Fordítóprogramok célja és szerkezete

Fordítóprogramok előadás (A,C,T szakirány)

Miért van szükség fordítóprogramokra?

Így kényelmes programozni

```
int sum = 0;
for( int i=0; i<len; ++i )
  sum += t[i];
```


Miért van szükség fordítóprogramokra?

Így kényelmes programozni

```
int sum = 0;
for( int i=0; i<len; ++i )
  sum += t[i];
```

Ezt tudja végrehajtani a számítógép

```
B9 00 00 00 00
B8 00 00 00 00
81 F9 0A 00 00 00
7D 06
03 04 8B
41
EB F2
```


Miért van szükség fordítóprogramokra?

- magasszintű programozási nyelvek
  - könnyebb programozni
  - közelebb a megoldandó problémához
  - platform-függetlenség
- gépi kód
  - gépközeli (numerikus utasításkódok, regiszterek, memóriahivatkozások ...)
  - erősen platform-függő
  - optimalizált
- Fordítóprogramokat *általában* a kettő közötti átalakításra használunk.

Assembly és assembler

- az assembly nyelvekben
  - a gépi kód utasításaihoz neveket (*mnemonik*okat) rendelünk
  - a regiszterekre névvel hivatkozunk
  - az egyes memóriacímeket címkékkel azonosítjuk
  - gyakran közbülső állomás a magasszintű nyelvről gépi kódra történő fordítás során
- assembler: assembly nyelvről gépi kódra fordít

Assembly és assembler

Gépi kód

```
B9 00 00 00 00
B8 00 00 00 00

81 F9 0A 00 00 00
7D 06
03 04 8B
41
EB F2
```

Assembly

```
mov ecx,0
mov eax,0
eleje:
cmp ecx,10
jge vege
add eax,[ebx+4*ecx]
inc ecx
jmp eleje
vege:
```

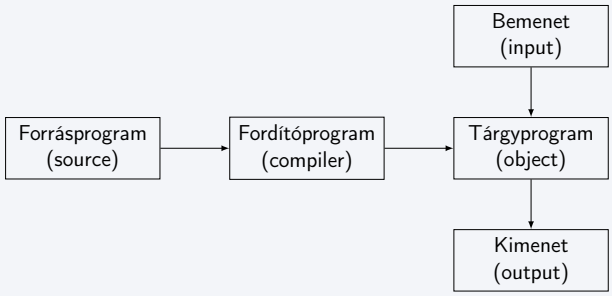
Példák

- magasszintű nyelvek fordítóprogramjai (compiler)
  - gcc, javac, ghc ...
- assembly nyelvek fordítóprogramjai (assembler)
  - nasm, masm, gas ...
- nyelvek közötti fordítás (translator)
  - Fortran $\Longrightarrow$ C, latex2html, dvips ...
- egyéb fordítóprogramok
  - latex
  - hardware leíró nyelvből áramkörök tervei

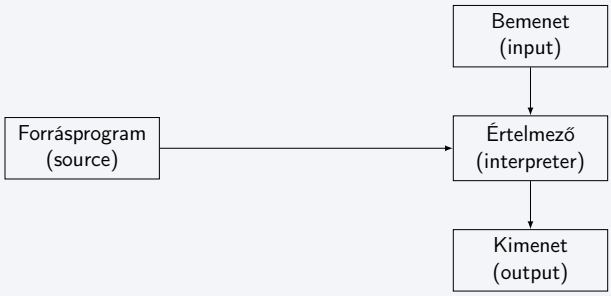
Fordítás és interpretálás

- Fordítás: A *forráskódot* a fordítóprogram *tárgyprogrammá* alakítja. A tárgyprogramokból a *szerkesztés* során *futtatható* állomány jön létre.
  - Fordítási idő: amikor a fordító dolgozik
  - Futási idő: amikor a program fut
- Interpretálás: A forráskódot az interpreter értelmezi és azonnal végrehajtja.
  - a fordítási és futási idő nem különül el

Fordítás



Interpretálás



Fordítás vs. interpretálás

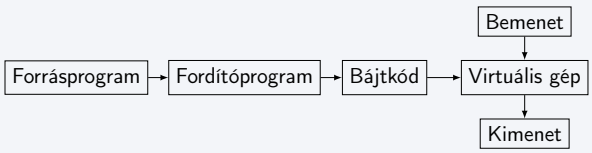
Fordítás
- gyorsabb végrehajtás
- a forrás alaposabb ellenőrzése
- minden platformra külön-külön le kell fordítani
- C++, Ada, Haskell ...

Interpretálás
- rugalmasabb (pl. utasítások fordítási időben történő összeállítása)
- minden platformon azonnal futtatható, ahol az interpreter rendelkezésre áll
- Perl, Php, Javascript ...

Fordítás + interpretálás

Fordítás és interpretálás egymásra építve (pl. *Java*):

1. A forráskód fordítása bájtkódra.
2. A bájtkód interpretálása virtuális géppel.

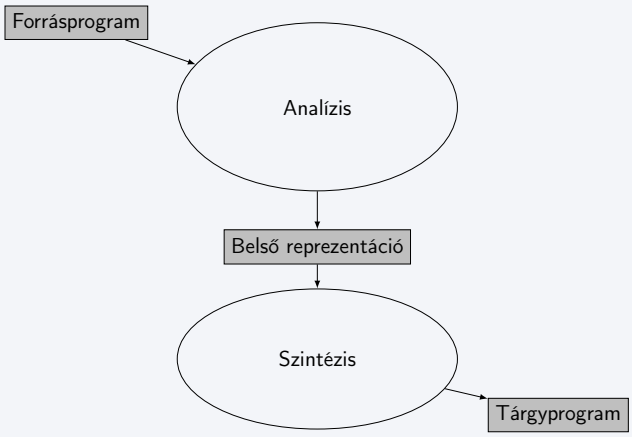


Virtuális gép: olyan gép szoftveres megvalósítása, amelynek a bájtkód a gépi kódja.

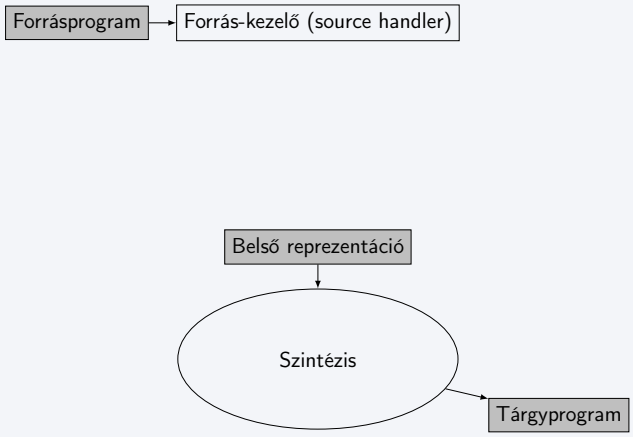
Fejlődés

- 1957, az első Fortran compiler: 18 emberévnyi munka
- azóta fejlődött a formális nyelvek és automaták elmélete
- ma: a fordítóprogramok létrehozásának egy része automatizálható
  - a programszöveget (szintaktikusan) elemző program: automatikusan generálható
  - a szemantikus ellenőrzést (pl. típusok) és kódgenerálást végző program: nem generálható automatikusan, de nem túl bonyolult
  - kódoptimalizálás: nehéz feladat
- Példa: elemzőgenerátorok

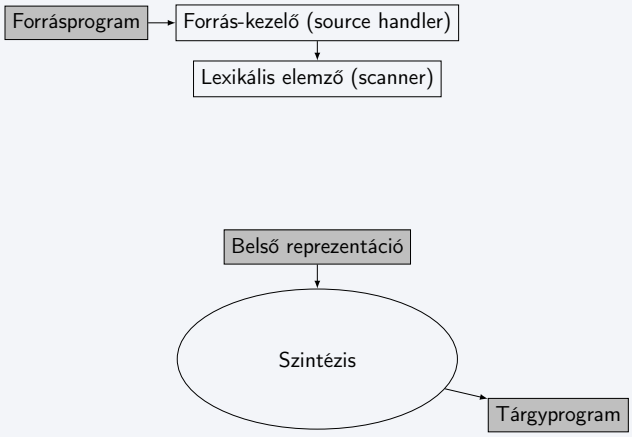
A fordítóprogramok szerkezete



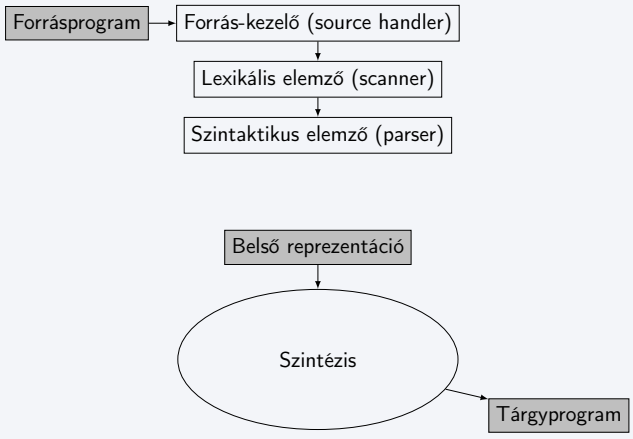
A fordítóprogramok szerkezete



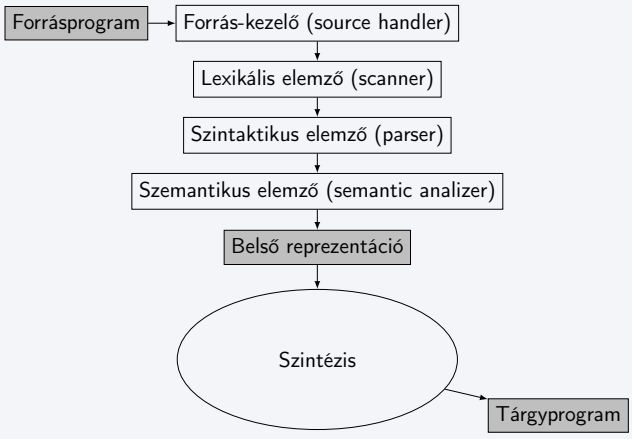
A fordítóprogramok szerkezete



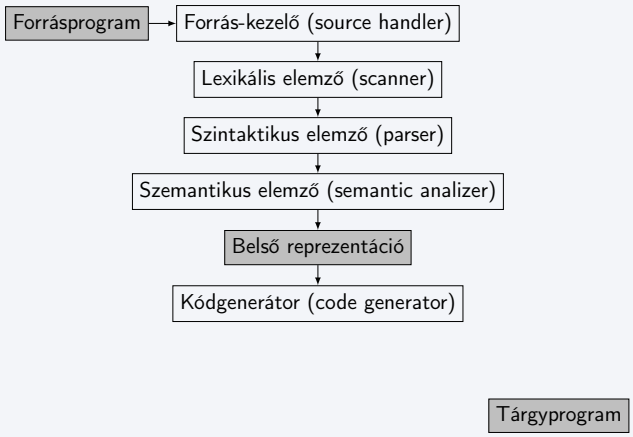
A fordítóprogramok szerkezete



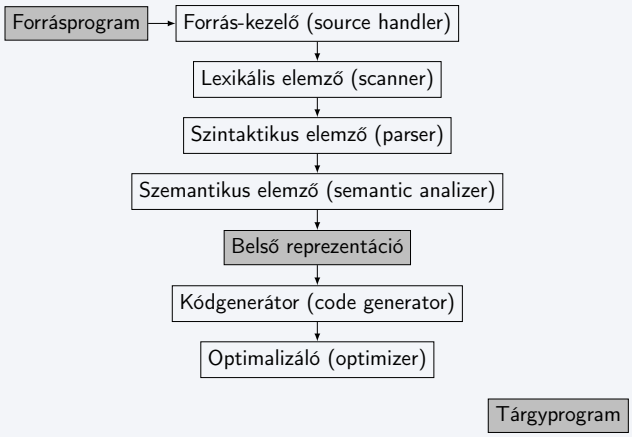
A fordítóprogramok szerkezete



A fordítóprogramok szerkezete

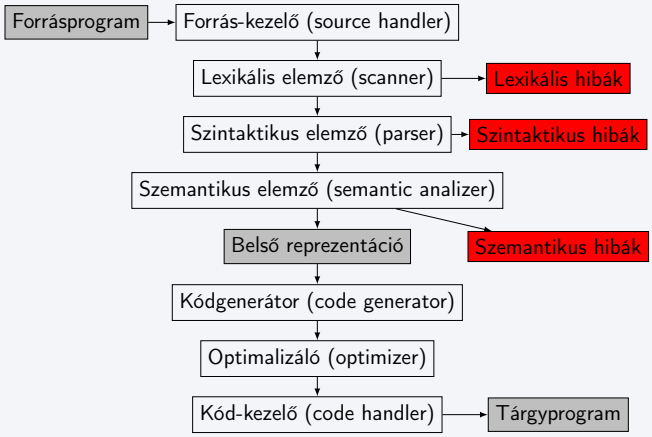


A fordítóprogramok szerkezete



A fordítóprogramok szerkezete

A fordítóprogramok szerkezete



Forrás kezelő

- bemenet: fájl(ok)
- kimenet: karaktersorozat
- feladata:
  - fájlok megnyitása
  - olvasás
  - pufferelés
  - az operációsrendszer-specifikus feladatok elvégzése
  - ha a fordítóprogram készít listafájlt, akkor ezt is kezeli

Példa

input.cpp $\Longrightarrow$ x␣=␣x␣+␣1;

Lexikális elemző

- bement: karaktersorozat
- kimenet: szimbólumsorozat, lexikális hibák
- feladata:
  - lexikális egységek (szimbólumok) felismerése: azonosító, konstans, kulcsszó...

Példa

x␣=␣x␣+␣1;

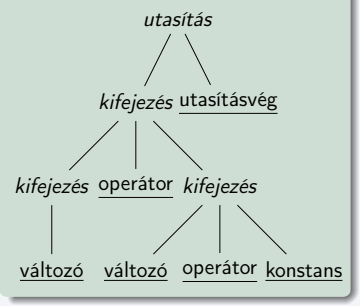
$\Longrightarrow$

változó[x] operátor[=] változó[x] operátor[+] konstans[1] utasításvég

Szintaktikus elemző

- bemenet: szimbólumsorozat
- kimenet: szintaxisfa, szintaktikus hibák
- feladata:
  - a program szerkezetének felismerése
  - a szerkezet ellenőrzése: megfelel-e a nyelv definíciójának?

Az `x = x + 1;` utasítás szintaxisfája



Szemantikus elemző

- bemenet: szintaktikusan elemzett program (szintaxisfa, szimbólumtábla, ...)
- kimenet: szemantikusan elemzett program, szemantikus hibák
- feladata:
  - típusellenőrzés
  - változók láthatóságának ellenőrzése
  - eljárások hívása megfelel-e a szignatúrának?
  - stb.

Szintaktikus hibák

```
x = x 1;
x = x + ;
if x==0) {}
```

Szemantikus hibák

```
if( ''alma'' == 3.14 ) {}
void f(int x) {} f(3,4);
{ int y; } y++;
```


Kódgenerátor

- bemenet: szemantikusan elemzett program
- kimenet: tárgykód utasításai (pl. assembly, gépi kód)
- feladata:
  - a forrásprogrammal ekvivalens tárgyprogram készítése

Az `x = x + 1;` utasítás egy lehetséges megvalósítása

```
mov eax,1
push eax
mov eax,[x]
pop ebx
add eax,ebx
mov [x],eax
```


Kódpotimalizáló

- bemenet: tárgykód
- kimenet: optimalizált tárgykód
- feladata: valamilyen szempontok szerint jobb kód készítése
  - pl. futási sebesség növelése, méret csökkentése
  - pl. felesleges utasítások megszüntetése, ciklusok kifejtése...
- optimalizáció végezhető
  - az eredeti programon (szemantikus elemzés után)
  - a tárgykódon (a kódgenerálás után)

Az `x = x + 1;` utasítás kódjának optimalizálása

```
mov eax,1
push eax
mov eax,[X]
pop ebx          ⟹          inc dword [X]
add eax,ebx
mov [X],eax
```


Emlékeztető: a fordítás komponensei

A fordítás menetei

- az egyes komponensek egymást követik
  - az egyik kimenete a másik bemenete
- ez nem jelenti azt, hogy pl. a lexikális elemzőnek be kell fejeződnie a szintaktikus elemzés előtt
  - a szintaktikus elemző meghívhatja a lexikálisat, amikor egy újabb szimbólumra van szüksége
- eredmény: egy utasításnak akár a tárgykódja is elkészülhet, amikor a következő utasítás még be sincs olvasva
- a kódoptimalizálásnál viszont jellemzően újra át kell olvasni az egész tárgykódot (akár többször is)

A fordítás menetszáma

A fordítás annyi menetes, ahányszor a programszöveget (vagy annak belső reprezentációit) végigolvassa a fordító a teljes fordítási folyamat során.