Programozási nyelvek III

Nyékyné Gaizler Judit
nyeky@elte.hu
http://people.inf.elte.hu/nyeky

Programozási nyelvek és automaták sáv:

- kötelező „mag”:
  - Automataelmélet
  - Formális szemantika
  - **Programozási nyelvek**

- szabadon választható témák:
  - Formális nyelvek és automaták
  - Formális szemantika
  - A Java és az Eiffel programozási nyelvek
  - Programozási nyelvek szeminárium
  - Funkcionális programozás
  - Sztochasztikus automaták
  - Mesterséges intelligencia nyelvek
  - stb.

Az oktatás célja

- a különböző magas szintű programozási nyelvek nyelvi elemeinek megismerése
- a jelen fejlődési irányainak áttekintése

Tematika

- Bevezetés
- A programozási nyelvek elemei
- Beépített adattípusok, változók, kifejezések
- Utasítások
- Alprogramok és paramétereik
- Absztrakt adattípusok
- Sablonok

Tematika (folyt.)

- Kivételek
- Objektum-orientált programozás
- Helyesség
- Párhuzamosság
- A könyvtártervezés elvei

Követelmények

- Mindenki választ egy nyelvet, s a választott nyelvről
  - Az órákon a jellemző tulajdonságok ismertetése
  - Leírás készítése/bővítése a fenti szempontrendszer alapján
- Vizsga
  - Írásbeli
  - Szóban javítási lehetőség

Cél	Segítség	Eszköz
Jó minőségű program **termék**	⇦ **Progr. módszertan** ⇦	**Progr. nyelv**

Szoftverek minőségi szempontjai

- helyesség
- megbízhatóság
- karbantarthatóság
- újrafelhasználhatóság
- kompatibilitás
- hordozhatóság
- hatékonyság
- stb.

Helyesség

A program **helyes,** ha pontosan megoldja a feladatot, megfelel a kívánt specifikációnak.

Alapvető: a pontos és minél teljesebb specifikáció.

Megbízhatóság

Egy programot **megbízhatónak** nevezünk, ha *helyes*, és abnormális - a specifikációban nem leírt - helyzetekben sem történik katasztrófa, hanem valamilyen „ésszerű" működést produkál.

Karbantarthatóság

A **karbantarthatóság** annak mérőszáma, hogy milyen könnyű a programterméket a specifikáció változtatásához adaptálni.

Egyes felmérések szerint a szoftver költségek 70 %-át szoftver karbantartásra fordítják!

A karbantarthatóság növelése szempontjából a két legfontosabb alapelv:

- a tervezési egyszerűség
- a decentralizáció (minél önállóbb modulok létrehozása)

Újrafelhasználhatóság

Az **újrafelhasználhatóság** a szoftver termékek azon képessége, hogy egészben vagy részben újra felhasználhatóak új alkalmazásokban.

Kompatibilitás

- A **kompatibilitás** azt mutatja meg, hogy **milyen könnyű a szoftver termékeket egymással kombinálni.**

Hordozhatóság

- A program **hordozhatósága** annak mérőszáma, mennyire könnyű a programot **más géphez, konfigurációhoz, vagy operációs rendszerhez - általában más környezethez** - átalakítani.

Hatékonyság

A program **hatékonysága** a futási idővel és a felhasznált memória méretével arányos - **minél gyorsabb, ill. minél kevesebb memóriát használ**, annál hatékonyabb.

Barátságosság

A program **emberközelisége, barátságossága** a **felhasználó** számára rendkívül fontos: ez megköveteli, hogy **az input logikus és egyszerű**, az **eredmények formája áttekinthető** legyen.

Tesztelhetőség

- A **tesztelhetőség, áttekinthetőség** a program karbantartói, fejlesztői számára fontos.

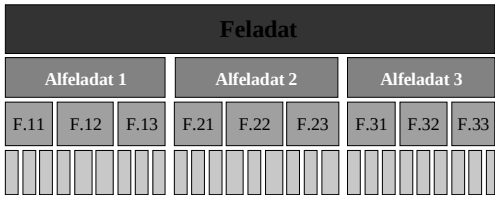
Programozási módszertan

- moduláris tervezés
  programjainkat egymástól minél inkább független, jól definiált kapcsolatban álló programegységekként tervezzük.
- strukturált
- objektumorientált

Moduláris dekompozíció

- A moduláris dekompozíció a feladat több egyszerűbb részfeladatra bontását jelenti, amely részfeladatok megoldása már egymástól függetlenül elvégezhető. Ennek segítségével csökkentjük a feladat bonyolultságát.
- Általában a módszert ismételten alkalmazzuk, azaz az alrendszereket magukat is dekomponáljuk. Ezzel lehetővé tesszük azt is, hogy a feladat megoldásán egyszerre több ember is dolgozzon. A módszer egy fával ábrázolható, ahol a fa csomópontjai az egyes dekompozíciós lépéseknek felelnek meg.

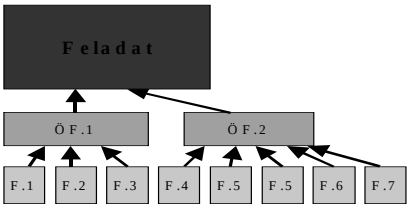
Moduláris dekompozíció



Moduláris kompozíció

- Olyan szoftver elemek létrehozását támogatja, amelyek szabadon kombinálhatók egymással.
- Programjainkat minél inkább már meglévő programegységekből, mint építőkövekből szeretnénk felépíteni.

Moduláris kompozíció



Moduláris érthetőség

A modulok önállóan is egy-egy értelmes egységet alkossanak, megértésükhöz minél kevesebb „szomszédos” modulra legyen szükség.

Moduláris folytonosság

A specifikáció „kis” változtatása esetén a programban is csak „kis” változtatásra legyen szükség.

Moduláris védelem

Célunk kell legyen a program egészének védelme az abnormális helyzetek hatásaitól. Egy hiba hatása egy - esetleg néhány - modulra korlátozódjon!

A modularitás alapelvei

- ***A modulokat nyelvi egységek támogassák***
  - A modulok illeszkedjenek a használt programozási nyelv szintaktikai egységeihez.
- ***Kevés kapcsolat legyen***
  - Minden modul minél kevesebb másik modullal kommunikáljon!
- ***Gyenge legyen a kapcsolat***
  - A modulok olyan kevés információt cseréljenek, amennyi csak lehetséges!

A modularitás alapelvei (folyt.)

- ***Explicit interface kell***
  - Ha két modul kommunikál egymással, akkor annak ki kell derülnie legalább az egyikük szövegéből.
- ***Információ elrejtés***
  - Minden információ egy modulról rejtett kell legyen, kivéve, amit explicit módon nyilvánosnak deklaráltunk.

A modularitás alapelvei (folyt.)

- ***Nyitott és zárt modulok***
  - Egy modult **zártnak** nevezünk, ha más modulok számára egy jól definiált felületen keresztül elérhető, a többi modul változatlan formában felhasználhatja.
  - Egy modult **nyitottnak** nevezünk, ha még kiterjeszthető, ha az általa nyújtott szolgáltatások bővíthetőek vagy ha hozzávehetünk további mezőket a benne levő adatszerkezetekhez, s ennek megfelelően módosíthatjuk eddigi szolgáltatásait.

Az újrafelhasználhatóság igényei

- ***A típusok változatossága***
  - A modulok többféle típusra is működjenek.
- ***Adatstruktúrák és algoritmusok változatossága***
- ***Egy típus - egy modul***
  - Egy típus műveletei kerüljenek egy modulba.
- ***Reprezentáció-függetlenség***

Mi a programozási nyelv?

- Egy jelölés …
- Eszköz
  - a számítógép vezérlésére
  - programozók közötti kommunikációra
  - algoritmusok leírására
  - magas szintű tervezésre
  - a feladat bonyolultságának kezelésére

Programozási nyelvek kialakulása

- gépi kód
- assembly nyelv
  - mnemonikok, címkék használata
  - assembler
- magas szintű nyelv
  - utasításkészlete független egy adott számítógép-architektúra utasításkészletétől: végrehajtásuk előtt egy fordítóprogramnak kell őket assembly kódra, illetve gépi kódra fordítani.

A Neumann-elvű nyelvek tulajdonságai:

- **A memória rekeszeinek absztrakciójaként bevezetik a *változó* fogalmát.**
- **A kiszámított értékeket tárolni kell. Az "eltárolás" művelete az *értékadás.***
- **Az *utasítások ismételten végrehajthatók*.**

Tervezési szempontok

- Jól definiált szintaktikai és szemantikai leírás
- Megbízhatóság
  - kivételkezelés, helyesség ellenőrzés
- Karbantarthatóság, kiterjeszthetőség
  - új adattípusok definiálása
  - operátorok túlterhelése
- Gyors fordíthatóság, hatékony tárgykód
- Ortogonalitás
- Hordozhatóság
- Általánosság

A nyelvek „fejlődése”

Ma: több, mint 2500 nyelv:

http://people.ku.edu/~nkinners/LangList/Extras/search.htm

Nyelvek osztályozása

- Paradigmák szerint:
  - blokkszerkezetű, procedurális
  - objektum alapú, osztály alapú, objektum-orientált
  - osztott, párhuzamos
  - funkcionális
  - logikai
  - adatbázis

Nyelvek osztályozása (folyt.)

- A nyelv célja szerint:
  - rendszerprogramozási
  - üzleti
  - tudományos
  - adat alapú
  - lista kezelő
  - vektor kezelő
  - string kezelő
  - parancsfeldolgozó

(Peter Wegner)

Nyelvi elemek, alapfogalmak

Általános fogalmak

- **A nyelv *szintaxisa* azoknak a szabályoknak az összessége, amelyek az adott nyelven írható összes lehetséges, *formailag helyes* programot (jelsorozatot) definiálják.**
  - **Reguláris kifejezések, BNF forma**
- **Az adott nyelv programjainak *jelentését* leíró szabályok összessége a nyelv *szemantikája*.**

Szintaxis és szemantika

Pl.: DD / DD / DDDD 01 / 02 / 2004
2004. január 2. vagy 2004. február 1.?

Szintaxis befolyásolja a programok megbízhatóságát:
FORTRAN:

DO 10 I =1.5
A I = X+B(I)
10 CONTINUE

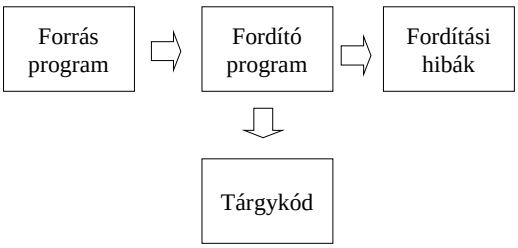
A programok *végrehajtása*

- **Az *interpreter* egy utasítás lefordítása után azonnal végrehajtja azt**
- **a *fordítóprogram* átalakítja a programot egy vele ekvivalens formára, ez lehet a számítógép által közvetlenül végrehajtható forma, vagy lehet egy másik programozási nyelv.**

Fordítás

- **A lefordítandó program a *forrásprogram*.**
- **A fordítás eredményeként kapott program a *tárgyprogram.***
- **Az az idő, amikor az utasítások fordítása folyik, a *fordítási idő.***
- **Az az idő, amikor az utasítások végrehajtása folyik, a *végrehajtási idő.***

Fordítás:

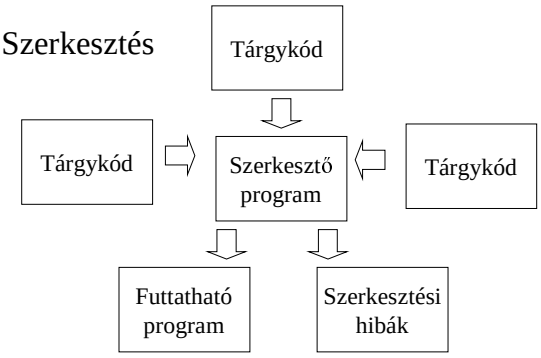


Programszerkezet

- ***fordítási egység vagy modul:*** **A nyelvnek az az egysége, ami a fordítóprogram egyszeri lefuttatásával, a program többi részétől elkülönülten lefordítható.**
- **Több modulból álló programok esetében a fordítás és a végrehajtás között: a *program összeszerkesztése*.**

2005.02.24. 44

Szerkesztés



Programegység, fordítási egység

- **A *programegység* egy részfeladatot megoldó, tehát funkcionálisan összefüggő programrész.**
- **A *fordítási egység* programegységek halmaza.**

2005.02.24. 46

A programegységek részei

- **A *specifikációs rész* írja le az egységnek a más egységből elérhető "kapcsolódási pontjait".**
- **A *törzs* tartalmazza az egység funkcióját megvalósító programot.**

2005.02.24. 47

```
procedure Hello;

with Text_Io; use Text_Io;
procedure Hello is
    S: String(1..40):=(others=>' ');
    Last: Natural;
begin
    Put("Hogy hívnak?");
    Get_Line(S, Last);
    New_Line;
    Put("Hello "&S(1..Last));
end Hello;
```

specifikáció

törzs

A törzs felépítése

- **A deklarációs rész azonosítók *deklaráció*it, valamint konstansok, típusok, objektumok és programegységek *definíció*it tartalmazhatja.**
- **Az *utasítássorozat* tartalmazza azokat az utasításokat, amelyek *végrehajtása* nyomán a programegység kifejti a hatását.**

Deklarációk

A deklarációs rész feldolgozását, a definíciók értelmezését *a deklarációs rész kiértékelésé*-nek mondjuk.

A deklarációs rész definícióinak kiértékelése történhet
***fordítási* (Modula-2, C) vagy**
***futási* (PL/I, Ada) időben.**

Az első esetben:
***statikus definíció-kiértékelés*,**

a másodikban:
***dinamikus definíció-kiértékelés*.**

Deklarációk (folyt.)

- **Változódeklaráció: állapottér megadásához formája általában:**
  azonosító: típus
  N: Natural;

vagy:

típus azonosító
int i;

Hatáskör, láthatóság

- **A programszövegnek azt a szakaszát, amelyen belül az adott deklaráció érvényben van, a deklaráció *hatáskör*ének nevezzük.**
- **A programszövegnek azt a szakaszát, ahol a deklarált azonosítóval az adott deklarációra hivatkozni lehet, a deklaráció *láthatósági kör*ének nevezzük.**

Blokkstruktúra

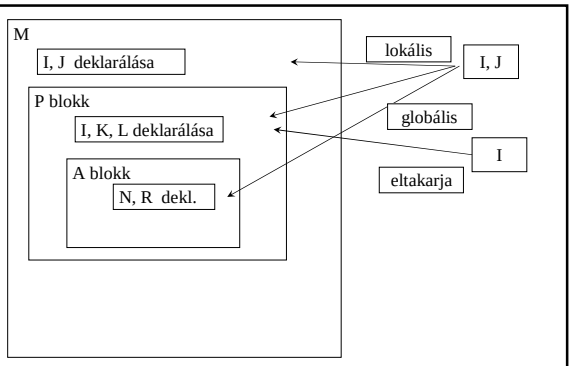
- **A *blokkstruktúra* a programegységek egymásba ágyazásával előálló hierarchikus struktúra.**
- **Blokkstruktúrában egy programegység deklarációinak hatásköre kiterjed az összes *tartalmazott* programegységre is.**

Globális - lokális:

- **Egy programegységben a tartalmazó programegységek által deklarált azonosítókat (a tartalmazott programegység szemszögéből) *globális* azonosítóknak,**
- **a programegységben deklarált azonosítókat magában a programegységben *lokális* azonosítóknak nevezzük.**

Globális- lokális:

- **Blokkstruktúra esetén egy programegységben egy globális azonosító újradeklarálható, ekkor a lokális deklaráció *eltakarja* a globális deklarációt.**



Azonosító túlterhelése

- **Egy azonosítónak ugyanabban a szövegrészben több deklarációja is érvényben lehet.**
- **A túlterhelést a leggyakrabban alprogramok nevei esetében teszik lehetővé.**

  2 + 3

  2.1 + 3.0

Erősen típusos nyelv

- **Egy nyelv *erősen típusos*, ha minden érték, objektum, formális paraméter és függvény típusa fordítási időben egyértelműen meghatározható.**

Memóriakezelés

- A változóhoz a memóriaterület hozzárendelése (*allokálás*-a)
  - *automatikusan*, a definíció kiértékelésekor
  - a programozó rendelkezik róla
- a változó által lefoglalt terület felszabadítása
  - automatikus,
  - a programozó hatáskörébe tartozik.

Élettartam

A változókhoz a szükséges memóriaterület lefoglalása (allokálása) és annak felszabadítása közötti <u>időt</u> a változó *élettartam*-ának nevezzük.

statikus változó:

- élettartama a program egész működése idejére kiterjed
- elhelyezése mindig az ún. *statikus* (main) *memória*-területre történik
- a statikus terület egyszer, a program betöltésekor kerül lefoglalásra

dinamikus változó:

- a program explicit módon foglal le területet, *a dinamikus* (heap) *memória*-területen, amire a címével, ún. *pointer*-rel lehet hivatkozni.
- egyes nyelvek tartalmaznak utasítást a dinamikus területek felszabadítására is.
- blokkstruktúrában a programegységek (nem statikus) lokális változói az ún. *végrehajtási verem*ben helyezkednek el.

Paraméterátadás

- A programegységek legelterjedtebb fajtája az *alprogram*. A nyelvek kétféle alprogramot használnak, ezek az *eljárás*ok és a *függvény*ek.
- Az alprogramok *formális paraméter*ekkel rendelkeznek, amelyeknek az *alprogram aktivizálásakor* (*hívásakor*) *aktuális paramétereket* kell megfeleltetni.

  (később…)

A programozási nyelvek rövid története

- FORTRAN (1950-es évek közepe-vége)
  - hangsúly a tudományos programozáson
- LISP (1950-es évek vége)
  - Szimbolikus listakezelés. MI közösség használja.
  - van benne rekurzió, dinamikus helyfoglalás és felszabadítás (szemétgyűjtés).
- ALGOL 60 (1960)
  - az adattípus fogalmának megjelenése
  - blokkszerkezet
  - érték- és név szerinti paraméterátadás

A programozási nyelvek rövid története

- COBOL (1960)
  - adatkezelés
  - angol-szerű szintaxis
  - hierarchikus adatszerkezetek megengedettek (rekordok rekordjai)
- BASIC (1960-as évek közepe)
  - érdekes tervezési cél: “a használó ideje fontosabb, mint a számítógép ideje”
  - diákok programozás oktatására
  - Interaktív interpretált környezet

A programozási nyelvek rövid története

- PL/I (1960-as évek közepe)
  - általános célúnak tervezték
  - megengedi, hogy párhuzamosan végrehajtódó taszkokat hozzunk létre
  - kezeli a futási idejü kivételeket
  - pointerek mint adattípusok megjelenése
  - tömbök részeire is hivatkozhatunk
- APL (1960 körül)
  - tömb és mátrix kezelés
- SNOBOL (1960-as évek közepe)
  - szövegkezelés
  - müveletek szövegminták hasonlítására

A programozási nyelvek rövid története

- SIMULA 67 (1967)
  - szimulációs célok.
  - osztály fogalmának bevezetése – az adatok és a rajta végezhető műveletek egységbezárása
  - öröklődés megjelenése
- ALGOL 68 (1968)
  - elsődleges tervezési cél: ortogonalitás.
  - a legtöbb ezt követő programozás nyelv merített a tervezéséből
- Pascal (1970-es évek eleje)
  - a programozás oktatására tervezték

A programozási nyelvek rövid története

- C (1970-es évek eleje)
  - rendszerprogramozásra
  - rugalmas, de megbízhatatlan
  - széleskörűen használják, részben a UNIX miatt
- Prolog (1970-es évek közepe)
  - Logikai, nem-procedurális nyelv
  - a programok állítások és szabályok halmazából állnak

A programozási nyelvek rövid története

- Ada (1970-es évek vége)
  - A Pentagon megbízásából
  - csomagok az absztrakt adattípusok kezelésére
  - Kivételkezelés
  - Sablonok (Generic)
  - Párhuzamosság támogatása (taszkok, randevúk)
- Smalltalk (1980)
  - Objektumorientált programozási nyelv
  - nem csak egy nyelv, hanem egy grafikus környezet is

A programozási nyelvek rövid története

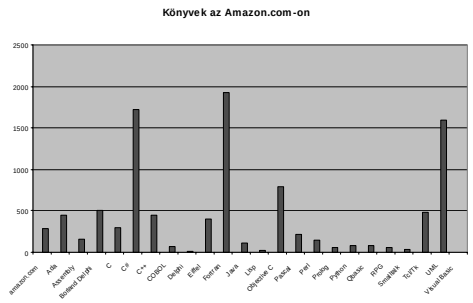
- C++ (1980-as évek eleje)
  - a SIMULA 67 és a Smalltalk sok objektumorientált jellemzőjét házasították össze a C-vel
  - tervezési cél: ne csökkenjen a hatékonyság a C-hez viszonyítva
  - Paraméterezett típusok (templates)
  - Kivételkezelés
  - Operátor túlterhelés
- Eiffel (1980-as évek eleje)
  - objektumorientált. egyszerűbb, mint a C++, nagyon jól átgondolt tervezés eredménye
  - *tervezés szerződéssel*: állításokat használ egy szerződés kifejezésére az alprogramok és a hívóik között

A programozási nyelvek rövid története

- Java (1990-es évek közepe)
  - kisebb, egyszerűbb, megbízhatóbb C++
  - minden kódot osztályokban kell megvalósítani
  - párhuzamosság támogatása (threadek)
  - szemétgyűjtés
  - Kivételkezelés
- C# - a Microsoft „válasza” a Javára
  - .NET alapú fejlesztés támogatása

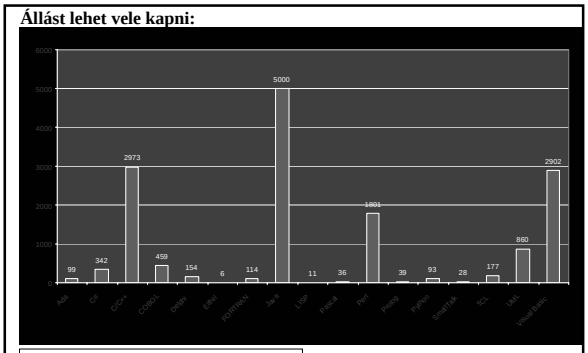
Mi tesz egy programozási nyelvet fontossá?

Sok ember tanulja:



2005. február

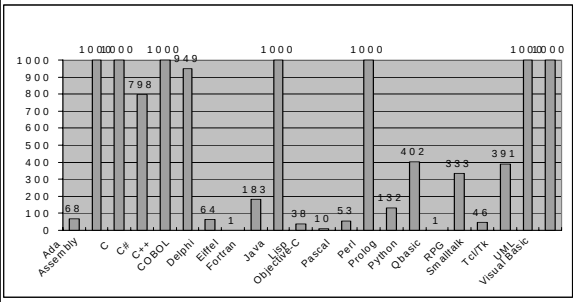
Állást lehet vele kapni:



A monster.com álláshirdetései: USA

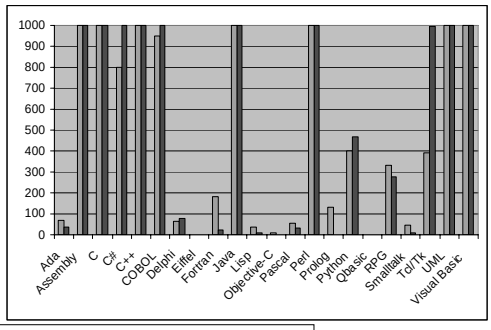
2004 január

Állást lehet vele kapni: 2005 február



A monster.com álláshirdetései: USA

Állást lehet vele kapni: 2005 február



A monster.com álláshirdetései: USA, India

Irodalom

- Sebesta, R.W.: Concepts of Programming Languages sixth ed. Addison-Wesley, 2004.
- Meyer, B.: Object-Oriented Software Construction 2nd ed. Prentice Hall, 1997.
- Nyékyné Gaizler J. (szerk.) et al.: Programozási nyelvek, Kiskapu, 2003.

Irodalom

- Böszörményi, L., Weich, C.: Programming in Modula-3, Springer, 1996.
- Cohen, N. H. : Ada as a second language; McGraw-Hill; 1996.
- Dahl, O.J., Myrhaug, B., Nygaard K.: The SIMULA67: Common Base Language; Publication no S-2, Norwegian Comp. Center, Oslo, May 1968; 1968.

Irodalom

- Goldberg, A., Robson, D.: Smalltalk-80: The Language and its Implementation; Addison-Wesley, Reading (Mass.); 1983.
- Gosling, J., Joy, B., Steele, G.: The Java Language Specification; Addison-Wesley; 1996.
- Horowitz, E.: Magasszintű programnyelvek; Műszaki, Bp.; 1987.
- Ada Reference Manual; 1995.
- Ada 95 Rationale; 1995.

Irodalom

- Liskov, B. H., Guttag, J.: Abstraction and Specification in Program Development; M.I.T.Press, Cambridge (Mass.); 1986.
- Marcotty, M., Ledgard, H.: The World of Programming Languages; Springer ; 1987.
- Meyer, B.: Eiffel, The Language; Prentice Hall; 1992.

Irodalom

- Meyer, B.: ISE Eiffel, The Libraries; Prentice Hall; 1995.
- Nyékyné Gaizler J. (szerk.) et al.: Az Ada 95 programozási nyelv; Egyetemi tankönyv, Budapest, 1999. ELTE Eötvös Kiadó.
- Nyékyné Gaizler J. (szerk.) et al.: Java 2 útikalauz programozóknak:1.3; Budapest, 2001, ELTE TTK Hallg. Alapítvány.

Irodalom

- Sethi, R.: Programming languages 2nd ed. Addison-Wesley, 1996.
- Shaw, M. (ed.): Alphard Form and Content; Springer; 1981.
- Stroustrup, B.: A C++ programozási nyelv; Kiskapu-Addison-Wesley; 2001.
- Wegner, P.: Dimensions of Object-Based Language Design; OOPSLA '87 Proceedings, 168-182.; 1987.
- Wirth, N.: Type Extensions; ACM Trans. on Prog. Lang. and Systems, Vol. 10, No.2, April 1988, 204-214.; 1988.