İNFORMATİKANIN TƏDRİSİ METODİKASI

UOT 007:378.147(075.8)

Leyla Elçin qızı Mehdiyeva
Gəncə Dövlət Universitetinin informatika müəllimi

C++ PROQRAMLAŞDIRMA DİLİNDƏ OBYEKTLƏRİN TƏSVİRİ

Лейла Эльчин гызы Мехдиева
преподаватель информатики
Гянджинского Государственного Университета

ОПИСАНИЕ ОБЪЕКТОВ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C++

Leyla Elchin Mehdiyeva
lecturer of Informatics at Ganja State University

DESCRIPTION OF OBJECTS IN C++ PROGRAMMING LANGUAGE

Xülasə. Məqalədə onsuz heç bir obyektyönlü proqram yazılmayan və proqramlaşdırmanın bünövrəsi hesab olunan obyektlərin təsviri üçün sintaktik qaydalara – siniflərə baxılmış, onun xüsusiyyətləri, metodları və növləri geniş şərh olunmuş, sinfin metodlarına istinad edən göstəricilərdən düzgün istifadə qaydaları verilmişdir. Qeyd edilmişdir ki, parametri abstarkt sinfə göstərici olan funksiya yaratmaq mümkündür. Proqramın icrası zamanı bu parametrin əvəzinə istənilən törəmə sinfin obyektinə başqa bir göstərici də vermək olar. Bu isə bir iyerarxiya daxilində istənilən tip obyektlərlə işləyən polimorf siniflər yaratmağa imkan verir.

Açar sözlər: *verilənlərin tipi, sinif, obyekt, sabit metod, sinfin sahələri, statik sahə*

Резюме. В статье рассмотрены синтаксические правила описания объектов – классов, без которых не пишется ни одна объектно-ориентированная программа и которые считаются фундаментом программирования, подробно описаны ее свойства, методы и виды, даны правила правильного использования указателей, ссылающихся на методы класса. Было отмечено, что можно создать функцию, параметр которой является указателем на класс abstarkt. Вместо этого параметра во время выполнения программы объекту любого производного класса также может быть присвоен другой указатель. Это позволяет создавать полиморфные классы, которые работают с объектами любого типа в иерархии.

Ключевые слова: *тип данных, класс, объект, фиксированный метод, поля класса, статическое поле*

Summary. In the article, syntactic regulations for the description of objects, without which no object – oriented program is written, and which are considered the fundamentals of programming, are considered-classes, its features, methods and types are widely interpreted, and the correct use of indicators referring to the methods of the class are given. It was noted that it is possible to create a function whose parameter is an indicator of the abstract class. During the execution of the program, this parameter can be replaced by another pointer to the object of any derivative class. This allows you to create polymorphic classes that work with any type of objects within a hierarchy.

Key words: *type of data, class, object, fixed method, class fields, static field.*

İstənilən proqramın məqsədi verilənləri emal etməkdir. Müxtəlif tip verilənlər müxtəlif şəkildə emal olunur və yaddaşda saxlanılırlar. İxtiyari alqoritmik dildə hər bir sabit, dəyişən, funksiya və ya ifadənin hesablanmış qiyməti müəyyən tipə məxsus olmalıdır.

Verilənlərin tipi aşağıda göstərilənləri təyin edir [1]:

- kompüterin yaddaşında verilənlərin daxili təsvirini;
- göstərilən tipdə dəyişənin ala biləcəyi qiymətlər çoxluğunu;

➢ göstərilən tipdə dəyişənə tətbiq edilə biləcək funksiya və əməliyyatları.

Bu xüsusiyyətlərə istinad edərək proqramçı real obyektlərin proqramda təsviri üçün hər bir dəyişənin tipini göstərməlidir. Əvvəlcədən tiplərin təsvir olunması kompilyatora proqramın müxtəlif konstruksiyalarının sintaksisini (düzgün yazılış formasını) yoxlamağa şərait yaradır. Qeyd edək ki, verilənlərin emalında istifadə olunacaq maşın kodları dəyişənlərin tipindən asılı olur.

C++ dilində verilənlərin tiplərini iki - əsas və mürəkkəb tiplərə ayırırlar. Tam, həqiqi, simvol və məntiqi dəyişənləri təsvir etmək üçün C++ proqramlaşdırma dilində 6 əsas tip müəyyən olunub. Bu tiplərə də istinad edərək proqramçı mürəkkəb tipləri proqramda göstərə bilər. Mürəkkəb tipli verilənlərə massivləri, köçürmələri, funksiyaları, strukturları, göstəriciləri, birləşmələri və sinifləri nümunə göstərmək olar.

C++ dilinin adı çəkiləndə yada düşən ilk anlayış bizim yeni daxil edəcəyimiz siniflər anlayışıdır. Burada qeyri-adi, yeni heç bir şey yoxdur. Sadəcə olaraq siniflərə *struct* tiplərin bir qədər fərqli və inkişaf etmiş növü kimi baxmaq olar. Belə ki, *struct* tipinin elementlərini tərtib edərkən biz standart tiplərdən və əvvəl yaratdığımız *struct* tiplərdən istifadə edirdik. Bu qayda siniflər üçün də keçərlidir.

Lakin siniflərin *struct* tiplərdən fundamental üstünlüyü ondadır ki, biz siniflərin tərkibinə nəinki hər hansısa tipdən olan dəyişən hətta funksiyaları da daxil edə bilərik. Bu imkan siniflərə çox fərqlilik verir və hal-hazırda C++ dilinin ən məşhur dillərdən biri olmasında müstəsna rol oynayır.

Sinif istifadəçi tərəfindən təyin olunan, verilənlərin abstrakt tipidir [4]. O özlüyündə real obyektin verilənlərinin və bu verilənlərlə işləmək üçün funksiyalar şəklində modelini nümayiş etdirir.

Sinfin verilənlərini sahələr (strukturun sahələrinə analoji olaraq), onun funksiyalarını isə metodlar adlandırırlar. Sahələr və metodlar sinfin elementləri adlandırırlar. Sinfin təsviri ilk görünüşdə aşağıdakı kimi olur:

```
class <ad> {
[private:]
<gizli elementlərin təsviri>
public:
<müraciət oluna bilən elementlərin təsviri>
}; //Təsvir nöqtəli vergüllə qurtarır
```

private və *public* müraciət spesifikatorları olub, sinfin elementlərinin görünməsini idarə edirlər. *private* xidmət sözündən sonra yazılmış elementlər yalnız sinfin daxilində görünən olurlar. Belə müraciət növü sinifdə susmaya görə qəbul edilir. Sinfin interfeysi *public* spesifikatorundan sonra təsvir olunur. İstənilən spesifikatorun fəaliyyəti növbəti spesifikatora və ya sinfin sonuna kimi davam edir. Proqramda *private* və *public* spesifikatorlarını bir neçə bölmədə vermək olar, bu zaman onların yazılma ardıcıllığı əhəmiyyət kəsb etmir.

Sinfin sahələri:

➢ sinfin tipindən başqa istənilən tip ola bilər (lakin bu sinfə istinad və ya göstərici kimi verilə bilərlər);

➢ *const* modifikatoru ilə təsvir edilə bilərlər, bu zaman yalnız bir dəfə inisiallaşdırılırlar (konstruktorun köməyilə) və dəyişdirilə bilməzlər;

➢ *static* modifikatoru ilə təsvir edilə bilərlər, lakin *auto, extern* və *register* kimi yox;

Sahələr təsvir edilən zaman qiymət ala bilməzlər.

Siniflər qlobal (blokdan xaricdə elan edildikdə) və lokal (blok daxilində elan edildikdə, məsələn, funksiya və və ya digər sinfin daxilində elan edildikdə) ola bilirlər.

Aşağıda lokal siniflərin bəzi xarakterik xüsusiyyətləri sadalanmışdır:

➢ lokal sinfin daxilində statik (*static*) və xarici (*extern*) dəyişənlərin, onun yazıldığı oblastın xarici funksiyalarının və elementlərinin tiplərindən istifadə etmək olar; bu oblastın avtomatik dəyişənlərindən istifadə qəti qadağandır;

➢ lokal sinif statik elementlərə malik ola bilməz;

➢ sinfin metodları yalnız həmin sinfin daxilində təsvir edilə bilərlər;

➢ əgər bir sinif digər sinfin daxilində verilibsə, onlar bir-birinin elementlərinə xüsusi müraciət hüququna malik deyillər və yalnız ümumi qaydalara görə elementlərə müraciət edə bilərlər.

Nümunə üçün kompüter oyununun personajının modelini verən bir sinif yaradaq. Bunun üçün əvvəlcə onun xüsusiyyətlərini (məsələn, qüvvələrin miqdarını, mərmilərin sayını) və davranışını göstərmək tələb olunur. (Qeyd edək ki, proqram nümunəsi yalnız sinfin yaradılması sintaksisini nümayiş etdirdiyi üçün sxematik verilir.)

```
class monstr {
int health, ammo;
public:
monstr(int he=100, int am=10) {health=he; ammo=am }
void draw(int x, int y, int scale, int position);
int get_health( ) {return health;}
int get_ammo( ) {return ammo;}
};
```

Bu sinif iki gizli sahəyə - *health* və *ammo* sahələrinə malikdir ki. bu sahələrdən də qiymətləri *get_health( )* və *get_ammo( )* metodları vasitəsilə almaq mümkündür. Bu nümunədə elə görünür ki, sahələrə metodlar vasitəsilə müraciət etməklə onu süni olaraq mürəkkəbləşdirmişik, lakin nəzərə almaq lazımdır ki, real siniflərin sahələri mürəkkəb dinamik struktura malik ola bilər və bu zaman sahə elementlərinin qiymətlərini almaq elə də asan olmur. Bundan əlavə, sinfin interfeysinə toxunmadan, onun bu strukturlarında dəyişiklik etmək imkanı ən mühüm haldır.

Sinfin bütün metodları onun gizli sahələrinə fasiləsiz müraciət etmək imkanına malik olurlar, başqa sözlə desək, sinfin funksiyalarının gövdələri sinfin *private* elementlərinin görünmə oblastına daxil olurlar [2].

Baxılan nümunədə üç metod təyin və bir metod (*draw* metodu) elan edilmişdir. Əgər metodun gövdəsi sinfin daxilində təyin edilmişdirsə, onda o quraşdırılmış (vstroennıy inline) funksiya adlanır. Bir qayda olaraq, qısa metodlar quraşdırılmış olurlar. Əgər sinfin daxilində metod yalnız elan (başlığı verilmişsə) edilmişsə, onda bu metod proqramın hər hansı bir hissəsində görünmə oblastına müraciət əməli (::) vasitəsilə hökmən təyin edilməlidir:

```
void monstr :: draw(int x, int y, int scale,
int position) {
/*metodun gövdəsi*/ }
```

Metodu sinfin daxilində *inline* direktivi vasitəsilə quraşdırılmış kimi də təyin etmək olar (adi funksiyaları belə təyin etmək məsləhətlidir):

```
inline int monstr :: get_ammo( ) {
return ammo;}
```

Hər bir sinifin daxilində həmin sinfin adı ilə üst-üstə düşən bir metod olur ki, bu da konstruktor adlanır və obyektin sinfini yaradan zaman o avtomatik olaraq çağırılır [3]. Konstruktor obyektin inisiallaşdırılması üçündür. Onun avtomatik çağırılması inisiallaşdırılmış dəyişənlərdən istifadə ilə bağlı səhvlərin baş verməməsinə zəmin yaradır.

Sinfin statik sahələrini və metodlarını *static* modifikatoru vasitəsilə təsvir etmək olar. Onlara yalnız sinfin daxilində müraciət edilə bilən qlobal dəyişənlər və ya funksiyalar kimi baxmaq olar. Statik sahələr, sinfin bütün obyektləri üçün ümumi olan verilənlərin saxlanması üçün istifadə edilir (məsələn, sinfin bütün obyektləri və ya bütün obyektlərin istifadə etdiyi resursa istinadlar). Belə sahələr sinfin bütün obyektləri üçün yeganə nüsxədə mövcud olur, yəni statik sahələrin nüsxələri alınmır.

Aşağıda statik sahələrin xüsusiyyətləri göstərilmişdir:

- Statik sahələr üçün yaradılan obyektlərin sayından asılı olmayaraq (hətta obyektlər olmasa belə) yalnız bir dəfə sahə inisiallaşdırılarkən yaddaş oblastı ayrılır. Statik sahələr fəaliyyət oblastına müraciət əməli vasitəsilə inisiallaşdırılırlar:

```
class A {
public :
 static int count; //Sinif daxilində elan
};
…
int A::count; //Qlobal oblastda sahə
susmaya görə sıfırla inisiallaşdırılır
// int A:: count=10; ixtiyari qiymətlə
inisiallaşdırılma
```

- Statik sahələrə müraciət sinfin adı və ya obyektin adı vasitəsilə edilir:

```
A*a, b ;
cout <<A::count<<a -> count <<b.count;
// eyni əməliyyat yerinə yetiriləcək
```

- Statik sahələrdə müraciət spesifikatorlarının fəaliyyəti mümkündür, odur ki, *private* kimi təsvir edilən statik sahələri yuxarıda göstərilən kimi, fəaliyyət oblastına müraciət əməli vasitəsilə dəyişdirmək olmaz. Bunu yalnız statik metodlar vasitəsilə etmək mümkündür.
- Statik sahəyə ayrılan yaddaş oblastı, *sizeof* əməli vasitəsilə obyektin yaddaşda tutduğu sahəni hesablayarkən nəzərə alınmır [1].

Statik metodlar sinfin statik sahələrinə müraciət etmək üçün vasitədir. Statik metodlar adətən yalnız statik sahələrə müraciət etdiklərindən və yalnız sinfin digər statik metodlarını çağıra bilmək imkanlarına malik olduqlarından

onlara *this* gizli göstəricisi verilmir. Statik metodlara statik sahələrdə olduğu kimi ya sinfin adı və ya da sinfin heç olmazsa bir obyekti yaradılmışsa, onda bu obyektin adı vasitəsilə müraciət edilir.

```
class A {
static int count; // count sahəsi gizlidir
public :
static void inc_ count( ) ; {count++;}
A::int count; //Qlobal oblastda daxilində elan
void f( ) {
A a; //a.count++ yazmaq olmaz, çünki count sahəsi gizlidir
//Sahənin statik metod vasitəsilə dəyişdirilməsi:
a.inc_count( ); // və ya A::inc_count( ); yazmaqla
}
```

Qeyd edək ki, statik metodlar sabit (const) və virtual (virtual) ola bilməzlər.

Əgər hər hansı bir sinfin bütün metodları digər sinfin gizli sahələrinə müraciət etmək imkanlarına malikdirlərsə, onda sinif bütövlükdə *friend* açar sözü ilə dost sinif elan edilir. Aşağıdakı nümunədə *mistress* sinfi *hero* sinfinə dost elan edilib:

```
class hero{
…
friend class mistress;
}
class mistress {
…
void f1 ( );
void f2 ( );
}
```

Burada *f1* və *f2* funksiyaları *friend* açar sözü ilə verilməsə də, *hero* sinfinə dost hesab olunurlar və bu sinfin bütün sahələrinə müraciət edə bilərlər. Qeyd edək ki, *friend* sözünün elan edilməsi müraciət spesifikatoru olmayıb, irsən ötürülmür.

Sinfin elementlərinə göstəricilər vasitəsilə müraciət etmək olar. Bunun üçün .* və ->* əməlləri təyin edilmişdir. Sinfin sahələrinə və metodlarına olan göstəricilər müxtəlif cür təyin edilirlər.

Sinfin metoduna olan göstəricinin yazılış formatı aşağıdakı kimidir:

```
qaytarılan_tip (sinfin_adı ::
*göstəricinin_adı) (parametrlər);
```

Məsələn, *monstr* sinfinin metodlarına olan göstəricilərin təsviri

```
int get_health ( ) {return health;}
int get_ammo ( ) {return ammo;}
```

eləcə də sinfin eyni strukturlu digər metodları aşağıdakı formatda yazılır:

```
int (monstr :: *pget ) ( );
```

Belə göstəricini funksiyanın parametri kimi də vermək olar ki, bu da metodun adını funksiyaya verməyə imkan verir:

```
void fun (int (monstr :: *pget ) ( )) {
(*this.*pget) ( ) ; // funksiyanın .* əməli ilə çağırılması
(this->*pget) ( ) ; // funksiyanın ->* əməli ilə çağırılması
}
```

Ünvanın götürülməsi metodu ilə də sinfin konkret metoduna göstərici vermək olar:

```
// Göstəriciyə qiymət mənimsədilməsi
pget= & monstr :: get_health;
monstr Təpəgöz, *p;
p=new monstr;
// funksiyanın .* əməli ilə çağrılması
int Təpəgöz_health=(Təpəgöz.*pget) ( );
// funksiyanın ->* əməli ilə çağrılması
int p_health=(p->*pget) ( );
```

Aşağıda sinfin metodlarına istinad edən göstəricilərdən düzgün istifadə qaydaları verilmişdir:

- metoda olan göstəriciyə yalnız uyğun başlıqları olan metodların ünvanını mənimsətmək olar;
- sinfin statik metoduna göstərici təyin etmək olmaz;
- metoda olan göstəricini, sinfin elementi olmayan adi funksiyaya olan göstərici ilə dəyişdirmək olmaz.

Adi funksiyaya olan göstəricilər kimi metodlara olan göstəricilər də adı məlum olmayan metodu çağırmaq zəruriyyəti meydana çıxdıqda istifadə edilirlər. Lakin adi funksiyaya və ya dəyişənə olan göstəricilərdən fərqli olaraq, metod olan göstərici konkret yaddaş ünvanına istinad etmirlər. Yerləşdirmə verdiyi üçün o daha çox massivin indekslərini xatırladır. Konkret yaddaş ünvanı müəyyən obyektə istinad edən göstərici ilə metoda istinad edən göstəricinin uyğunluğundan alınır. Qeyd edək ki, göstərici ilə çağırılan metodlar virtual ola bilərlər. Bu zaman ob-

yektin tipinə uyğun göstəricinin istinad etdiyi metod çağırılır.

Sinfin sahələrinə olan göstəricinin yazılış formatı aşağıdakı kimidir:

verilənlərin_tipi (sinfin_adı :: *göstəricinin_adı);

Göstəricini təyin edərkən onun inisiallaşdırılmasını aşağıdakı formada daxil etmək olar:

&sinfin_adı :: *sahənin_adı; // Sahə public olmalıdır

Əgər *health* sahəsi *public* kimi elan olunsa, onda ona istinad edən göstərici aşağıdakı kimi yazılar:

int (monstr :: *phealth) =& monstr :: health;

cout <<Təpəgöz.*phealth ; // .* əməli ilə müraciət

cout << p->*phealth ; // ->* əməli ilə müraciət

Göründüyü kimi sinfin sahələrinə olan göstəricilər adi göstəricilər kimi deyillər, onlara qiymət mənimsədildikdə konkret yaddaş ünvanına istinad etmirlər (buna səbəb yaddaş sahəsinin sinif üçün deyil, onun obyektləri üçün ayrılmasıdır).

Sinif yaradılarkən faydalı məsləhətlər: Bir qayda olaraq məlumdur ki, sinif istifadəçi tərəfindən təyin edilən tipdir və o gizli sahələrə (*private*) və aşağıdakı funksiyalara malik olmalıdır:

- *Konstruktorlar* – sinfin obyektlərini inisiallaşdırmaq üçün təyin edilirlər;
- Sinfin xüsusiyyətlərini göstərən *metodlar yığım*ı (sinfin gizli sahələrinin qiymətlərini qaytaran metodlar const modifikatoru ilə yazılmalıdırlar. Bu onu göstərir ki, metodlar sahələrin qiymətini dəyişməməlidirlər);
- Sinfin təyinatından asılı olaraq onun obyektləri üçün nüsxələrinin alınması, mənimsətmə, müqayisə etmə, yaratma və s. bu kimi əməliyyatların icrası üçün *əməllər yığımı*;

Əgər sinfin interfeysi vasitəsilə (yəni, funksiyaların işləməsi üçün sinfin gizli sahələrinə müraciət tələb edilmədikdə) bir və ya bir neçə siniflə işləyən funksiyalar mövcuddursa, interfeys çox məlumat saxlamasın deyə onları sinifdən xaricdə təsvir etmək olar. Məsələn:

```
namespace Staff {
class monstr {/* … */};
class hero {/* …*/};
void interact (hero, monstr);
… }
```

Heç olmasa bir virtual metoda malik olan sinif *abstrakt sinif* adlanır [2]. Abstrakt siniflər törəmə siniflərdə mövcudluğu ehtimal edilən ümumi anlayışları konkretləşdirmək üçün nəzərdə tutulub. Abstrakt siniflər digər siniflər tərəfindən yalnız baza sinfi kimi istifadə oluna bilirlər, onların obyektlərini yaratmaq olmaz, çünki sinfin istənilən təmiz virtual metodunun birbaşa və ya dolayı yolla çağırılması proqramın icrasında səhvliklər törədir.

Abstrakt sinifləri təyin edən zaman aşağıdakıları nəzərə almaq lazımdır:

- Abstrakt sinifdən, parametrin tipini və funksiyanın qaytardığı qiymətin tipini təsvir edərkən aşkar şəkildə tiplərin verilməsində istifadə etmək olmaz;
- Əgər inisiallaşdırma zamanı müvəqqəti obyekt yaratmaq tələb olunmursa, onda abstrakt siniflərə göstərici və istinadları elan etməyə icazə verilir;
- Əgər abstrakt sinfin övladı olan sinifdə bütün təmiz virtual funksiyalar təyin edilməyibsə, o da abstrakt sinif hesab edilir.

Beləliklə, parametri abstarkt sinfə göstərici olan funksiya yaratmaq mümkündür. Proqramın icrası zamanı bu parametrin əvəzinə istənilən törəmə sinfin obyektinə başqa bir göstərici də vermək olar. Bu isə bir iyerarxiya daxilində istənilən tip obyektlərlə işləyən polimorf siniflər yaratmağa imkan verir.

Problemin aktuallığı: obyektyönlü proqramlaşdırma proqramlaşdırma sistemlərində ən əsas və vacib istiqamətlərdən biri hesab olunur. Bu sahənin inkişaf etdirilməsi və onun imkanlarından istifadə edərək yaradılan hər bir proqram məhsulu informasiya cəmiyyətinin mənafeyinə xidmət edir. Məqalədə bu istiqamətdə məsələlərə toxunulması onun aktuallığını bəyan edir.

Problemin yeniliyi: sinif və abstrakt siniflər yaradılarkən faydalı məsləhətlərin verilməsi, lokal siniflərin və sinfin statik sahələrinin bəzi xarakterik xüsusiyyətləri sadalanması və praktik nümunələrlə verilməsidir.

Problemin praktik əhəmiyyəti: məqalədə geniş şərh edilən C++ dilində istifadə olunan sinif tipi, onun xüsusiyyətləri və imkanları və istifadəsi qaydaları haqqında alınmış qənaətlərdən, bilik və bacarıqlardan, praktik nümunələrdən ali və orta ixtisas müəllimləri bəhrələnərək laborator tapşırıqların hazırlanmasında, eləcə də proqramlaşdırma sahəsində biliyə malik hər bir şəxs yeni proqram utilitlərinin yaradılmasında istifadə edə bilər.

Ədəbiyyat:

1. Страуструп, Б. Программирование: принципы и практика использования C++ . Москва-Санкт-Петербург -Киев: Вильямс, - 2013
2. Həsənova, M.A. C++ proqramlaşdırma dili. -Gəncə, -2014.
3. Pələngov, Ə. Həsənova, M. C++ və Java proqramlaşdırma dilləri: Dərs vəsaiti. -Bakı, -2019.
4. Ишкова, Э.А. C++ начала программирования. Учебник для вузов. -Москва: Бином-Пресс, - 2009.
5. Шилдт, Г. C++: методики программирования Шилдта. Москва-Санкт-Петербург-Киев: Вильямс, -2009.

E-mail: n.leyla95@mail.ru
Rəyçilər: *ped.elm.dok. prof.* **Ə.Q. Pələngov**
tex.ü.fəls.dok. dos. **M.A. Həsənova**
Redaksiyaya daxil olub: 24.09.2021.