Az összetettebb feladatok megoldásához gyakran kell bevezetnünk olyan adattípusokat, amelyek nem szerepelnek a C++ alaptípusai között. Az ilyen úgynevezett *felhasználói típusok* értékeinek számítógépes ábrázolásáról, azaz reprezentálásáról, valamint a típus műveleteinek implementálásáról is nekünk kell gondoskodni. Egy típus orientált programban egy vagy több ilyen felhasználói típusú objektum (változó) található, a program pedig a típusok műveleteit hajtja végre egy vagy több adott objektumra.

Felhasználói típusok megvalósítása

Vannak olyan felhasználói típusok, amelyek nevezetes szerkezettel rendelkeznek (például tömb vagy rekord), és a típus műveletei ezen a szerkezeteken megszokott standard műveletek. Ilyenkor a típus megvalósítására a minden magas szintű nyelvben megtalálható típuskonstrukciós eszközöket használjuk. A tömbökről korábban már részletesen beszéltünk. Ezek rögzített hosszúságú sorozatok, amelyek elemeire az indexük segítségével hivatkozhatunk. A rekord szerkezetű típusok a `struct` szerkezettel írhatóak le, és az ilyen típusú változókra (objektumokra) érvényes a mezők (szelektorok) szerinti hivatkozás művelete. Jól használható az `enum` szerkezet egy olyan (véges sok értékkel rendelkező) típus értékeinek felsorolására, amely az értékek összehasonlításain kívül nem rendelkezik más típusművelettel. Az `union` szerkezet csak első látásra tűnik alkalmas eszköznek az unió (alternatív) szerkezetű típusok definiálására (de nem rendelkezik olyan művelettel, amely az unió szerkezetű változóban éppen tárolt érték típusát megmutatná), ezért ezt nem fogjuk használni.

Azokat a felhasználói típusokat, amelyek a fenti eszközökkel nem tudunk definiálni, elsősorban azért, mert egyedi típusműveletekkel rendelkeznek, osztályokkal (`class`) valósítjuk meg. Egy osztályban megadható a típusértékek reprezentációja (adattagok) és a típus értékeivel végezhető műveletek (metódusok).

A felhasználói típusokat megvalósító osztályokat a reprezentációjuk módja szerint két részre osztjuk. Azokat, amelyek reprezentációjánál nem használunk dinamikus elemeket (nem szerepel * a reprezentációban, nem kell az adattagokat `new` utasítással létrehozni), *egyszerű osztályoknak* nevezzük.

Absztrakt típus megvalósítása osztállyal

A típus-megvalósítás eszköze az osztály. Egy adott típus változója (azaz a típust leíró osztály objektuma) a típus lehetséges értékeit, a típusértékeket veheti fel. Abban a kódrészben, ahol ilyen változóval dolgozunk, nem kell, nem kell mást tudni a típusról, mint azt, hogy melyek a lehetséges értékei és ezekkel milyen műveleteket lehet elvégezni.

A típusműveleteket (metódusokat) az osztály publikus (`public`) részében deklaráljuk. Ezek olyan speciális függvények, amelyeknek legalább egy paramétere mindig van: az a változó (objektum), amelyben tárolt típusértékre a metódust alkalmazni akarjuk. Éppen ezért nem is kell feltüntetni ezt a paramétert a metódus deklarálásánál, a metódus hívásánál pedig a változó (objektum) nevét, amire a művelet vonatkozik, a metódus neve elé kell írni, és egy ponttal kell attól elválasztani. A metódus többi paraméterét – ha vannak ilyenek – a szokásos módon, a metódus neve utáni zárójelben kell felsorolni.

Egy adott típusú változó (osztály-objektum) használatakor nem szabad arra gondolni, hogy egy típusértéket milyen formában ábrázoljuk a számítógépen. Az osztályban a típusértéket reprezentáló adattagokat ezért célszerű elrejteni azon kódrész elől, ahol az osztály objektumaival dolgozunk. Ehhez az adattagokat privát (`private`) minősítéssel látjuk el. Az

privát részben definiált elemek hatásköre csak az adott osztályra, illetve az osztály metódusainak definícióira terjed ki, máshol használni őket, hivatkozni rájuk nem lehet. Privát nemcsak adattag, hanem olyan metódus is lehet, amelyet csak az osztály metódusai hívhatnak meg.

A metódusok teljes definícióját (ezek a típusműveletek implementációi) általában az osztályon kívül adjuk meg, az osztáűlyban csak a metódus deklarációja (szignatúrája) szerepel. Indokolt (rövid, legfeljebb egy-két értékadást tartalmazó metódustörzs) esetben használhatunk úgynevezett implicit (inline) definíciót.

Egy osztály tartalmazhat típus-definíciókat is, így egy osztályba elhelyezhetünk egy másik osztályt is. Ez utóbbi lehetősséggel egyelőre nem élünk.

Típus orientált program szerkezete

Az objektum-orientált programban a felhasználói típusokat a program többi részétől elkülönítve, egy-egy osztályban definiáljuk. Egy forrás-állományú programkód esetén az osztályok definíciói megelőzik a `main()` függvényt, valamint a `main()` függvény előtti függvény-deklarációkat, a metódusaik definíciói pedig az egyéb függvény-definíciókkal együtt a `main()` függvény után helyezkednek el.

Nagyobb programok esetén egy-egy felhasználói típust külön állományban helyezünk el. Ilyenkor is egy fej- és forrásállomány párt használunk: a `h` kiterjesztésű, például `tipus.h` fejállomány az osztály definícióját tartalmazza, a `cpp` kiterjesztésű, például `tipus.cpp` forrásállomány pedig az osztály metódusainak definícióit. A fejállományt mind a `tipus.cpp` forrásállományba, mind az összes olyan forrásállományba, ahol az adott típust használni akarjuk, be kell „inklúdolni”.

Kódolási megállapodások

1. A megvalósítandó típus műveleteit az osztály publikus részében, metódusként deklaráljuk.
2. Az osztály privát részében a típus egy értékét reprezentáló adattagokat adjuk meg.
3. A privát részében deklaráljuk azokat a segédfüggvényeket is, amelyeket kizárólag az osztály metódusai használnak.
4. Legalább egy konstruktort mindig definiáljunk! A konstruktor feladata a reprezentáló elemek feltöltése olyan kezdeti értékekkel, amelyek kielégítik a típusinvariánst.
5. Egyszerű osztályban akkor definiálunk destruktort, amikor a konstruktorban olyan inicializáló utasítást használtunk, amelynek van befejező párja. (Például fájloknál az open-close utasításpár.)