Programski prevodioci 1

Konstrukcija tabele simbola za Mikrojavu, prvi deo – objekti i opsezi

- Uvod
- Organizacija tabele simbola
- Predstavljanje informacija o objektima
- Predstavljanje informacija o opsezima imena
- Operacije nad tabelom simbola

Uloga tabele simbola u kompajleru

- Pamćenje informacija o svim deklarisanim objektima:
  - Ime, tip, vrednost (za konstante), adresa (za promenljive i metode), parametri (za metode), itd
- Nalaženje informacija o korišćenim objektima
  - Mapiranje imena $\rightarrow$ tip, vrednost, adresa,...

Preliminarni interfejs tabele simbola

- Prema drugim delovima kompajlera

```
class Table {
   static Object insert(String name,...);
   static Object find(String name,...);
}
```

- Uvod
- Organizacija tabele simbola
- Predstavljanje informacija o objektima
- Predstavljanje informacija o opsezima imena
- Operacije nad tabelom simbola

Interna organizacija tabele simbola

- Dinamička struktura podataka: skup objektnih i strukturnih zapisa
- Objektni zapisi (instance klase **Object**) čuvaju informaciju o deklarisanim imenima
- Strukturni zapisi (instance klase **Struct**) čuvaju informacije o tipovima u programu (na njih ukazuju objektni zapisi)

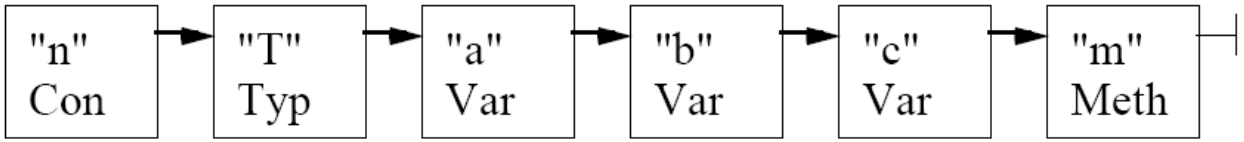
Organizacija tabele simbola: moguće strukture podataka

- Jednostruko ulančana (neuređena) lista
- Binarno leksički uređeno stablo
- Heš tabela

TS u vidu jednostruko ulančane liste

- Primer deklaracija mikroJave:

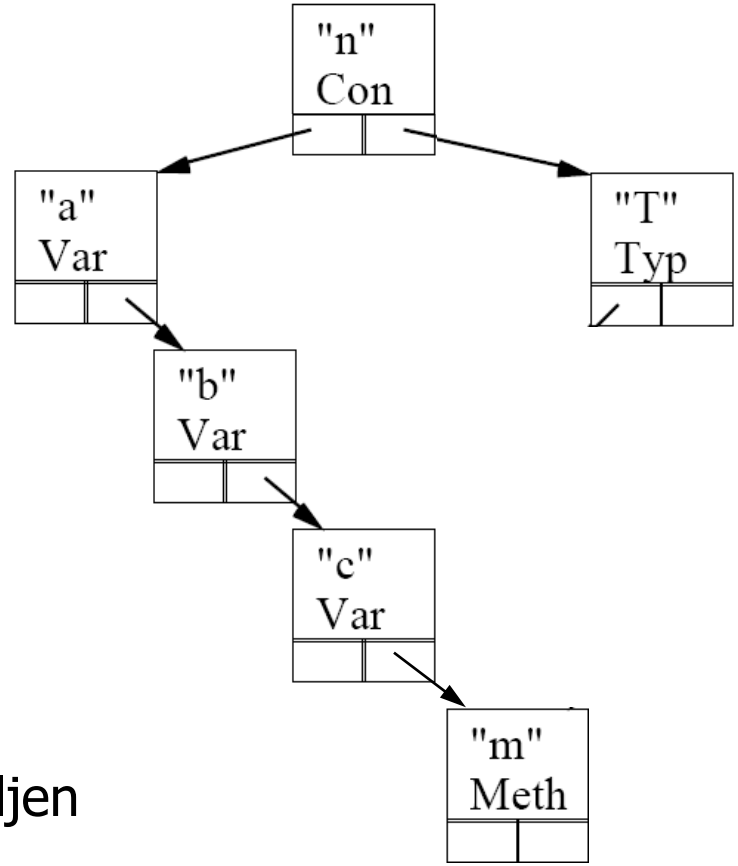
```
final int n=10;
class T {…}
int a, b, c;
void m(){…}
```



- Prednosti: jednostavnost, zapamćen redosled deklarisanja
- Nedostatak: sporost pretrage ako kada ima mnogo imena
- (u nastavku predavanja o TS biće usvojena ova realizacija)

TS u vidu binarnog stabla

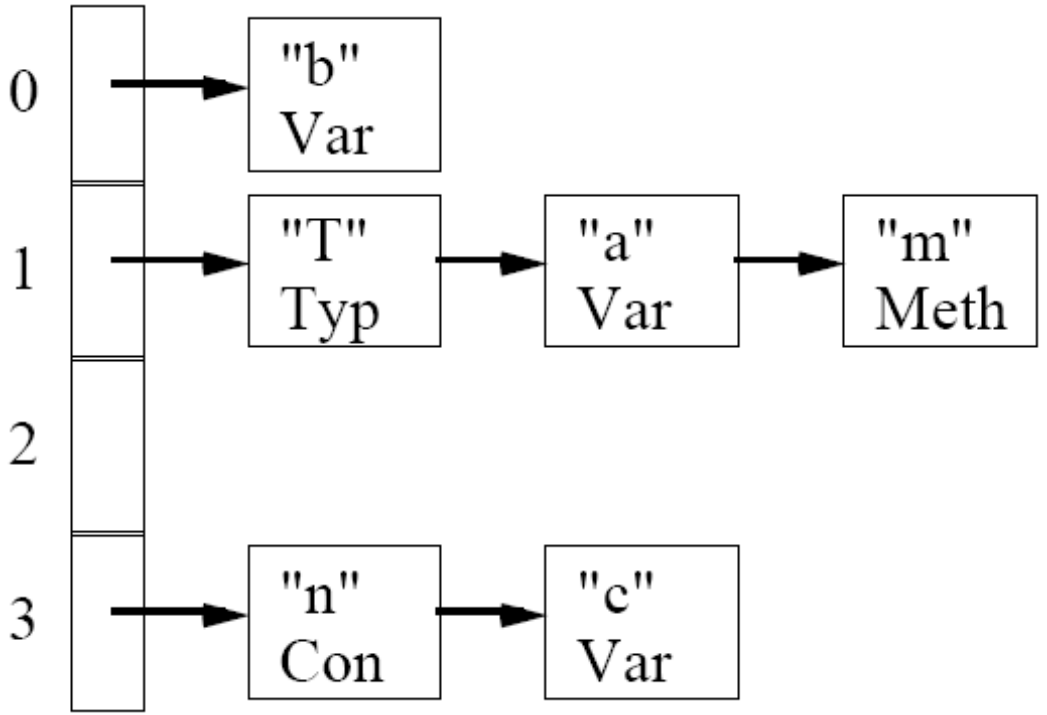
```
final int n=10;
class T {…}
int a, b, c;
void m(){…}
```



- Prednosti: brža pretraga (log n)
- Nedostaci: veća potrošnja prostora i vremena pri ažuriranju (isplati se samo ako ima mnogo deklaracija)
- Originalni redosled deklaracija je izgubljen

TS u vidu heš tabele

```
final int n=10;
class T {…}
int a, b, c;
void m(){…}
```



- Prednost: brzina
- Nedostaci: komplikovanije u odnosu na listu, redosled deklaracija se gubi

- Uvod
- Organizacija tabele simbola
- Predstavljanje informacija o objektima
- Predstavljanje informacija o opsezima imena
- Operacije nad tabelom simbola

Zapisi o objektima

- Svako deklarisano ime se pamti se u jednom objektnom zapisu
- Vrste objekata:

```
static final int
  Con = 0, // constants
  Var = 1, // variables, parameters
  Typ = 2, // types
  Meth = 3, // methods
  Fld = 4; // fields
```

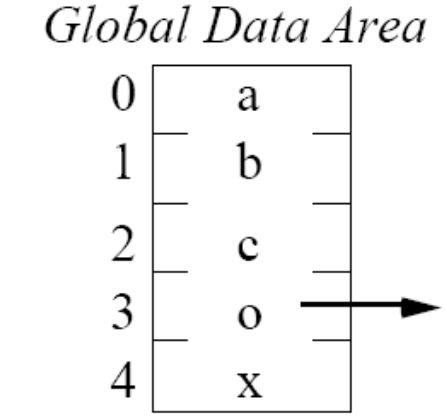
Informacije o objektu

- Za sve vrste objekata:
  - ime, vrsta, tip, sledeći zapis
- Dodatne specifične informacije:
  - Za konstante: vrednost
  - Za promenljive: adresa, nivo deklarisanosti (lokalne, globalne)
  - Za polja klase: pomeraj
  - Za metode: adresa, broj parametara, lista formalnih parametara i lokalnih objekata

Informacije o objektu: adrese

- Globalnih promenljivih
  - U mjVM se čuvaju u oblasti globalnih podataka:

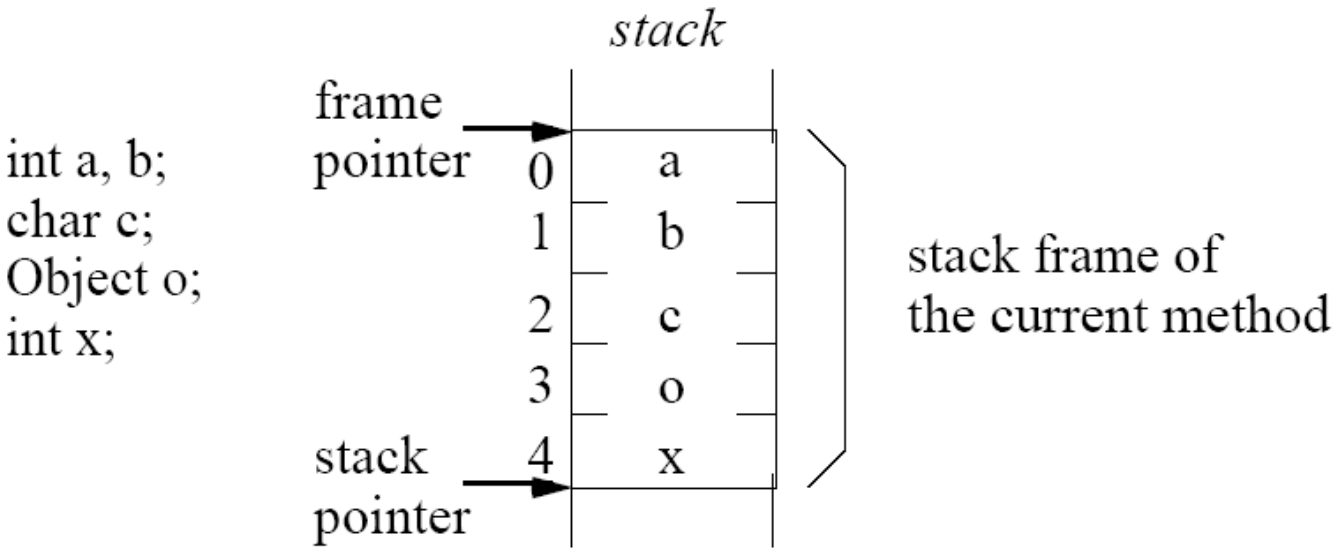
```
int a, b;
char c;
Object o;
int x;
```



  - Svaka mJ promenljiva zauzima po 1 memorijsku reč
  - Adrese promenljivih su relativne u odnosu na početak oblasti globalnih podataka
  - Adrese se dodeljuju sukcesivno

Informacije o objektu: adrese

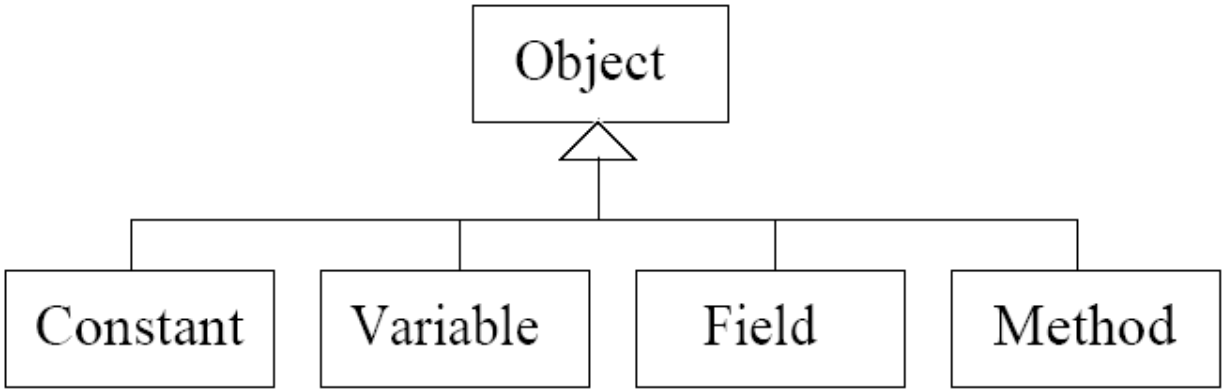
- Lokalnih promenljivih
  - U mjVM se čuvaju na steku poziva (metoda), u aktivacionom zapisu (okviru) metoda:



  - Svaka mJ promenljiva zauzima po 1 memorijsku reč
  - Adrese promenljivih su relativne u odnosu na pokazivač okvira (frame pointer)
  - Adrese se dodeljuju sukcesivno

Struktura podataka za predstavljanje objekata

- Pravolinijsko rešenje u duhu OO:



- Kada se pristupa atributima podklasa zahteva eksplicitnu konverziju (nešto komplikovaniji kod)

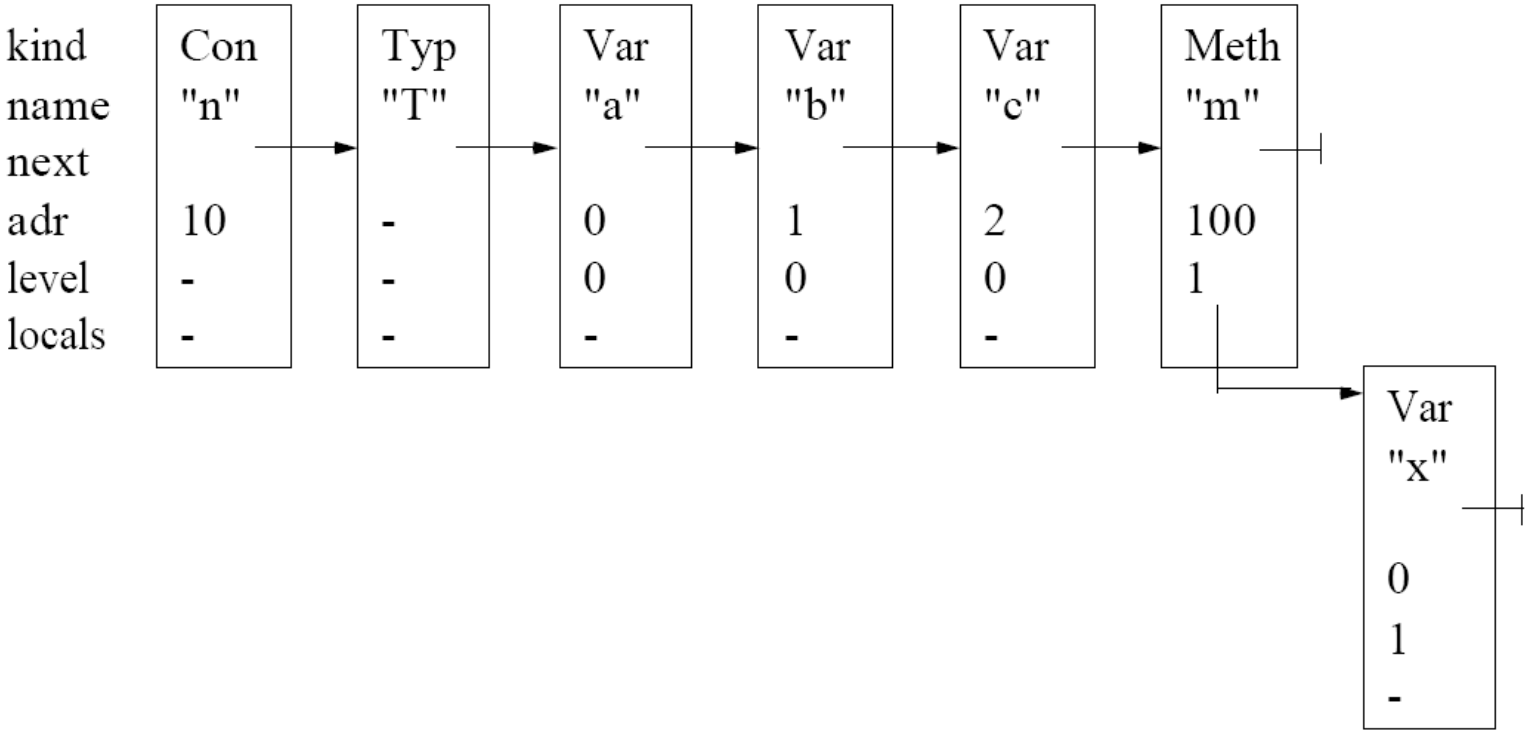
Struktura podataka za predstavljanje objekata

- Za naše potrebe, usvojićemo "ravnu" implementaciju: svi atributi će biti deklarisani u istoj klasi Object

```
class Object {
    static final int Con = 0, Var = 1, Typ = 2, Fld = 3,
    Meth = 4;
    int kind; // Con, Var, Typ, Fld, Meth
    String name;
    Struct typ;
    Obj next;
    int adr; // Con: value; Var, Fld, Meth: address
    intlevel; // Var: declaration level; Meth: no.of.
    parameters
    Obj locals; // Meth: local objects
}
```

Primer

final int **n** = 10;
class **T** {…}
int **a**, **b**, **c**;
void **m** (int **x**) {…}

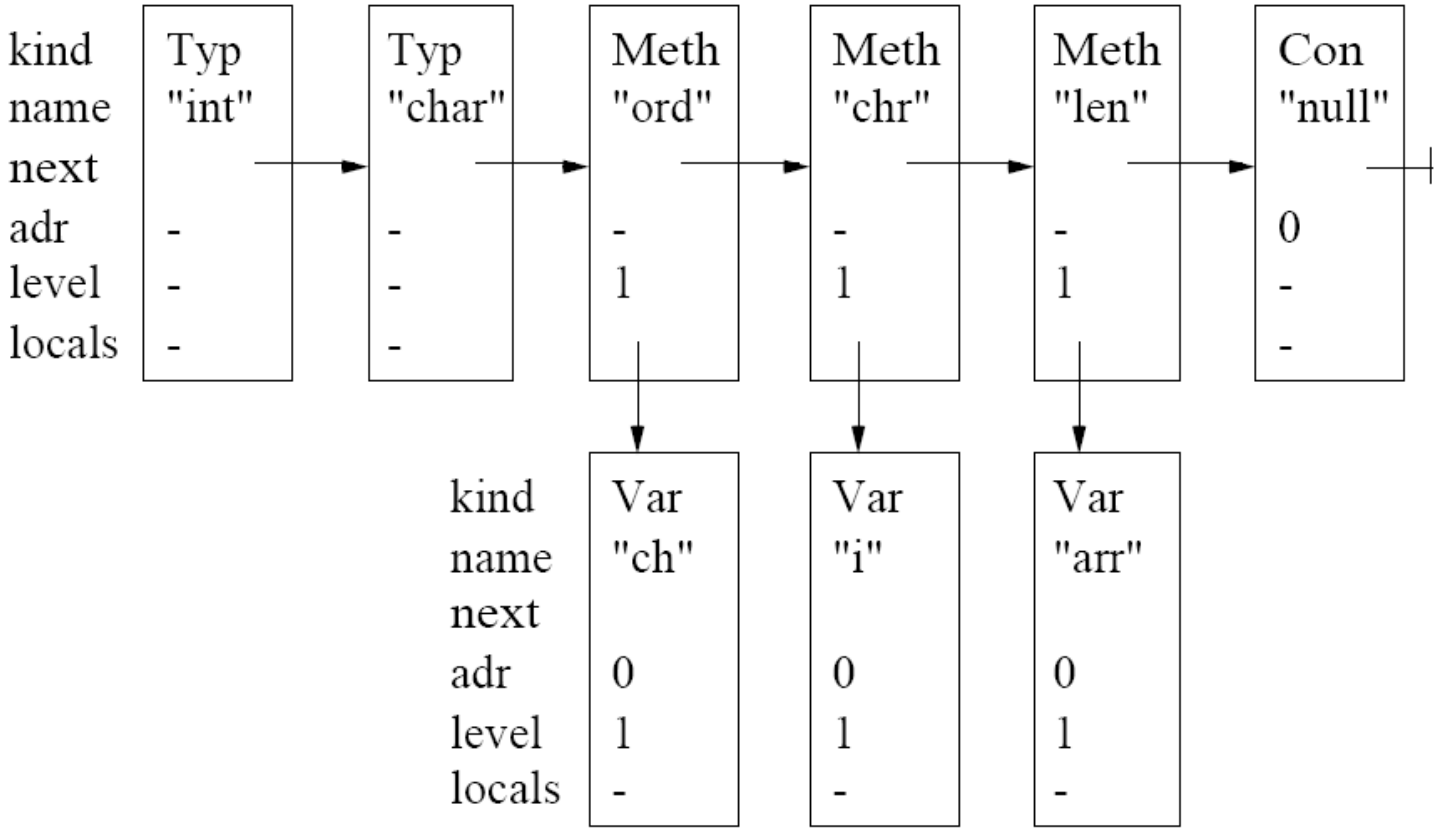


Preddeklarisana imena

- Šta se unapred deklariše u TS (na početku prevođenja):
  - Standardni tipovi: int, char
  - Standardni metodi: ord(ch), chr(i), len(arr)
  - Standardne konstante: null

Struktura podataka za preddeklarisana imena

- Posebna lista objekata:



- **Druga, lošija varijanta**: posebne ključne reči
  - Zahteva posebne terminale u gramatici
  - Komplikovanije, niz specijalnih slučajeva pri pretrazi i ažuriranju TS

- Uvod
- Organizacija tabele simbola
- Predstavljanje informacija o objektima
- Predstavljanje informacija o opsezima imena
- Operacije nad tabelom simbola

Predstavljanje opsega važenja imena u TS

- Hijerarhija opsega imena u programu:
  - Opseg preddeklarisanih imena “universe”
  - Opseg globalnih imena u celom programu
  - Opseg klase (polja i metodi određene klase)
  - Opseg metoda (parametri i lokalna imena)
- Svaki opseg je posebna kolekcija (lista) objekata

Predstavljanje opsega važenja imena u TS

- Za predstavljanje opsega uvodimo zapis Scope:

```
class Scope {
    Scope outer; // to outer scope
    Obj locals; // to the objects of this scope
    int nVars; // number of variables in this scope (for addr.
allocation)
}
Scope topScope; // this is an attribute of class Tab
```

- Kolekcija tipa steka pamti trenutno aktivne opsege u programu

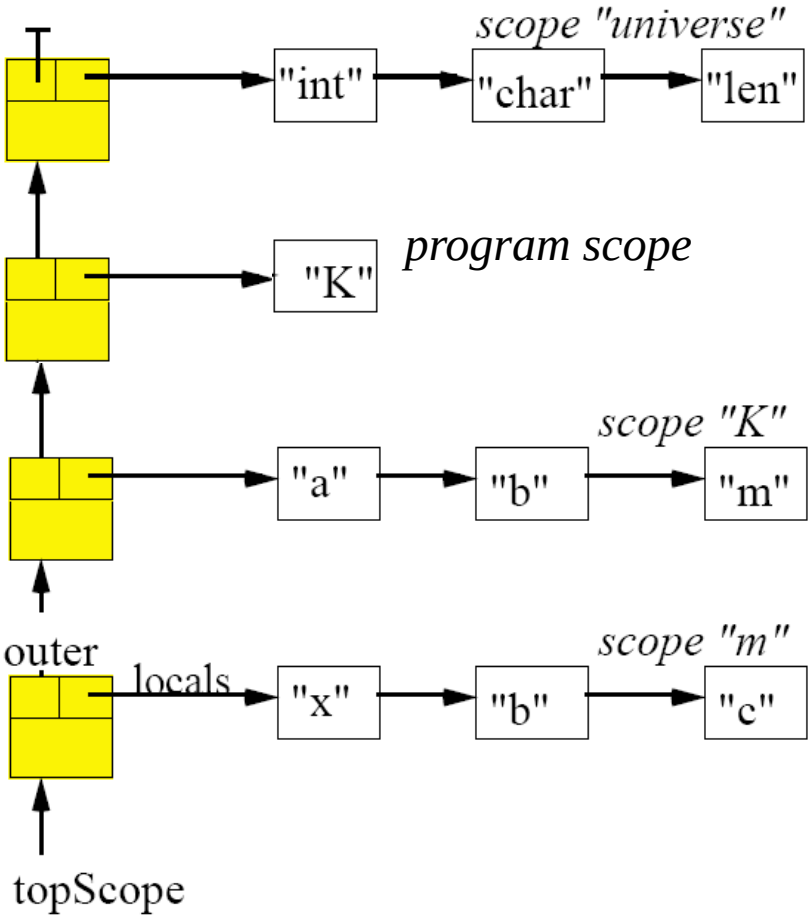
Primer

```
class K
{
   int a, b;

{
   void m   (int x)
      int b, c;

   {
      …
   }

   …
}
```



- Uvod
- Organizacija tabele simbola
- Predstavljanje informacija o objektima
- Predstavljanje informacija o opsezima imena
- Operacije nad tabelom simbola

Pretraga imena u TS

- Prilikom svakog korišćenja nekog simbola poziva se find

```
obj = tab.find(name);
```

- Pretraga počinje u listi topScope
- Ako se ne nađe tamo, pretraga se nastavlja u sledećem okružujućem opsegu (sledeći na steku ispod)

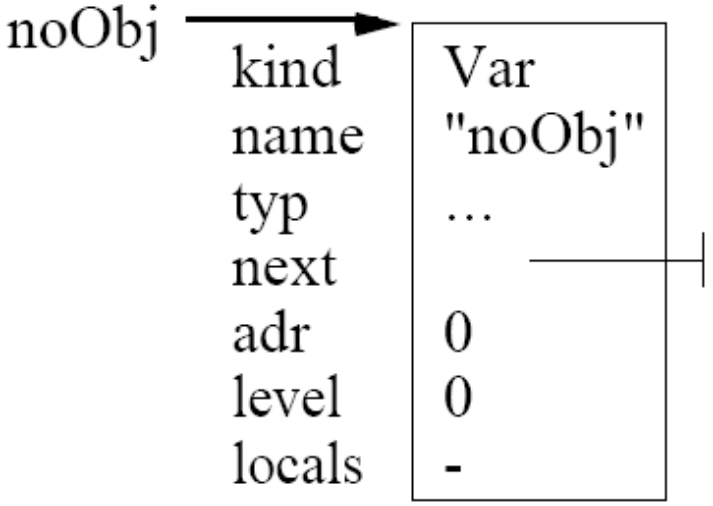
Pretraga imena u TS

- Algoritam:

```
static Object find (String name) {
   for (Scope s = topScope; s != null; s = s.outer)
     for (Object p = s.locals; p != null; p = p.next)
                 if (p.name.equals(name)) return p;
   error(name + " not found");
   return noObj;
}
```

Pretraga imena u TS

- Ako se ime nigde ne nađe vraća se **noObj**
- Radi se o posebnom preddeklarisanom objektu



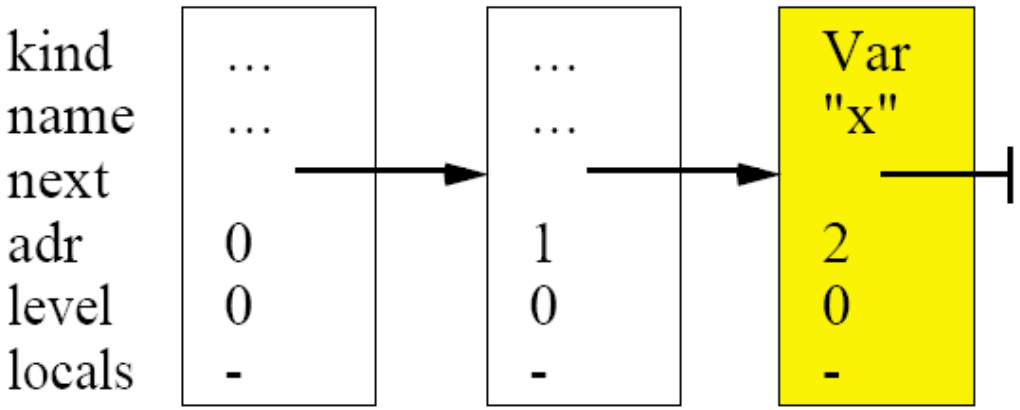
- Ovako se rezultat find lakše koristi, nego da je vraćena **null** vrednost

Ubacivanje imena u TS

- Za svaku deklaraciju pozivamo insert:

  obj = Tab.insert(kind, name, typ);

- Aktivnosti unutar insert():
  - Proverava da li već postoji objekat istog imena (samo) u topScope, ako ima => greška
  - U suprotnom, kreira novi Object(kind, name, type)
  - Dodeljuje mu adresu (prvu slobodnu)
  - Dodeljuje nivo deklaracije (global=0, local=1)
  - Dodaje objekat na kraj liste topScope



Gde se poziva insert()?

- U parseru, unutar smena za deklaracije.

- **Primer**: deklaracija promenljive

VarDecl = Type_{typ} $\text{ident}_{\text{name}}$ (: Tab.insert(Obj.Var, name, typ) :)
{ "," $\text{ident}_{\text{name}}$ (: Tab.insert(Obj.Var, name, typ) :) }.

Obrada opsega

- Dva nova metoda u interfejsu tabele simbola: za otvaranje i zatvaranje opsega

- Za otvaranje opsega:

```
static void openScope() {
   Scope s = new Scope();
   s.outer = topScope; topScope = s;
}
```

- Poziva se kada parser naiđe na početak opsega (početak programa, iza imena klase, ili metoda)
- Na vrh steka opsega stavlja novi opseg

Obrada opsega

- Za zatvaranje opsega:

```
static void closeScope() {
   topScope = topScope.outer;
}
```

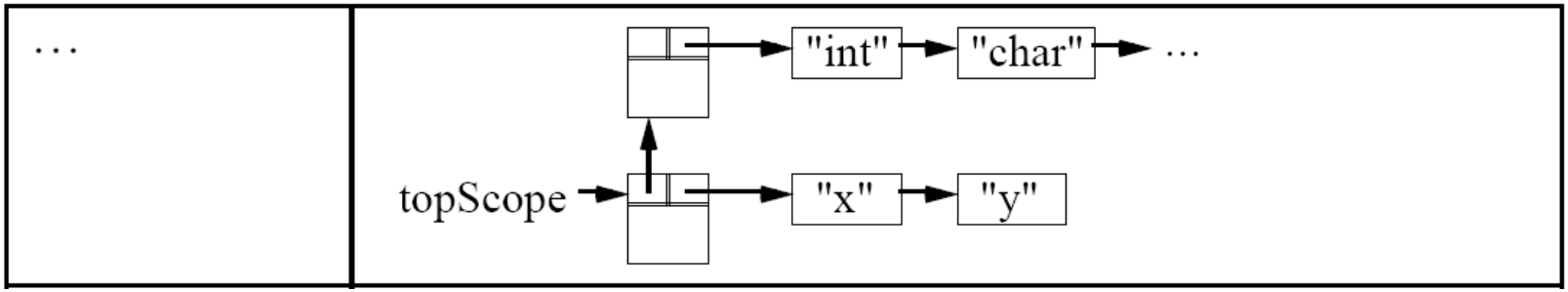
- Poziva se kada parser naiđe na kraj opsega (kraj programa, } od klase ili metoda)
- Sa vrha steka opsega skida tekući opseg

Primer upotrebe

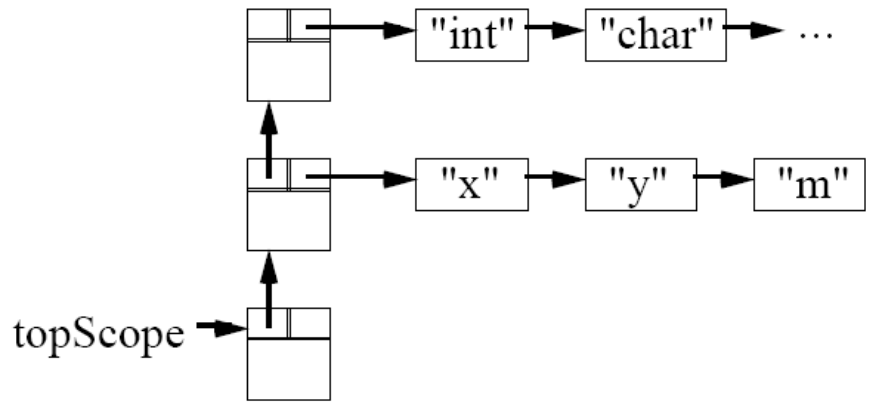
```
MethodDecl
= Type[typ]
    ident[name] (: meth=Tab.insert(Obj.Meth, name, typ);
                   Tab.openScope() :)
...
    "}"          (: meth.locals = Tab.topScope.locals;
                   Tab.closeScope(); :).
```

- Napomena: atribut typ je tipa Struct, name tipa String, meth tipa Object

Primer upotrebe



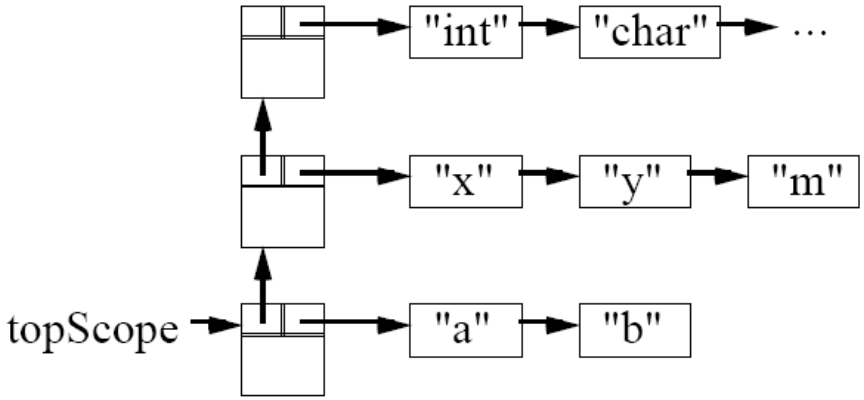
Primer upotrebe

method m	insert("m", …); openScope(); 

Primer upotrebe

```
    int a, b;
{
    …
```

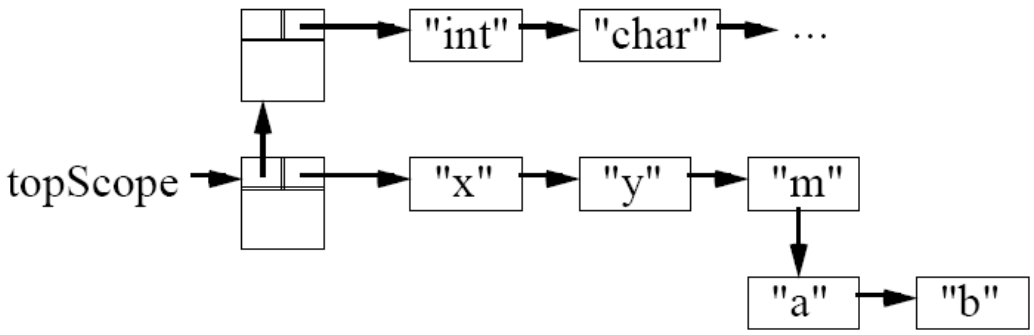
```
insert("a", …);
insert("b", …);
```



Primer upotrebe

```
}
...
```

```
m.locals = topScope.locals;
closeScope();
```



Interfejs tabele simbola (finalno)

```
class Tab {
    static Scope topScope; // current top scope
    static int level; // nesting level of current scope
    static Struct intTyp; // predeclared types
    static Struct charTyp;
    static Struct nullTyp;
    static Struct noTyp;
    static Obj chrObj; // predeclared objects
    static Obj ordObj;
    static Obj lenObj;
    static Obj noObj;
    static Obj insert (int kind, String name, Struct typ);
    static Obj find (String name);
    static void openScope();
    static void closeScope();
}
```