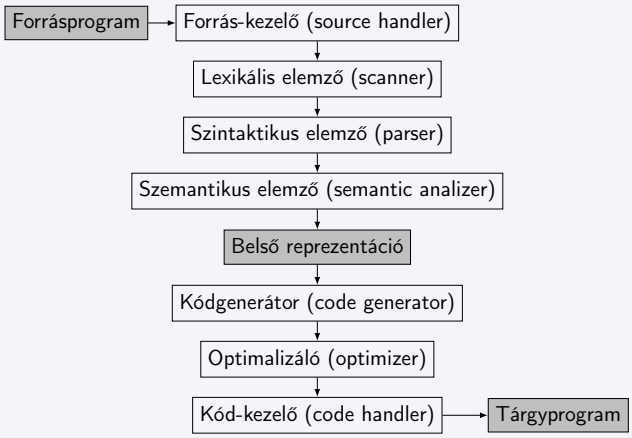
# Szimbólumtábla-kezelés

Fordítóprogramok előadás (A, C, T szakirány)



## Emlékeztető: a fordítás lépései






## Információáramlás

**Programszöveg**

```
x = y + 2;
```

- Lexikális elemző: lexikális elemek sorozatát állítja elő

**Lexikális elemek**

változó operátor változó operátor számkonstans utasításvég

- Szintaktikus elemző: felépíti a szintaxisfát



## Információáramlás

- Szemantikus elemző:
  - deklarálva van-e az $x$ és $y$ változó?
  - $x$ és $y + 2$ kifejezések típusa azonos-e?
  - stb.
- Kódgenerálás: utasítások generálása a tárgyprogram nyelvén, amelyek megvalósítják az értékadást.



## Információáramlás

- Szemantikus elemző:
  - deklarálva van-e az $x$ és $y$ változó?
  - $x$ és $y + 2$ kifejezések típusa azonos-e?
  - stb.
- Kódgenerálás: utasítások generálása a tárgyprogram nyelvén, amelyek megvalósítják az értékadást.

A szemantikus elemzéshez és a kódgeneráláshoz nem elég a lexikális elemek sorozata és a szintaxisfa!



## Információáramlás

- A szemantikus elemzés és a kódgenerálás számára szükséges kiegészítő információk:
  - konstansok értéke
  - változók típusa
  - függvények, operátorok szignatúrája
  - a tárgykódba már bekerült változók, függvények címe
  - részkifejezések típusa, hozzárendelt tárgykód
  - stb.

## Információáramlás

- A szemantikus elemzés és a kódgenerálás számára szükséges kiegészítő információk:
  - konstansok értéke
  - változók típusa
  - függvények, operátorok szignatúrája
  - a tárgykódba már bekerült változók, függvények címe
  - részkifejezések típusa, hozzárendelt tárgykód
  - stb.
- Ezeket az információkat két módon tároljuk:
  - szimbólumtábla
  - szemantikus értékek



## Szimbólumtáblában tárolt információk

- szimbólum neve
- szimbólum attribútumai:
  - definíció adatai
  - típus
  - tárgyprogram-beli cím
  - definíció helye a forrásprogramban
  - szimbólumra hivatkozások a forrásprogramban



## Szimbólum neve

- a szemantikus elemző és a kódgenerátor a szimbólumokat a nevükkel azonosítja
- a nevek általában változó hosszúságúak lehetnek
  - érdemes dinamikus memóriában tárolni
  - a tábla csak egy mutatót tartalmaz
  - pl. C++ *std* :: *string* választás esetén pont ez történik...



## Definíció adatai

- változó
  - típus
  - módosító kulcsszavak: const, static ...
  - címe a tárgyprogramban (függ a változó tárolási módjától)



## Definíció adatai

- változó
  - típus
  - módosító kulcsszavak: const, static ...
  - címe a tárgyprogramban (függ a változó tárolási módjától)
- függvény, eljárás, operátor
  - paraméterek típusa
  - visszatérési típus
  - módosítók
  - címe a tárgyprogramban



## Definíció adatai

- típus (típusleíró, típusdeszkriptor)
  - egyszerű típusok: méret
  - rekord: mezők nevei és típusleírói
  - tömb: elem típusleírója, index típusleírója, méret
  - intervallum-típus: elem típusleírója, minimum, maximum
  - unio-típus: a lehetséges típusok leírói, méret

## Tárgyprogram-beli cím

- A változók címét a definiáláskor határozza meg a fordító.
  - globális és statikus változók:
    - az adatszegmens kezdetétől számított relatív cím
    - az utoljára elhelyezett változó utáni szabad helyre
  - lokális változók (alprogramban vagy belső blokkban deklarálva):
    - a veremben kap helyet
    - az aktuális veremkereten belüli relatív cím
    - az ebben a blokkban utoljára deklarált változó után
  - dinamikusan foglalt változók
    - a heap memóriában kapnak helyet
    - címük csak a program futásakor derül ki
    - mutatókkal hivatkozunk rájuk, csak azonak van „neve"
    - deklarált mutatók az előző két kategóriába esnek



## Tárgyprogram-beli cím

- Az alprogramok tárgyprogram-beli címét is definiáláskor határozza meg a fordító.
  - kódszegmensen belüli relatív cím
  - az utoljára definiált alprogram utáni szabad helyre



## Hivatkozások a szimbólumra

- deklaráció, definíció és hivatkozások sorainak száma a forrásprogramban
- hasznos lehet:
  - jó hibaüzenetek generálásához
  - kereszthivatkozás-lista létrehozásához („Hol melyik változót használjuk?")



## (Túl) egyszerű példa

**Forrásprogram**

```
1. double d;
2. double fv( int i )
3. {
4.   int j = i*i;
5.   return d*j;
6. }
```

| Név | Fajta | Típus | Param. | Def. | Hivatk. |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |



## (Túl) egyszerű példa

**Forrásprogram**

```
1. double d;
2. double fv( int i )
3. {
4.   int j = i*i;
5.   return d*j;
6. }
```

| Név | Fajta | Típus | Param. | Def. | Hivatk. |
|---|---|---|---|---|---|
| "d" | változó | double | | 1 | |
| | | | | | |
| | | | | | |
| | | | | | |



## (Túl) egyszerű példa

**Forrásprogram**

```
1. double d;
2. double fv( int i )
3. {
4.   int j = i*i;
5.   return d*j;
6. }
```

| Név | Fajta | Típus | Param. | Def. | Hivatk. |
|---|---|---|---|---|---|
| "d" | változó | double | | 1 | |
| "fv" | függvény | double | int | 2 | |
| "i" | paraméter | int | | 2 | |
| | | | | | |

## (Túl) egyszerű példa

**Forrásprogram**

```
1. double d;
2. double fv( int i )
3. {
4.   int j = i*i;
5.   return d*j;
6. }
```

| Név | Fajta | Típus | Param. | Def. | Hivatk. |
|---|---|---|---|---|---|
| "d" | változó | double | | 1 | |
| "fv" | függvény | double | int | 2 | |
| "i" | paraméter | int | | 2 | 4 |
| "j" | változó | int | | 4 | |



## (Túl) egyszerű példa

**Forrásprogram**

```
1. double d;
2. double fv( int i )
3. {
4.   int j = i*i;
5.   return d*j;
6. }
```

| Név | Fajta | Típus | Param. | Def. | Hivatk. |
|---|---|---|---|---|---|
| "d" | változó | double | | 1 | 5 |
| "fv" | függvény | double | int | 2 | |
| "i" | paraméter | int | | 2 | 4 |
| "j" | változó | int | | 4 | 5 |



## A szimbólumtábla műveletei

- keresés
  - szimbólum használatakor
- beszúrás
  - új szimbólum megjelenésekor
  - tartalmaz egy keresést is: „Volt-e már deklarálva?"

Ezek hatékonysága nagy mértékben befolyásolja a fordítóprogram sebességét!



## Hatókör, láthatóság, élettartam

- hatókör: „Ahol a deklaráció érvényben van."



## Hatókör, láthatóság, élettartam

- hatókör: „Ahol a deklaráció érvényben van."
- láthatóság: „Ahol hivatkozni lehet rá a nevével."
  - része a hatókörnek
  - az elfedés miatt lehet kisebb, mint a hatókör



## Hatókör, láthatóság, élettartam

- hatókör: „Ahol a deklaráció érvényben van."
- láthatóság: „Ahol hivatkozni lehet rá a nevével."
  - része a hatókörnek
  - az elfedés miatt lehet kisebb, mint a hatókör
- élettartam: „Amíg memóriaterület van hozzárendelve."

## Példa: hatókör

```
int x = 2;
cout << x << endl;
{
  cout << x << endl;
  int x = 3;
  cout << x << endl;
}
cout << x << endl;
```



## Példa: láthatóság

```
int x = 2;
cout << x << endl;
{
  cout << x << endl;
  int x = 3;
  cout << x << endl;
}
cout << x << endl;
```



## Hatókör és láthatóság kezelése szimbólumtáblával

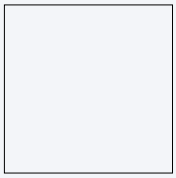
- A szimbólumokat egy verembe tesszük.
- Keresés:
  - a verem tetejéről indul
  - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.



## Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
  - a verem tetejéről indul
  - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```

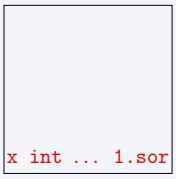




## Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
  - a verem tetejéről indul
  - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```

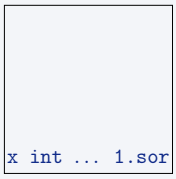





## Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
  - a verem tetejéről indul
  - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```



## Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
  - a verem tetejéről indul
  - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

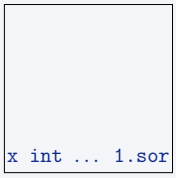
```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```




## Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
  - a verem tetejéről indul
  - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```

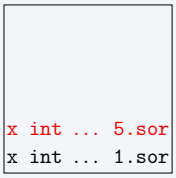





## Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
  - a verem tetejéről indul
  - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```

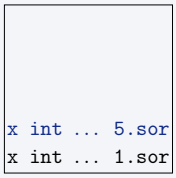





## Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
  - a verem tetejéről indul
  - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```

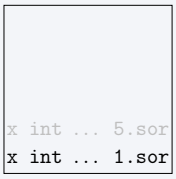





## Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
  - a verem tetejéről indul
  - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```

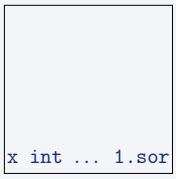





## Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
  - a verem tetejéről indul
  - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```



## Verem szimbólumtábla

- „Meddig kell a szimbólumokat törölni a blokk végén?"
  - Számon kell tartani a blokk kezdetét a szimbólumtábla vermében!



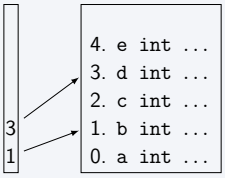
## Verem szimbólumtábla

- „Meddig kell a szimbólumokat törölni a blokk végén?"
  - Számon kell tartani a blokk kezdetét a szimbólumtábla vermében!
- Ötlet: blokk-index vektor
  - ez is egy verem
  - blokk kezdetén: *set*
    - az új blokk első szimbólumára mutató pointert teszünk a blokk-index vektorba
  - blokk végén: *reset*
    - a blokk-index vektor tetején lévő pointer mutatja, hogy meddig kell törölni az elemeket
    - a szimbólumok törlése után a blokk-index vektor legfelső elemét is törölni kell
- Név: verem szimbólumtábla



## Példa

```
int a
{
  int b;
  int c;
  {
    int d;
    int e;
    ...
  }
}
```






## Hatékonyságnövelés

- verem-szimbólumtábla:
  - keresés: lineáris
  - beszúrás (mivel tartalmaz egy keresést is): lineáris



## Hatékonyságnövelés

- verem-szimbólumtábla:
  - keresés: lineáris
  - beszúrás (mivel tartalmaz egy keresést is): lineáris
- hatékonyabb adatszerkezetek:
  - kiegyensúlyozott fa:
    keresés és beszúrás is logaritmikus
  - hash-tábla:
    keresés és beszúrás is konstans műveletigényű
    (jól megválasztott hash-függvény esetén)



## Faszerkezetű szimbólumtábla

- Minden blokkhoz egy fa tartozik.
  - A szimbólumtábla sorai a fa csúcsaiban helyezkednek el.
- A fákat egy veremben tároljuk.
  - Új blokk kezdetén egy új (üres) fát teszünk a verembe.
  - A blokk szimbólumait ebbe a fába láncoljuk be.
  - Időnként kiegyensúlyozzuk a fát.
    (Ha a nyelvben van külön deklarációs része a blokkoknak, akkor elég ennek a végén kiegyensúlyozni.)
  - A blokk végén a teljes fát törölni kell a veremből.
- Keresés:
  - A verem tetején lévő fában kezdjük.
  - Továbblépünk az előző fára, ha addig nem volt találat.
  - Az első előfordulásig keresünk.

## Hash-szerkezetű szimbólumtábla

- szimbólumtábla sorai: veremben (mint a verem-szimbólumtábla esetén)
  - van blokk-index vektor is
- hash-függvény: a szimbólum nevét egy hash-értékre képezi le
- hash-tábla: a hash értékekhez hozzárendeli a legutóbbi ilyen hash-kódú szimbólum pozícióját a veremben
  - ha még nem volt ilyen kódú szimbólum: nullpointer vagy speciális érték
  - ha több ilyen kódú szimbólum is van: láncolni kell őket a legutóbbitól a régebbiek felé



## Keresés a hash-szerkezetű szimbólumtáblában

- a hash-függvénnyel meghatározzuk a keresendő szimbólum hash-értékét
- a hash-táblából megkapjuk az ilyen kódú szimbólumok láncolt listáját
- végigmegyünk a listán az első találatig



## Beszúrás a hash-szerkezetű szimbólumtáblába

- a szimbólumot (és attribútumait) a verem tetejére helyezzük
- a hash-függvénnyel meghatározzuk a beszúrandó szimbólum hash-értékét
- a hash táblában a kapott hash értékhez bejegyezzük a verem-beli pozíciót
- ha volt már ilyen hash-kódú szimbólum, akkor az új szimbólumhoz beláncoljuk



## Törlés a hash-szerkezetű szimbólumtáblából

- blokk végén a blokk-index vektor tetején lévő elem mutatja, hogy meddig kell törölni a szimbólumokat
- egy szimbólum törlése:
  - előállítjuk a hash-értékét
  - a hash-táblába a kapott hash-értékhez beírjuk a törlendő elemhez láncolt következő elem pozícióját (ha nincs hozzá láncolva elem, akkor a hash táblába az üres lista jelzés kerül)
  - eltávolítjuk az elemet a verem tetejéről
- a blokk összes szimbólumának törlése után a blokk-index vektor legfelső elemét is töröljük



## Minősített nevek kezelése

```
namespace a
{
  int x = 1;
  namespace b
  {
    int x = 2;
    void print()
    {
      cout << a::x << x << endl;
    }
  }
}
```



## Minősített nevek kezelése

```
namespace a
{
  int x = 1;
  namespace b
  {
    int x = 2;
    void print()
    {
      cout << a::x << x << endl;
    }
  }
}
```

A (külső) $x$ és $b$ szimbólumokhoz fel kell jegyezni a szimbólumtáblába, hogy az $a$ névtérhez tartoznak.

## Minősített nevek kezelése

```
namespace a
{
  int x = 1;
  namespace b
  {
    int x = 2;
    void print()
    {
      cout << a::x << x << endl;
    }
  }
}
```

A (belső) *x* és *print* szimbólumokhoz fel kell jegyezni a szimbólumtáblába, hogy a *b* névtérhez tartoznak.



## Minősített nevek kezelése

```
namespace a
{
  int x = 1;
  namespace b
  {
    int x = 2;
    void print()
    {
      cout << a::x << x << endl;
    }
  }
}
```

A nem minősített nevű szimbólumok keresése nem változik.



## Minősített nevek kezelése

```
namespace a
{
  int x = 1;
  namespace b
  {
    int x = 2;
    void print()
    {
      cout << a::x << x << endl;
    }
  }
}
```

Az *a* :: *x* szimbólum keresésekor olyan *x*-et kell keresni a szimbólumtáblában, amihez fel van jegyezve, hogy az *a* névtérben van.



## Szimbólumok importálása

```
namespace a
{
  int x = 1;
}
using namespace a;
int main()
{
  cout << x << endl;
  return 0;
}
```

- A névterek szimbólumait a veremből nem törölni kell, hanem feljegyezni egy másik tárterületre.
- A *using* direktíva használatakor az importált névtér szimbólumait be kell másolni a verembe (vagy legalább hivatkozást tenni a verembe erre a névtérre).



## Osztályok

- A rekordokhoz hasonló: típusleírót kell készíteni hozzá.
- Láthatóság szabályozása: a mezőkhöz és tagfüggvényekhez fel kell jegyezni, hogy milyen a láthatóságuk (*public*, *protected*, *private*).
- Az osztályok névteret is alkotnak: (statikus) adattagjai minősített névvel is elérhetők, ilyenkor az előbb látott technikát lehet használni.