PROGRAMOZÁSI NYELVEK ÖSSZEHASONLÍTÓ ELEMZÉSE*

Nyékyné Gaizler Judit, *nyeky@ludens.elte.hu*
Eötvös Loránd Tudományegyetem
Általános Számítástudományi Tanszék

Abstract

The course 'Programming Languages 3' takes place in the education process of software experts at the University Eötvös Loránd for the students who are interested in programming languages. It should help the students understand the principles that underlie all languages. It gives an overview of the possible components of languages, and compares the tools of various languages supporting procedural, iterational and data abstraction. The comparison is based on the languages Ada, Modula, Pascal, C, FORTRAN, COBOL, PL/1, C++, Eiffel, Lisp, Prolog, Miranda, Simula, CLU, Alphard, Smalltalk, Objective C, Turbo Pascal, Oberon, Delphi, Simula, Trellis, Beta, POOL and Java.

1. Bevezetés

Az ELTE programtervezõ matematikus képzésében a hallgatók a negyedik évben specializálódhatnak: számos informatikai és matematikai témából választanak négyet az érdeklõdési körüknek megfelelõen. Ezen témakörök egyike a Programozási nyelvek és automaták sáv, amely számos kutatási és alkalmazási területet foglal magába. Szükségesnek láttuk, hogy oktatásunkban a jelentõsebb témák valamilyen formában megjelenjenek, így a sáv kötelezõ és szabadon választható speciális elõadásokból ill. szemináriumokból áll. A kötelezõ "mag" egy-egy elõadás az automataelmélet, a programozási nyelvek szemantikája és a programozási nyelvek témakörébõl.

A szabadon választható témákból a hallgatóknak öt tárgyat kell felvenniük és teljesíteniük. A speciálkollégiumok helye a tanrendben nem kötött, azt egy hallgató bármikor felveheti, ha a tárgy elõfeltételeinek megfelel. A választékot (jelenleg Formális nyelvek és automaták szeminárium, Sztochasztikus automaták, Formális szemantika szeminárium, Formális szemantika 2., Programozási nyelvek szeminárium, az Eiffel ill. a C++ programozási nyelv, Funkcionális programozás, Mesterséges intelligencia nyelvek) terveink szerint folyamatosan aktualizáljuk, hogy így a hallgatóknak áttekintési lehetõséget adjunk a legújabb fejlõdési irányzatokról. Az elõzõeken kívül meghirdetünk speciális elõadásokat pl. a Java, Delphi, SuperPascal stb. programozási nyelvekrõl is.

2. Az oktatás célja

A Programozási nyelvek 3. tárgy a téma "záróféléve", a hallgatók ekkor már számos programozási nyelvet ismernek, így látniuk kell, hogy az egyes nyelvek nemcsak egyszerûen jelölésformát adnak az algoritmus leírásához, hanem nyelvi eszközeikkel támogatják a program bonyolultságának kezelését is. Az **oktatási cél** itt a különbözõ magasszintû programozási nyelvek közös nyelvi elemeinek összehasonlító elemzése, és a jelen fejlõdési irányainak áttekintése - egy esetleges majdani nyelvtervezõ szemével is.

3. A tematika

A tárgy tematikájának kialakításánál építettünk az irodalomban hozzáférhetõ [pl. 1,2,3,4] tapasztalatokra, ugyanakkor figyelembe vettük, hogy melyek azok a nyelvek, amelyeket a programozó matematikus hallgatók az elsõ lépcsõben kötelezõen elsajátítanak (Ada, Modula-2, C, Pascal, FORTRAN,

COBOL, PL/1), ill. melyek azok, amelyek megismeréséhez a sáv speciálkollégiumai nyújtanak segítséget (C++, Eiffel, Lisp, Prolog, Miranda, Simula, Modula-3, CLU, Alphard, Smalltalk, Objective C, Turbo Pascal, Delphi, Simula, Trellis, Beta, POOL, Java stb.). Ez utóbbiakkal önállóan is megismerkedhetnek a hallgatók, ezt részben az irodalom megadásával [5-31] részben a Programozási nyelvek labor Augusta szerverén elhelyezett, a hallgatók által gondozott, folyamatosan bõvülõ nyelvleírásokkal (http://augusta.elte.hu/) segítjük.

A Programozási nyelvek tárgy felépítése a következõ:

1. Tudatosítjuk a hallgatókban, hogy elsõdleges célunk a jó minõségû programtermék létrehozása, ezt szolgálják a különbözõ programozás módszertani megfontolások, s ezeket támogatják a különbözõ, konkrét nyelvi eszközök. Ahogy helytelen a módszertant egy-egy konkrét nyelven keresztül tanítani, hasonlóan zsákutcába vezet, ha azt mondjuk, hogy a használt programozási nyelv nem is fontos a módszertan szempontjából. Ez olyan - ahogy B. Meyer írja -, mint a madár szárnyak nélkül. A gondolat nem szeparálható el a kifejezés lehetõségeitõl. Nem véletlen, hogy a programozásban egy nyelv sem lett egyeduralkodóvá, ahogy az sem, hogy egyre újabb és újabb programozási nyelveket terveznek, amelyek eszközeikkel egyre jobban támogatják a különbözõ módszertani elgondolások, követelmények gyakorlatba való átültetését.

2. Áttekintjük a programozási nyelvek fejlõdését, jellemezzük lehetséges osztályaikat - interaktív (pl. Basic, APL, dBase, Miranda, Lisp), strukturált (pl. Pascal, C, Ada) , szigorúan típusos (pl. Pascal, Ada, ANSI C, Miranda), objektum orientált (pl. Simula, Smalltalk, C++, Eiffel, Ada95), procedurális (pl. C, Pascal, FORTRAN, Basic, COBOL, Ada), funkcionális (Lisp, Miranda), párhuzamos (pl. Ada, LINDA, POOL), rendszerprogramozási (pl. C, FORTH), üzleti célú adatkezelõ (pl. COBOL, RPG, Ada), adatbázis-kezelõ (pl. dBase, SQL, Supernova), listakezelõ (pl. Lisp, Miranda), logikai (pl. Prolog), tömbkezelõ (pl. APL), szövegkezelõ (pl. SNOBOL), kiadványszerkesztõ (pl. TeX, LaTeX), parancs (pl. Perl) és negyedik generációs nyelvek (pl Supernova).

3. A nyelvi elemek tárgyalása három fõ témakört ölel fel: az eljárások, a vezérlés és az adatabsztrakció megvalósítási lehetõségeit.

3.1. Az eljárási absztrakciók megvalósításánál megvizsgáljuk, hogy a különbözõ nyelvekben az eljárások és függvények lehetnek-e önálló fordítási egységek (procedurális szemlélet) - vagy kizárólag absztrakt adattípusok mûveletei írhatóak le velük (objektum orientált szemlélet). Sorra vesszük a specifikáció megadásának lehetséges módjait, a paraméter átadás-átvételi módokat, programegységek egymásbaágyazhatóságának, globális és lokális változók használatának kérdéseit. Összehasonlítjuk a funkcionális és imperatív programozási nyelveket.

3.2. A vezérlési szerkezetek tárgyalása a lehetséges utasítástípusok elemzésével kezdõdik. Áttekintjük a utasítássorozatok, valamint a feltételes és az ismétlõdõ végrehajtás támogatására adott nyelvi elemeket, külön foglalkozunk az iterátorokkal. Elemezzük a nyelvek kivételes és hibás helyzeteket kezelõ eszközeit, összehasonlítva pl. az Ada, a CLU, az Eiffel, a C++ exception kezelési politikáját. Vizsgáljuk a párhuzamos - esetleg fizikailag is több processzoron futó - végrehajtás támogatását, a kommunikáció különbözõ formáinak megvalósítását az egyes programozási nyelvekben.

3.3. A beépített adattípusok, típuskonstrukciós eszközök (pl. rekord, tömb) áttekintése után az absztrakt adattípus megvalósítását, az objektum orientált programozást, és a helyességbizonyítást támogató nyelvi elemek tárgyalása külön hangsúlyt kap.

Absztrakt adattípusok létrehozásánál szigorúan különválasztjuk a mindenki által elérhetõ *típusspecifikáció*t - ahol leírjuk a típusértékhalmaz nevét, esetleges invariánsát, és a típusmûveletek specifikációját -, a típusértékhalmaz rejtett *reprezentációjá*t, és a típusmûveletek specifikációjától valamint a választott reprezentációtól függõ, szintén rejtett *implementációjá*t.

Az egyes nyelveket minõsíti, hogy adnak-e, és milyen szintû támogatást **absztrakt adattípus** létrehozására? Mindenki által egyformán látható-e az interface, mint pl. a Modula-2-ben, az Adában stb., vagy van lehetõség ún. szelektív láthatóság megadására, ahol a típus tervezõje az egyes mûveletekrõl döntheti el, hogy kik láthatják, ahogy pl. az Eiffel teszi, milyen következményekkel jár, ha a nyelv bizonyos speciális esetekben lehetõséget ad a rejtett reprezentációhoz való hozzáférésre, ahogy például a friend a C++-ban? **Külön fordítható egység**-e egy absztrakt adattípus, mint pl. a CLU-ban a cluster, az Eiffelben a class, vagy egy package-be ill. modulba kell 'becsomagoljuk', mint az Adában ill. a Modulában? Írhatunk-e **típusinvariánst, elõ- és utófeltételeket** a mûveletek specifikációjába, mint az Eiffelben, vagy az Alphardban? Lehet-e az adott típusú objektum létrehozásakor a típusinvariánst **konstruktorok** segítségével beállítani? Az **operátorok túlterhelésével** a saját adattípusaink használatát a beépítettekéhez teljesen hasonlóvá tehetjük-e, mint a C++-ban, Eiffelben vagy az Adában? Van-e lehetõség **típussal paraméterezett** absztrakt adattípus-minták létrehozására, amelyeket a konkrét felhasználáskor példányosítunk (pl. Ada, CLU, Eiffel generic, C++ template stb.)? Hogy kezeli a nyelv ekkor a törzsben használható mûveleteket?

Az objektum orientált programozási módszert a nyelvek tervezõi szempontjából felfoghatjuk úgy, mint igényt az absztrakt adattípusok megvalósításán kívül az öröklõdés és a polimorfizmus, valamint a dinamikus kötés támogatására. A már létezõ nyelvek nagy részében (pl. Turbo Pascal, Modula-2, Ada95) új nyelvi elemeket vezettek be erre a célra, sõt bizonyos esetekben - pl. C++ - így már ténylegesen egy új nyelv is keletkezett. Érdemes megjegyezni, hogy ezek a nyelvi kiterjesztések általában jól megkülönböztethetõek a kifejezetten objektum orientált programozás támogatására tervezett nyelvektõl - pl. Smalltalk, Eiffel, stb. (Így például a 'kiterjesztett' nyelvek esetében általában külön kulcsszóra - virtual - van szükség annak jelzésére, hogy egy mûveletet a változó dinamikus típusának megfelelõen akarunk végrehajtatni, s ez csak pointerekhez kapcsolt objektumok esetén mûködik elvárásainknak megfelelõen, míg az objektum orientáltnak tervezett nyelvek esetén ez az alapértelmezés, s a változók 'természetesen' referenciák az objektumokra.)

Az öröklõdés támogatása számos további kérdést felvet, ezek egy csoportja a öröklõdés és a láthatóság viszonyára vonatkozik. Az örökös mindent megkap, de vajon mindent lát-e? Az Eiffel tervezõi úgy döntöttek, hogy ezek ortogonális fogalmak: így az örökös szabadon rendelkezik saját jellemzõi láthatóságáról, akár öröklöttek, akár újonnan bevezetettek. Más nyelvek, például a Simula, vagy a C++ tervezõi három láthatósági szintet vezettek be - a private láthatóságú jellemzõk, noha részt vesznek az öröklésben, nem hozzáférhetõek a leszármazott számára, a protected jellemzõk azok, amelyeket - bár a felhasználók számára rejtettek - az örökös szabadon manipulálhat, s a public-ként definiáltak láthatóak a külvilág számára is.

Megengedi-e a nyelv típusspecifikációk leírását, reprezentáció nélkül, vagy csak részleges reprezentációval? Az ilyen, ún. absztrakt osztályoknak természetesen még nem lehetnek objektumaik, a polimorfizmus és a dinamikus kötés kihasználásával azonban az általános és egységes kezelésük lehetõvé válik.

Ha a nyelv támogatja a többszörös öröklõdést, azaz egy új osztályt definiálhatunk egynél több másik osztály leszármazottjaként, akkor eszközökkel kell rendelkezzen a névkonfliktusok feloldására is. Kézenfekvõ lehetõség az átnevezés - ahogy pl. az Eiffel tervezõi kialakították, de a C++ például ilyen eseteket csak az õsökre való minõsített hivatkozással tud kezelni, s ezt különbözõ 'ügyeskedésekkel' rejthetjük el a késõbbi leszármazottak elõl.

A leszármazott osztály az õsosztálytól örökölt mûveleteket általában nemcsak kibõvítheti, hanem specializálhatja is, megadhatja az erre az osztályra érvényes speciális implementációját. Pl. alakzatok hierarchiájában minden osztálynak lehet egy megjelenítõ, kirajzoló mûvelete, de a sokszögek ezt természetesen másképp implementálják, mint a körök. A megbízhatóság szempontjából ugyanakkor nagyon fontos lenne, hogy ezek az átdefiniálások ne adhassanak tetszõleges implementációt az adott mûvelethez, hanem csak olyat, ami az eredeti specifikációnak egy alspecifikációját valósítja meg. Az erre vonatkozó ellenõrzés a nyelvek többségénél (C++, Ada95, Smalltalk, Simula, Modula, Pascal stb.) kimerül a mûveletek

ún. specifikációs sorának (név, formális paraméterek száma és típusa) elõírásában, az Eiffel az, amely az új elõ- és utófeltételekre is ad megkötéseket.

4. Összefoglalás

Az Eötvös Loránd Tudományegyetemen a programozási nyelvek oktatását a lényeges fogalmak - változók, kifejezések, utasítások, hatáskör, eljárások, absztrakt adattípusok, kivételkezelés, párhuzamosság, öröklõdés stb. áttekintésével zárjuk. Úgy gondoljuk, hogy ha megértetjük ezeket a fogalmakat, és megvalósításukat a különbözõ programozási nyelvekben, többet érünk el, mint néhány programozási nyelv elszigetelt oktatásával.

A nyelvek tervezõi a különbözõ felmerülõ kérdésekre különbözõ megoldási javaslatokat adtak. Ezek ismerete segíti a hallgatókat, hogy az egyetem elvégzéséig kialakuljon bennük egy nyelvektõl független szemléletmód, és egy alkalmazkodási képesség, ami megkönnyíti számukra a különbözõ konkrét környezetekbe való beilleszkedést. Ez a megközelítés - bár kellõ óvatosságot javaslunk a programozási nyelvek Bábel tornyának bõvítésénél - újabb nyelvek tervezésekor is nagy segítséget jelenthet.

Tapasztalataink szerint a Programozási nyelvek ilyen megközelítése érdekli a hallgatókat, számos szakdolgozat is született az elmúlt években a témából.

Irodalom:

[1] HOROWITZ, E.: Magasszintû programnyelvek
Mûszaki, Budapest, 1987.

[2] MARCOTTY, M. - LEDGARD, H.: The World of Programming Languages
Springer Verlag, 1986.

[3] BARON, N.S.: Computer Languages
Penguin, London, 1986.

[4] FISCHER, A.E. - GRODZINSKY, F.S.: The Anatomy of Programming Languages
Prentice-Hall, 1993.

[5] BEIL, D. H.: Adatállomány-feldolgozás COBOL nyelven
Mûszaki Könyvkiadó, Budapest, 1985.

[6] CARGILL, T.: C++ Programming Style
Addison Wesley, Reading (Mass.), 1992

[7] DAHL, O. J., MYRHAUG, B., NYGAARD K.: The SIMULA67: Common Base Language
Publication No. S-2, Norwegian Computer Center, Oslo, May 1968

[8] GOLDBERG, A., ROBSON, D.: Smalltalk-80: The Language and its Implementation
Addison-Wesley, Reading (Mass.), 1983.

[9] HABERMANN,N.A. ADA for Experienced Programmers
Addison-Wesley, 1983

[10] JENSEN K., WIRTH N., A Pascal programozási nyelv leirása
Mûszaki Könyvkiadó, Budapest, 1988.

[11] KING, K.N. TopSpeed Modula-2, Language Tutorial
Jensen & Partners International, 1990.

[12]KOZICS, S.: Az ADA programozási nyelv
ELTE jegyzet, 19 92.

[13] KOZICS, S. : Az ALGOL60, a FORTRAN, a COBOL és a PL/1 programozási nyelvek
ELTE jegyzet, 1992.

[14] LIPPMAN, S. B.: A C++ Primer
Addison-Wesley, Reading (Mass.), 1989.

[15] LISKOV, B. H., GUTTAG, J.: Abstraction and Specification in Program Development
M.I.T. Press, Cambridge (Mass.), 1986.

[16] LÕCS, GY., VIGASSY, J.: A FORTRAN programozási nyelv
Mûszaki Könyvkiadó, Budapest 1981.

[17] MEYER, B.: Object-Oriented Software Construction
Prentice Hall, New York, 1988.

[18] MEYER, B.: Eiffel - The Language
Prentice Hall, Hertforshire, 1992.

[19] NAGY, K.: Struktúrált programozás COBOL nyelven
Számok, Budapest, 1980.

[20] NORTON, P., YAO, P.: Borland C++ Programming for Windows
Bantam, New York, 1992.

[21] RÁKOSI, M.: PL/1
Mûszaki Könyvkiadó, Budapest 1974.

[22] Reference Manual for the ADA Programming Language
1995.

[23] RITCHIE: The C Programming Language
Prentice-Hall, 1978.

[24] SCHAFFERT,C.,COOPER,T.,BULLIS,B.,KILIAN,M.,WILPOLT,C.: An Introduction to Trellis/Owl
OOPSLA '86 Proceedings, September 1986, p. 9-16.

[25] SHAW, M.,(ed.): Alphard Form and Content
Springer- Verlag New York, 1981.

[26] STROUSTRUP, B.: The C++ Programming Language, Second Edition
Addison-Wesley, Menlo Park (Calif.), 1993.

[27] WATT, D. A., WICHMANN, B. A., FINDLAY, W.: ADA Language and Methodology
Prentice Hall, 1987

[28] WEGNER, P.: Dimensions of Object-Based Language Design
OOPSLA '87 Proceedings, 168-182.

[29] WIRTH, N., From Modula to Oberon
Software-Practice and Experience Vol 18 (7),661-670 (July 1988)

[30] WIRTH, N., The Programming Language Oberon
Software-Practice and Experience Vol. 18(7),671-690 (July 1988)

[31] WIRTH,N.,GUTKNECHT,J.: Project Oberon
The Design of an Operating System and Compiler
Addison-Wesley Publ. C., 1992