Adatbázis és szoftverfejlesztés elmélet

Témakör 4.

Összefoglalás

1. A kódolás eszközei

- Általános szövegszerkesztő
- Programozói szövegszerkesztő
- Fejlesztői környezet
- Vizuális fejlesztői környezet

Általános szövegszerkesztő
Az általános (clear text) szövegszerkesztők bármely nyelven való programozáshoz használhatók. Ilyenek lehetnek Windowson a Jegyzettömb, Notepad++, stb. Linuxon mcedit, nano, gedit, stb.

Programozói szövegszerkesztő
Általában olyan szövegszerkesztő, amelyet kifejezetten programozók számára készítettek. Sok olyan funkciót találunk benne, amely megkönnyíti a programok írását. Több platformos, sok nyelvet támogató ilyen szövegszerkesztő például a Scite. A Scite az útvonalba lévő fordítókat, interpretereket felismeri, így azok azonnal használhatók. A programok fordítás után futtathatók.

Fejlesztői környezet
Általában valamely programozási nyelvhez tartozó olyan program, amely a programozói munkát a lehető legkönnyebbé igyekszik tenni. Általában tartalmaz hibakövető eszközöket. A fordítás és a futtatás néhány kattintással történik. CodeBlocks (C, C++, D, de bármely nyelv integrálható) Dev-C++ FreePascal IDE, Eclipse stb.

Vizuális fejlesztői környezet
A program felületét vizuális eszközökkel tudjuk megtervezni, a felület egyes elemeihez eseményeket tudunk rendelni. CodeBlocks (wxWidget projektel), MS Visual Studio, Delphi, Lazarus, NetBeans, Eclipse+plugin stb.

2. Programkészítési technikák

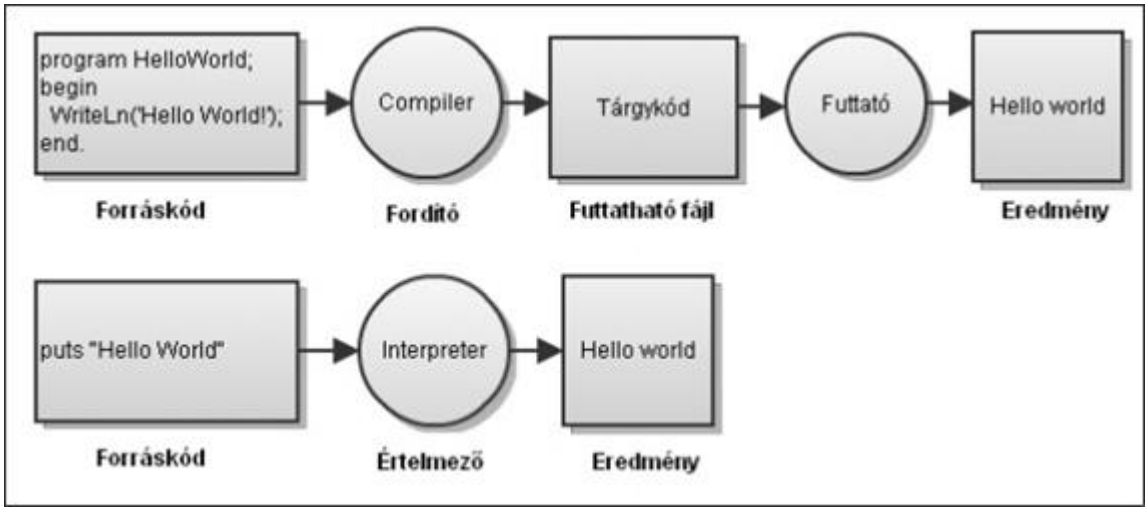
Fordító
Egy program készítése és használata három szakaszra osztható
1. A programot megfogalmazzuk egy programozási nyelven (Ez a forráskód)
2. Gépikódú utasításra lefordítjuk és futtathatóvá szerkesztjük.
3. A program futtatását az operációs rendszer végzi. Az ilyen programot szokás natív kódnak is nevezni.

Értelmező
Az implementált programozási nyelvek egy részéhez nem fordítót készítenek, hanem értelmezőt (interpreter-t). Az értelmező a programot csak a futtatással egy időben fordítja le gépkódú utasítások sorozatára. Így magát a forráskódot futtatjuk közvetlenül. Az így előállított program - természetéből adódóan - lassabban fut, mivel előbb le kell azt fordítani. A program futtatásához kell egy értelmező szoftver az operációs rendszerre.

Bájtkód
A bájtkód előállításánál a fordítós és az értelmezős technika együtt kerül alkalmazásra. A forráskódot lefordítjuk, de nem gépikódra. Egy **köztes kódra** fordítjuk, melynek neve **bájtkód**. Ha futtatni akarjuk előbb az értelmezővel fordítjuk gépikódra, ami elindul. Ez elvileg gyorsabb mint a szimpla értelmezős technika, mivel az eredeti forrást már előfordítottuk.

3. A fordítás lépései

A fordítás jellemzően nem egy menetben történik. Először egy úgy nevezett tárgykód (object file) készül el a forrásállomány(ok)ból. Ez Windowsos rendszereken „.obj”, Unix alapú rendszereken „.o” kiterjesztésű fájlokat jelent. Ez után egy szerkesztő állítja össze a futtatható állományt. Ennek haszna főleg több forrásfájlnál mutatkozik meg. A fordításhoz tehát két programra (programmodulra) van szükség: fordítóra és szerkesztőre. A fejlesztői környezetek, és ma már a legtöbb fordító parancs ezt a két műveletet egyben megcsinálja.



4. Az értékadás

Az értékadó utasítás

Az értékadó utasítás formailag *„azonosító = kifejezés”* alakú.

Ennek segítségével állíthatjuk be, hogy az „azonosító” milyen értéket képviseljen a programunkban ezen időpillanattól kezdve. Az azonosító általában valamilyen változó neve, de ezen értékadó utasítással állíthatjuk be a konstansok értékét is.

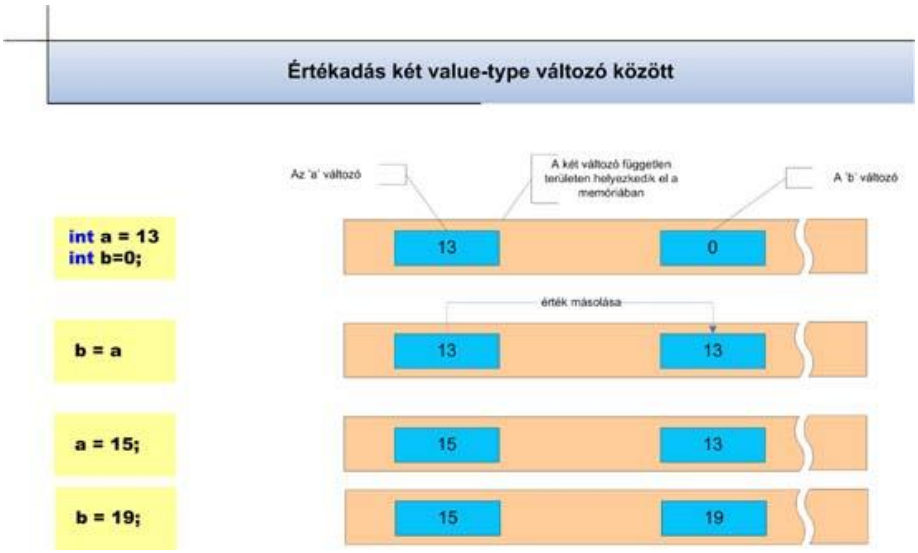
A kifejezés lehet egy kódba írt literális konstans, lehet egy másik azonosítóval képviselt érték, vagy - sok esetben - operátorokból és operandusokból épül fel. Az operátorok az adott operandusokon értelmezett műveleteket jelölnek, végrehajtásuk után egy-egy újabb értéket állítanak elő. Az értékek típusát az adott operátor és operandusok ismeretében meg lehet határozni. A kifejezésnek ennek megfelelően mindig levezethető a típusa. Ezt a típust a kifejezés típusának nevezzük.

A szabály szerint egy értékadó utasításban a kifejezés típusának vagy egyformának kell lennie az azonosító által képviselt típussal, vagy léteznie kell implicit típuskonverziónak a „kifejezés típusa -> azonosító típusa” irányában.

Értékadás érték (egyszerű) típusú változók között

Az érték típusú változókhoz tartozó memóriaterületen (valahány bájtnyi terület) az adott változók értékei tárolódnak. Két ilyen változó közötti értékadás során e bájtok másolódnak át egyik területről a másikra. Az átmásolás után a két változó a továbbiakban nem áll kapcsolatban egymással. Pl.:
double a =12.3; // az ’a’ memóriaterületének feltöltése

double b = a; // 12.3 átmásolása a ’b’ területére



Értékadás referencia típusú változók között

A referencia típusú változók elsődlegesen egy memóriacímet tárolnak. A változó tényleges, típusának megfelelő értéke ezen a memóriacímen található. Ilyen típus a karakterlánc (string) típus is.

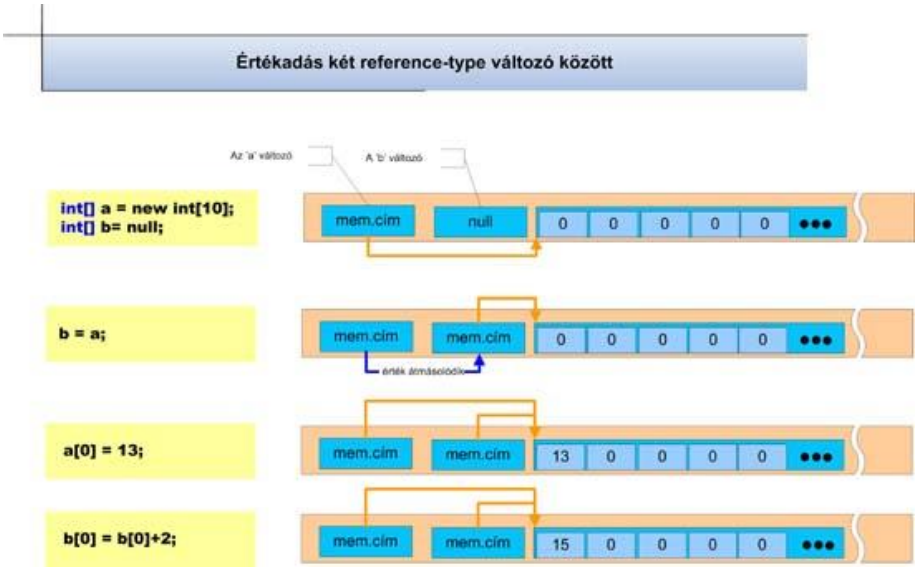
String s = ”almafa”;

… esetén a memóriában eltárolásra kerül valahol az ”almafa” szöveg (14 byte-on), majd az s változóba bekerül ezen memóriaterület címe.

String m = s;

Két referencia típusú változó közötti értékadás során nem a típushoz tartozó érték kerül lemásolásra (duplikálás), hanem csak a memóriacímek másolódnak le.
A fenti esetben az m változóba átmásolódik az s-ben tárolt memóriacím. Így e pillanattól kezdve az m változó is az ”almafa” szövegre mutat, az m változó értéke is az ”almafa” szöveg lesz.

Ennek az az előnye, hogy egy memóriacím pl. 4 byte-os adat, melynek lemásolása nagyon gyorsan végrehajtható. Ugyanakkor egy String érték akár több ezer karakter is lehet, melynek lemásolása viszont lassú és időigényes, nem is beszélve a memóriapocsékolásról.



5. A típuskonverzió

Olyan művelet, mely során egy kifejezés értékének típusát megváltoztatjuk.

A programozási nyelvek többségében ismert az automatikus (implicit) típuskonverzió, amikor egy kifejezés kiértékelése közben a különböző reprezentációjú adatok típusegyeztetése – beavatkozás nélkül - megtörténik. Továbbá létezik a különböző standard eljárások és függvények segítségével elvégzett ún. kikényszerített (explicit) típuskonverzió.

Implicit típuskonverzió

A fordítóprogram pontosan ismeri minden értéknek a típusát, és azzal is tisztában van, hogy melyik típusra kellene megváltoztatni azt. Erre szükség lehet egy értékadó utasítás végrehajtása során a bal és a jobb oldal típusának egyeztetésekor, vagy esetleg valamely operátor végrehajtása előtt a két operandus típusait kell közös típusra hozni. Az implicit típuskonverzióról akkor beszélünk, amikor a típusát-alakítást a fordítóprogram önállóan veszi észre és hajtja végre. Az implicit típuskonverzió során az új típusnak a neve nem szerepel az adott ponton a program szövegében, nincs „látható” nyoma.

Explicit típuskonverzió

Ha egy kifejezés típusa nem megfelelő a számunkra, direkt módon is előírhatjuk a konverziót. Az explicit típuskonverzió során egy-egy kifejezésben szereplő érték típusát megváltoztatjuk. Ennek során az értéket képviselő operandus elé az új típus nevét leírjuk zárójelben:

```
int a = 12;
double b = (double)a*2;
```

A fenti példában a ’12’ értéket double-ra változtattuk. Nem minden típusú értékből lehet explicit

módon valamely más típusú értéket készíteni. Például egy karakterlánc „cseresznye” értéket értelmetlen dolog valós típussá alakítani, azaz explicit módon sem lehet azt konvertálni.

Kérdések:

1. Sorolja fel a kódolás eszközeit!
2. Mi jellemzi a programozói szövegszerkesztő programokat?
3. Mi jellemzi a fejlesztői környezeteket, beleértve a vizuálisakat is?
4. Hogyan áll elő a futtatható program fordító program használata esetén?
5. Hogyan működik a bájtkódos programkészítés?
6. Hogyan fut egy értelmezőhöz készített program?
7. Hogyan épül fel egy értékadó utasítás? Mi jellemzi a két oldalát?
8. Mutassa be az értékadás folyamatát az egyszerű értéktípusú adatok esetében!
9. Ismertesse az értékadás mikéntjét referencia típusú változók esetében!
10. Mi a típuskonverzió?
11. Mikor beszélünk implicit típuskonverzióról?
12. Mit jelent az explicit típuskonverzió?