;          // f2() ünvanı mənimsədilir.
                       // f2()-nin ünvana görə
    (*ptr)();          çağırılması.
                       // (*ptr)();- ekvivalent
    ptr();             çağırılma
}

Proqramın icrasının nəticəsi:
    Load f2()
    Load f1()
    Load f1()
    Press any key to continue
```

Burada, göstərici_adı-nın qiyməti funksiyanın ünvanıdır, * operatoru isə funksiyaya ünvana görə müraciət etməyə imkan verir. Lakin funksiyanın mötərizələrsiz *ptr(); şəklində yazılması səhv olacaq. Məsələ ondadır ki, mötərizələr * - ünvana görə müraciət operatorundan daha yüksək prioritetə malikdirlər. Nəticə olaraq, sintaksisə uyğun olaraq əvvəlcə ptr() funksiyasına müraciət edilməsinə cəhd ediləcəkdir və nəticədə adayükləmə əməliyatı baş verəcəkdir ki, bu da sintaksis xəta verəcək.

Təyin edilən zaman funksiya göstəricisi qiymətləndirilə bilər. İlkin qiymət kimi təyin edilən göstəriciyə uyğun funksiyanın ünvanı, tipi və parametrləri isitifadə edilməlidir.

Funksiya göstəricisinə mənimsətmə zamanı funksiyanın qaytarma qiymətinin və arqumentlərinin tipinin mənimsətmə operatorunun sol və sağ tərəfindəkinə uyğun gəlməsinə diqqət etmək lazımdır. Funksiyanın gələcəkdə göstərici vasitəsilə çağırılması zamanı da bunu diqqətə almaq lazımdır, belə ki, funksiyaya ünvana görə müraciət zamanı istifadə edilən faktik parametrlərin tipi və sayı çağırılan funksiyanın formal parametrlərinə uyğun gəlməlidir:

```
char f1(char) {...}     // Funksiyanın təyini.
char f2(int) {...}      // Funksiyanın təyini.
void f3(float) {...}     // Funksiyanın təyini.
int* f4(char *){...}    // Funksiyanın təyini.
char (*pt1)(int);         // Funksiya göstəricisi.
char (*pt2)(int);         // Funksiya göstəricisi.
void (*ptr3)(float) = f3; // Qiymətləndirilən
                          // göstərici.

void main()
{
    pt1 = f1;  // Səhv   - arqumentin uyğunsuzluğu.
    pt2 = f3;  // Səhv -   tip uyğunsuzluğu
               //(qiymət
               və arqument ).
    pt1 = f4;  // Səhv -   Tiplərin uyğunsuzluğu.
    pt1 = f2;  // Doğrudur.
    pt2 = pt1; // Doğrudur.
    char c = (*pt1)(44); // Doğrudur.
    c = (*pt2)('\t');    // Səhv - arqumentin
                         // uyğunsuzluğu.
}
```

Növbəti proqram funksiyanın göstəricilər vasitəsilə çağırılmasını əyani əks etdirir.

```
#include <iostream>
using namespace std;
// eyni arqumentli və tipli funksiyalar:
int add(int n, int m) { return n + m; }
int division(int n, int m) { return n/m;
} int mult(int n, int m) { return n * m;
} int subt(int n, int m) { return n - m;
} void main()
{
    int (*par)(int, int); // Funksiya göstəricisi.
    int a =   6, b = 2;
    char c = '+';
    while    (c != ' ')
    {
        cout << "\n Arguments: a = " << a
                 <<", b = " << b;
        cout << ". Result for c = \'" << c << "\':";

        switch (c)
        {
            case '+':
                par = add;
                c = '/';
                break;
            case '-':
                par = subt;
                c = ' ';
                break;
            case '*':
                par = mult;
                c = '-';
                break;
            case '/':
                par = division;
```

```
                c = '*';
                break;
        }
   cout << (a = (*par)(a,b))<<"\n"; // Ünvana görə çağirma
        }
}

Proqramın icrasının nəticəsi:
    Arguments: a = 6, b = 2. Result for c = '+':8
    Arguments: a = 8, b = 2. Result for c = '/':4
    Arguments: a = 4, b = 2. Result for c = '*':8
    Arguments: a = 8, b = 2. Result for c = '-':6
    Press any key to continue
```

Dövr c dəyişəninin qiymətinin poşluq olmasına qədər davam edir. Hər bir iterasiyada par göstəricisi funksiyalardan birinin qiymətini alır və c-nin qiyməti dəyişir. Proqram nəticəsində onun operatorlarının icra ardıcıllığını asanlıqla izləmək olar.

Funksiya göstəriciləri massivi

Funksiya göstəricisi massivdə də birləşdirilə bilər. Məsələn, float (*ptrArray[4]) (char); - 4 funksiya göstəricisi ehtiva edən ptrArray massivinin təyin edilməsi. Burada massivin hər bir elementi char tipində parametrə malikdir və float tipində qiymət qaytarır. Bu funksiyalardan, məsələn, üçüncüsünə müraciət etmək üçün növbəti operator tələb olunur:

```
float a = (*ptrArray[2])('f');
```

Bir qayda olaraq, massivin indeksləşdirilməsi 0-dan başlayır və massivin 3-cü elelmentinin qiyməti 2-dir.

Funksiya göstəriciləri massivini bütün mümkün menyuları olan, yəni menyu vasitəsilə idarə edilən proqram hazırlayarkən istifadə etmək rahatdır. Bunun üçün, gələcək istifadəciyə təqdim edilən proqram, ünvanları funksiya göstəriciləri massivində saxlanılan funksiya kimi tərtib edilir. İstifadəci ona təqdim edilən menyu bölmələrindən birini seçə bilər (sadə halda seçilən bölmənin nömrəsini daxil edə bilər). İndeks kimi bölmənin nömrəsinə görə massivdən uyğun funksiyanın ünvanı seçilir. Funksiya bu ünvana görə müraciət edilməsi tələb olunan işin icrasını təmin edir. Bu cür yanaşmanın reallaşdırılmasının ümumi sxeminin özünü növbəti "fayyaların emalı" proqramı əyani nümayiş etdirir:

```
#include <iostream>

using namespace std;

/* menyu hazırlamaq üçün funksiyanın təyini
   (funksiyalar real iş görmür):*/
void act1 (char* name)
{
    cout <<"Create file..." << name;
}

void act2 (char* name)
{
    cout << "Delete file... " << name;
}
void act3 (char* name)
{
        cout << "Read file... " << name;
}
```

```
void act4 (char* name)
{
    cout << "Mode file... " << name;
}
void act5 (char* name)
{
    cout << "Close file... " << name;
}
void main()
{
    // Göstəricilər massivinin yaradılması və
    qiymətləndirilməsi
    void (*MenuAct[5])(char*) =
                        {act1, act2, act3, act4, act5};
    int number; // Seçilən menyu pölməsinin nömrəsi.
    char FileName[30]; // Faylın adı üçün sətir
    //dəyişəni.
    // Menyunun reallaşdırılması
    cout << "\n 1 - Create";
    cout << "\n 2 - Delete";
    cout << "\n 3 - Read";
    cout << "\n 4 - Mode";
    cout << "\n 5 - Close";
    while (1)  // Sonsuz dövr.
    {   while (1)
        { /* Dövr düzgün nömrənin daxil edilməsinə
             kimi davam edir.*/
             cout << "\n\nSelect action:
             "; cin >> number;
             if (number>>= 1 && number <= 5)
                 break; cout << "\nError number!";
        }
        if (number != 5)
        {
             cout << "Enter file name: ";
```

```
            cin >> FileName;//Fayl adının daxil edilməsi.
          }
          else break;
    // Funksiyanın göstəricisi vasitəsilə çağırılması:
              // (*MenuAct[number-1])(FileName);
      }  // Sonsuz dövrün sonu.
}
```

Menyunun bölmələri 5 - bağlama nömrəsi daxil edilənə kimi təkrarlanacaq.

5. Ev tapşırığı

1. İstifadəçinin seçiminə görə ikiölçülü matrisin istənilən yerinə sətir və ya sütun əlavə edən proqram yazmalı.
2. MxN (M sətir, N sütun) ölçülü matri verilir. Onun elementlərinə qiymət verməli və massivin sətirlərini/sütunlarını verilmiş sayda, verilmiş istiqamətdə sürüşdürməli.