Programski prevodioci 1

Konstrukcija tabele simbola za Mikrojavu, drugi deo – tipovi

- Uvod
- Predstavljanje tipova
- Provere tipova

Uvod

- Svaki programski konstrukt (deklaracija, izraz, iskaz) ima svoj tip
- Vrste tipova:
  - osnovni; u Mikrojavi int, char.
  - izvedeni; U mikrojavi nizovi, klase.
- Tipovi imaju strukturu, tj. osobine:
  - memorijsko zauzeće
  - za izvedene: tipovi elemenata, definicije članova, …

- Uvod
- Predstavljanje tipova
- Provere tipova

Predstavljanje tipova

- Tipove možemo predstavljati tzv. *izrazima tipa* (engl. type expressions).
  - osnovni tip predstavlja izraz tipa
  - ime tipa predstavlja izraz tipa
  - izvedeni tipovi se dobijaju primenom tzv. konstruktora izvedenog tipa na neki izraz tipa.
- Pogodan metod za predstavljanje izraza tipa su grafovi. U slučaju Mikrojave, grafovi tipova sadržaće **Struct** čvorove i po potrebi **Obj** čvorove (npr. za imena članova klasa).

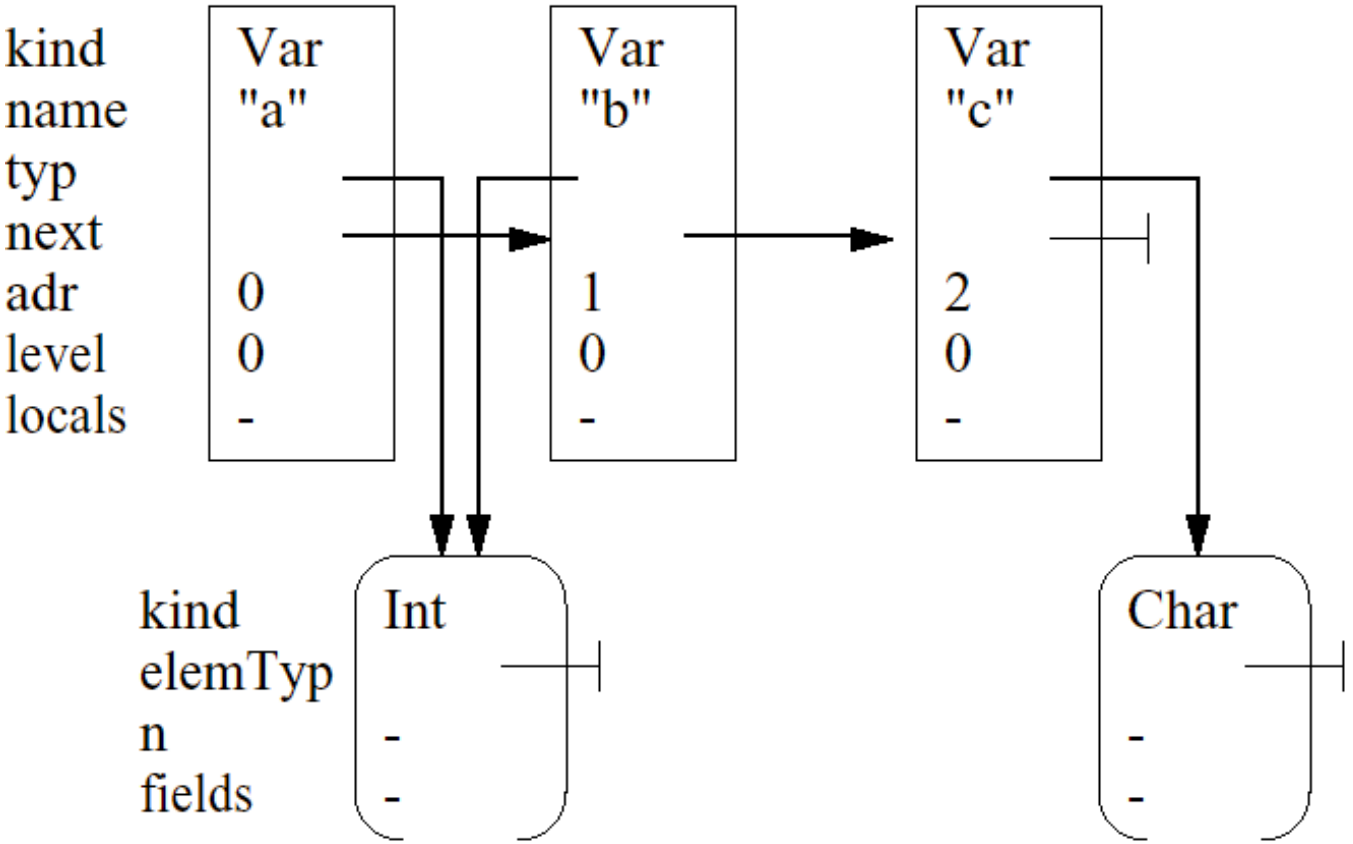
Predstavljanje tipova grafom

- Izgled **Struct** čvora:

```
class Struct {
      static final int // kinds of types
            None = 0, Int = 1, Char = 2,
            Arr = 3, Class = 4;

      int kind; // None, Int, Char, Arr, Class

      Struct elemTyp; // Arr:elem.type,
                      // Class:superclass

      int n; // Class: number of fields
      Obj fields; // Class: list of fields
}
```

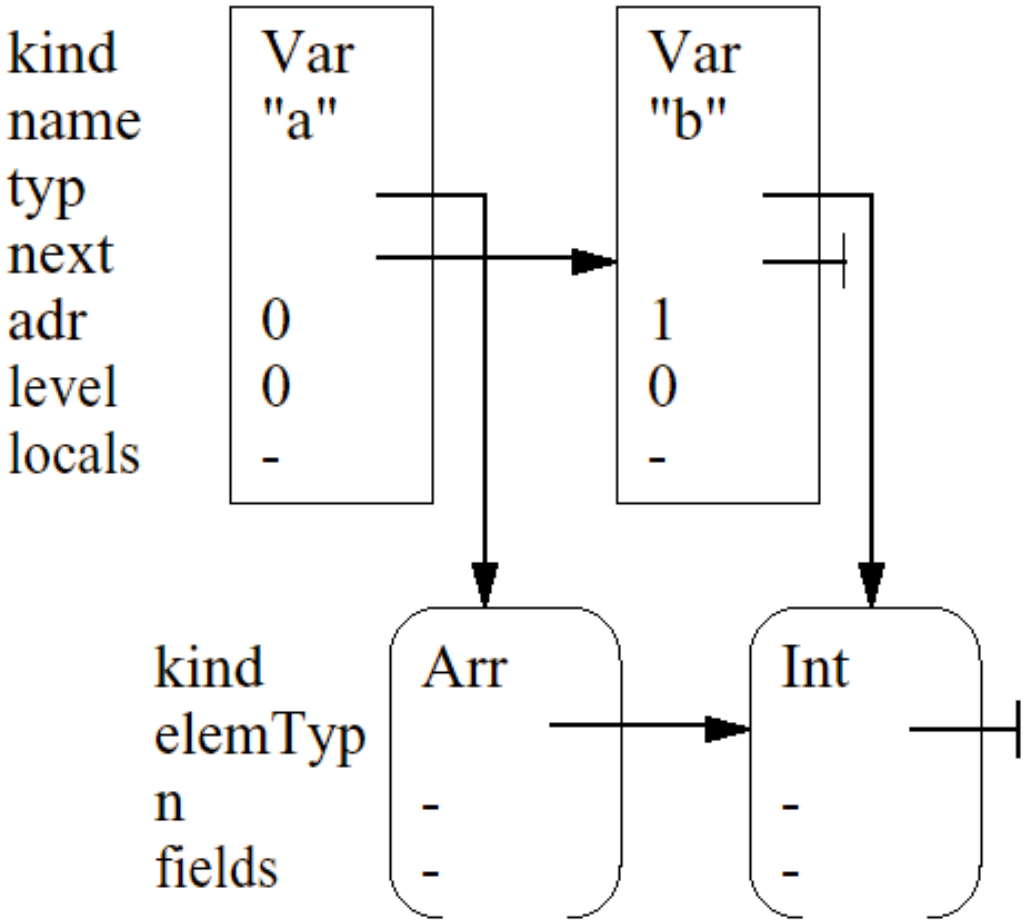
Grafovi osnovnih tipova

```
int a, b;
char c;
```



Predstavljanje tipa niza

```
int[] a;
int b;
```

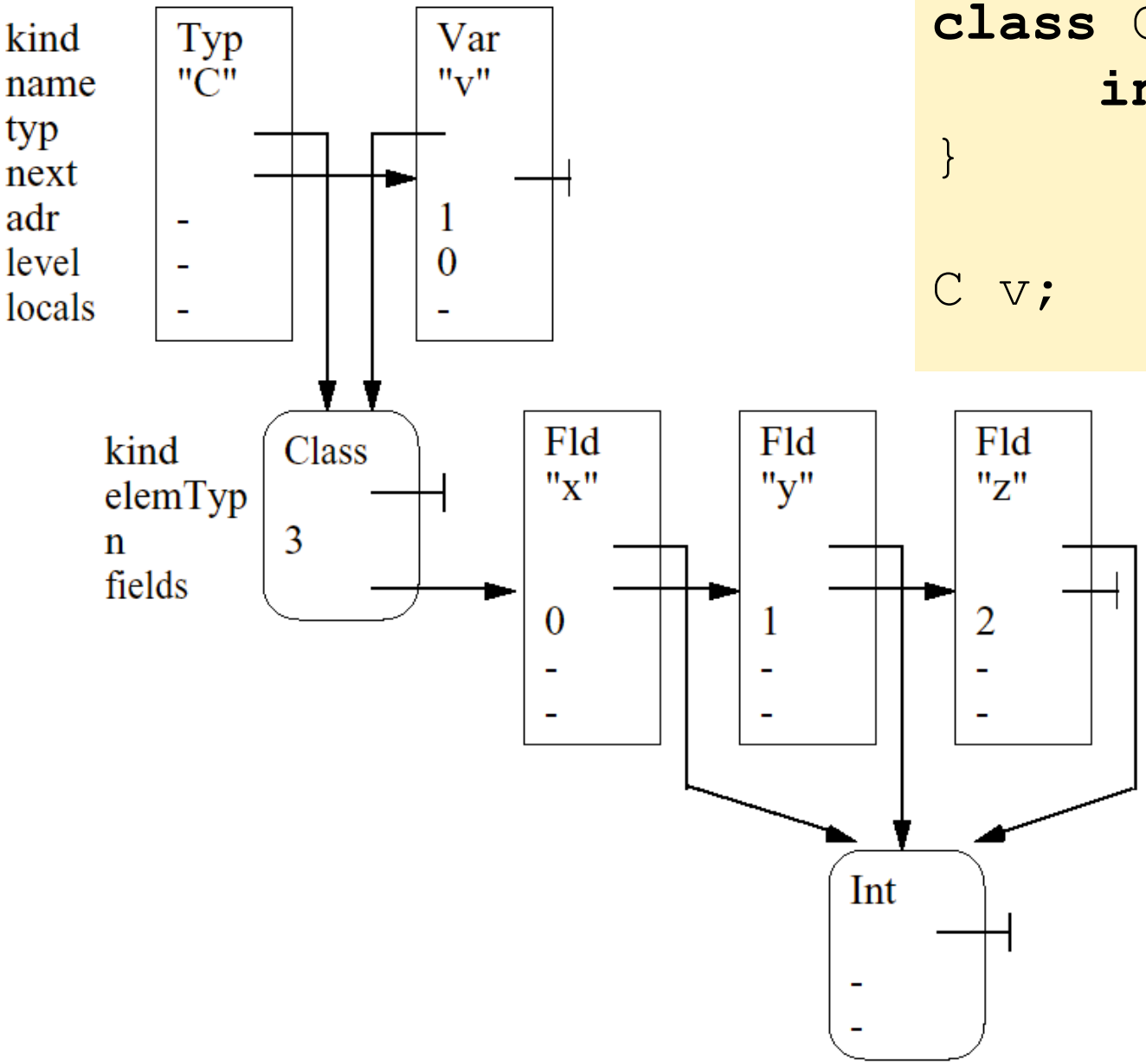


Predstavljanje tipa klase, primer 1

- Ovaj primer je bez metoda i nasleđivanja.
- Kada parser naiđe na deklaraciju klase, on stvara odgovarajući čvor (**Obj**) u tabeli simbola.
- U tom čvoru polje *type* pokazuje na čvor tipa (**Struct**), čije polje *fields* treba da pokazuje na polja klase čija se deklaracija obrađuje.

```
class C {
      int x, y, z;
}

C v;
```

Predstavljanje tipa klase, primer 1



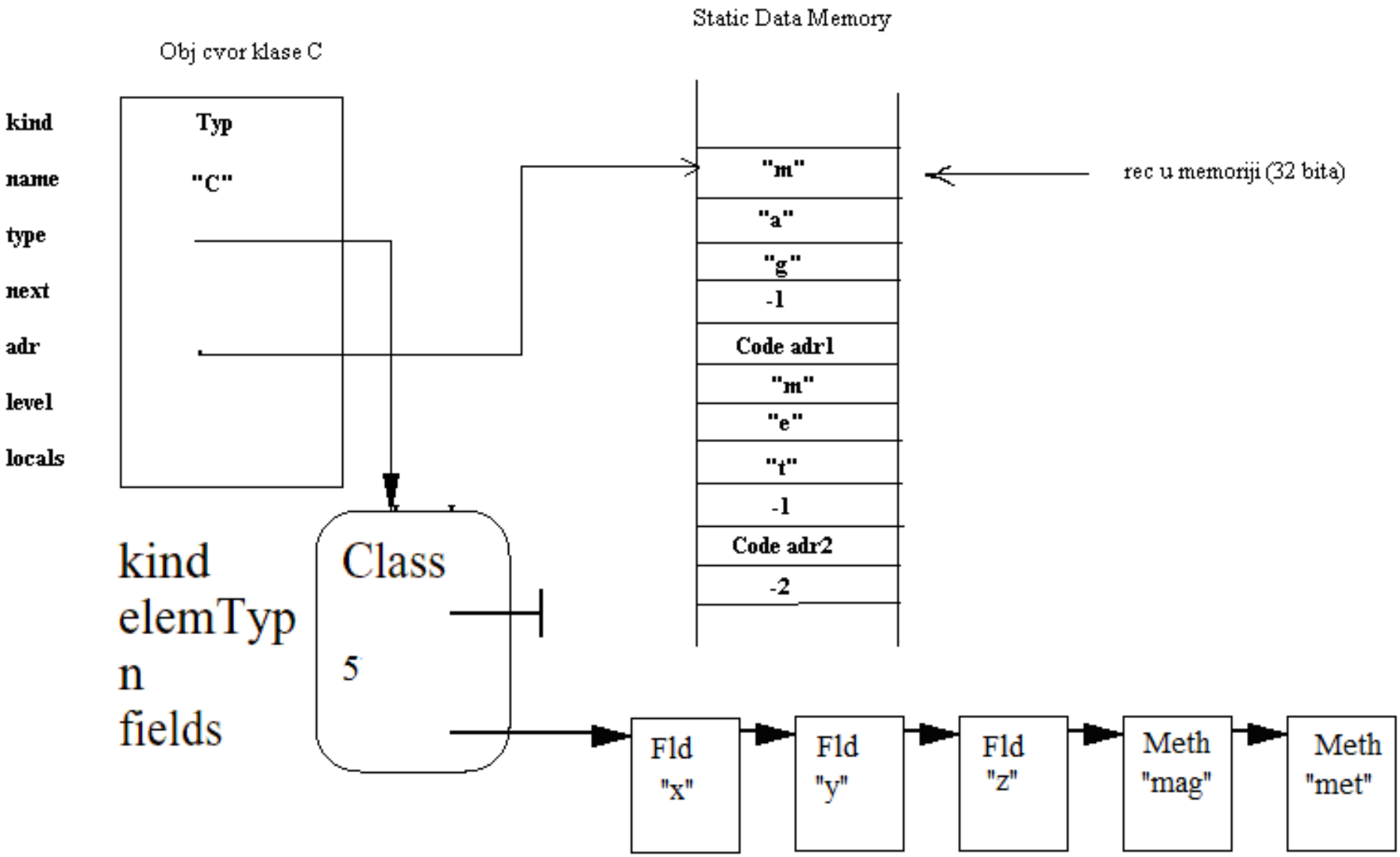

```
class C {
      int x, y, z;
}

C v;
```

Predstavljanje tipa klase, primer 2

- (dinamički metodi)
- Svaka klasa, osim glavne koja predstavlja program, ima virtuelnu tabelu metoda (v-table). V-tabela se čuva u Static Data memoriji. Adresa prve reči v-tabele u Static Data memoriji se pamti u Obj čvoru klase, u tabeli simbola, u polju adr.

```
class C {
      int x,y, z;
      int mag() {
            return 3;
      }
      void met(int q) {
            q=q+1;
      }
}
```

Predstavljanje tipa klase, primer 2



Predstavljanje tipa klase, primer 2

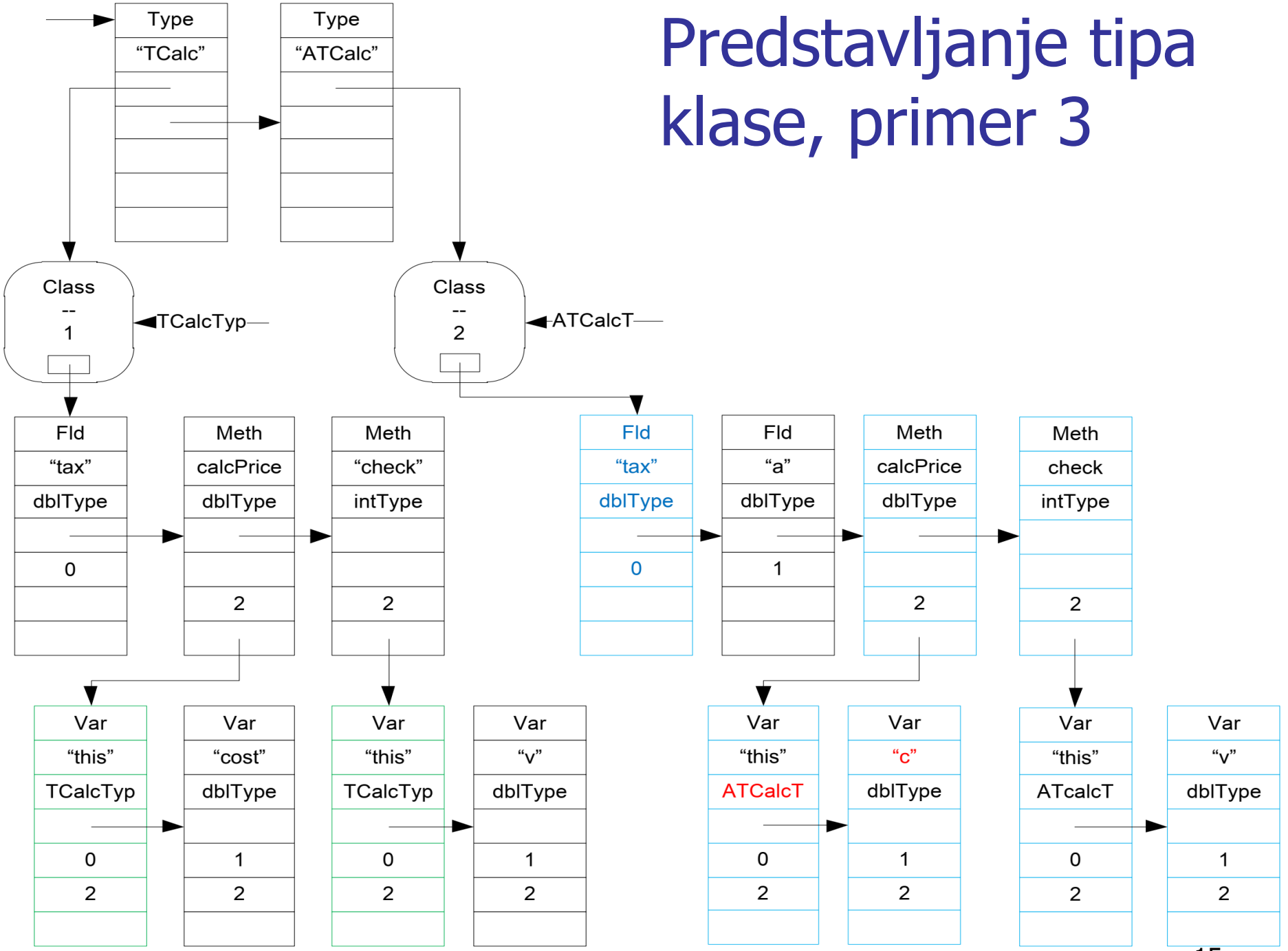
- Code adr1 je adresa početka koda u programskoj memoriji mikrojava virtuelne mašine koji odgovara metodi *mag*, slično, Code adr2 je adresa početka koda u programskoj memoriji koji odgovara metodi *met*.
- U ovom primeru v-tabela zauzima 11 reči u Static Data oblasti virtuelne mašine. Broj -2 označava kraj tabele, a -1 razdvaja imena metoda.
- Prvi formalni argument **Obj** čvora dinamičkog metoda je uvek **this**. Argumentu **this** odgovara isti **Struct** čvor kao i klasi unutar koje se metoda deklariše.

Predstavljanje tipa klase, primer 3

- Ovaj primer ilustruje nasleđivanje klasa

```
class TCalc {
   double tax;
   {    double calcPrice(double cost) { return cost*(1+tax); }
        int check(double v) { return d>0; }
   }
}
class ATCalc extends TCalc {
   double a;
   {    double calcPrice(double c) { return (cost*a)*(1*tax); }
   }
}
```

Predstavljanje tipa klase, primer 3



Predstavljanje tipa klase, primer 3

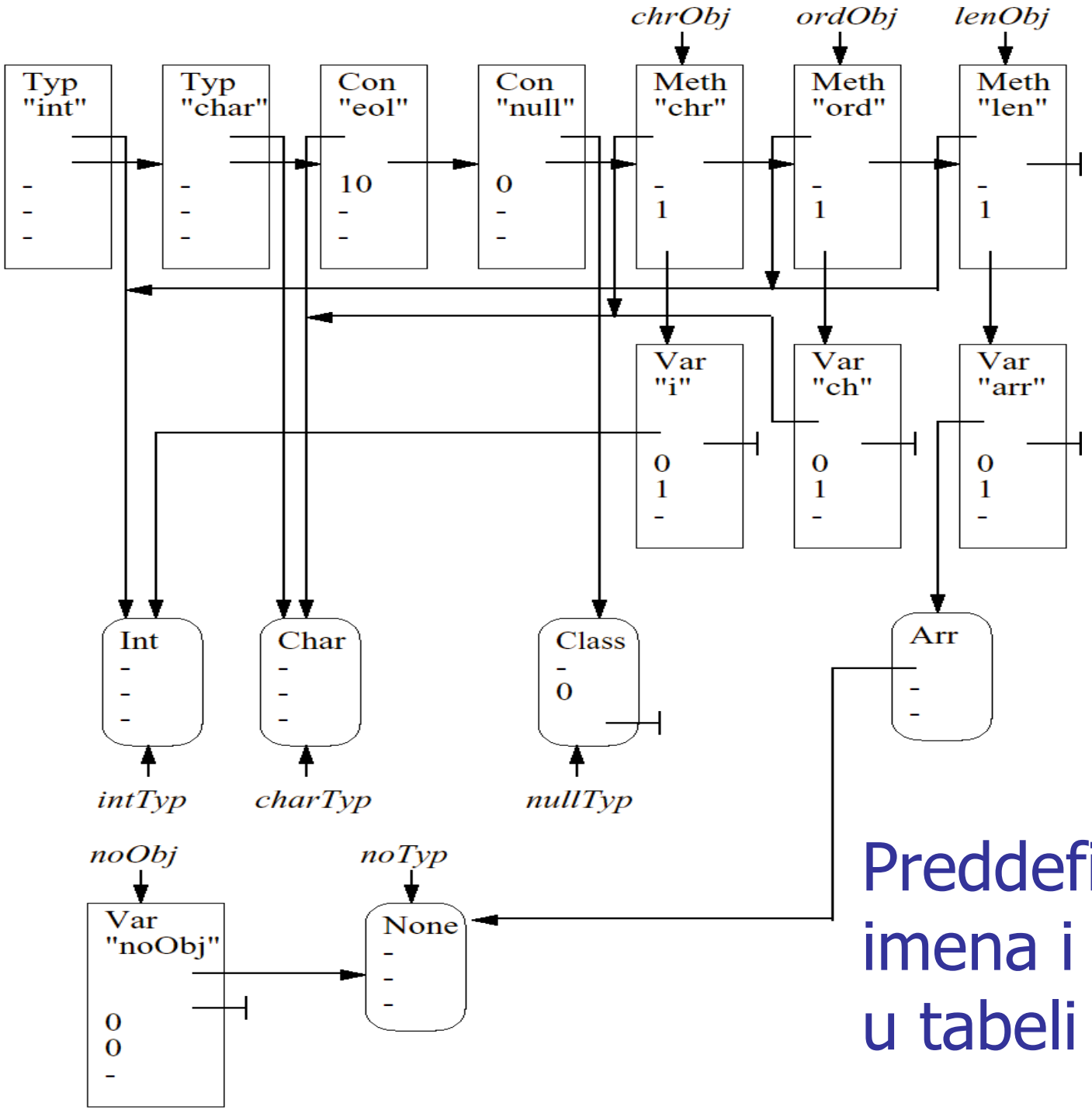
- U slučaju da klasa nasleđuje drugu klasu, prvo se u tabeli simbola pronalazi čvor natklase, pa se prave kopije čvorova koji predstavljaju polja (ne metode) natklase, i te kopije se redom stavljaju u tabelu simbola kao polja potklase.
- Posle toga se obrađuju polja koja deklariše sama potklasa, i stvaraju novi **Obj** čvorovi koji se nadovezuju na čvorova nasleđenih polja (extension on the right).

Predstavljanje tipa klase, primer 3

- Kada su sva polja stavljena u tabelu simbola dolazi na red ubacivanje **Obj** čvorova dinamičkih metoda. Ti čvorovi se nadovezuju na čvorove polja. Opet će se prvo pogledati čvorovi natklase u tabeli simbola, napraviti kopije čvorova njenih metoda (i kopije formalnih argumenata i lokalnih promenljivih) i redom nadovezati na čvorove polja.

Predstavljanje tipa klase, primer 3

- Zatim se obrađuju metode koje deklariše potklasa. Potpis svake metode deklarisane u potklasi se prvo uporedi sa potpisima metoda nasleđenih iz natklase, da bi se utvrdilo da li je metoda redefinisana. Ako nije, za tu metodu se pravi novi čvor u tabeli simbola, a ako jeste neće se praviti novi čvor, već će biti modifikovan postojeći čvor (naravno u grupi čvorova potklase) koji odgovara toj metodi.
- Čvor se modifikuje zato što je moguće da redefinisana metoda nema ista imena formalnih argumenata i broj i tipove lokalnih promenljivih kao metoda koju redefiniše. Takođe redefinisana metoda ima zaseban kod u programskoj memoriji, pa treba promeniti i polje adr čvora metode.



Preddefinisana imena i njihovi tipovi u tabeli simbola

- Uvod
- Predstavljanje tipova
- Provere tipova

Sistem tipova

- **Sistem tipova** je skup pravila kako se dodeljuju tipovi različitim delovima programa (deklaracijama, izrazima, iskazima).
- Za mikrojavu sistem tipova definisan je sekcijama specifikacije: *A.3 Semantika i A.4 Kontekstni uslovi*.
- Proverom tipa implementira se određeni sistem tipova. Provera može biti:
  - **statička**, ako je vrši kompajler, ili
  - **dinamička**, ako se izvodi u vreme izvršavanja programa (pod uslovom da objekti u programu nose informaciju o tipu u vreme izvršavanja). Primer: C++ RTTI – Run time type information, typeid i dynamic_cast operatori.

Sistem tipova

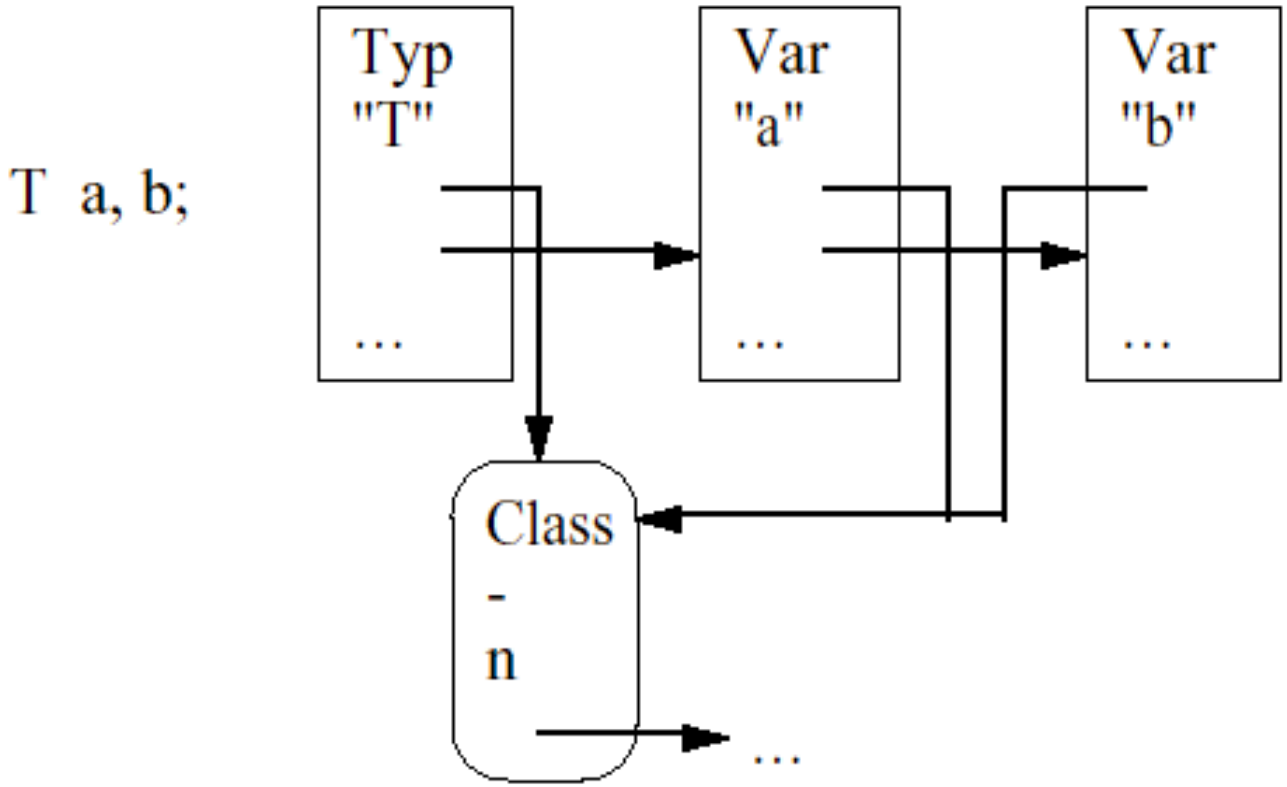
- Jezik se naziva **strogo tipiziranim** (engl. strongly typed) ako definicija jezika podržana statičkim i dinamičkim proverama tipova garantuje da u izvršavanju programa neće doći do neprimećene greške upotrebe pogrešnog tipa (u izrazima, pozivima funkcija itd): Java, C++, C#.
- Suprotnost je **slabo tipizirani** (weakly typed) jezik, npr. Perl, PHP, Javascript.

Ekvivalentost izraza tipova

- Kada se za dva izraza tipa može reći da predstavljaju međusobno zamenljive tipove?
- Postoje dve vrste ekvivalencije tipova:
  - **po imenu**: dva tipa su ekvivalentna ako su reprezentovana istim čvorom tipa u grafu (tj. imaju isto ime).
  - **strukturna**: dva tipa su ekvivalentna ako imaju istu strukturu (topologiju grafa)
- U Mikrojavi ekvivalentnost tipova je po imenu, sa izuzetkom nizova: dva niza su ekvivalentna ako su im tipovi elemenata ekvivalentni.

Ekvivalentost tipova po imenu

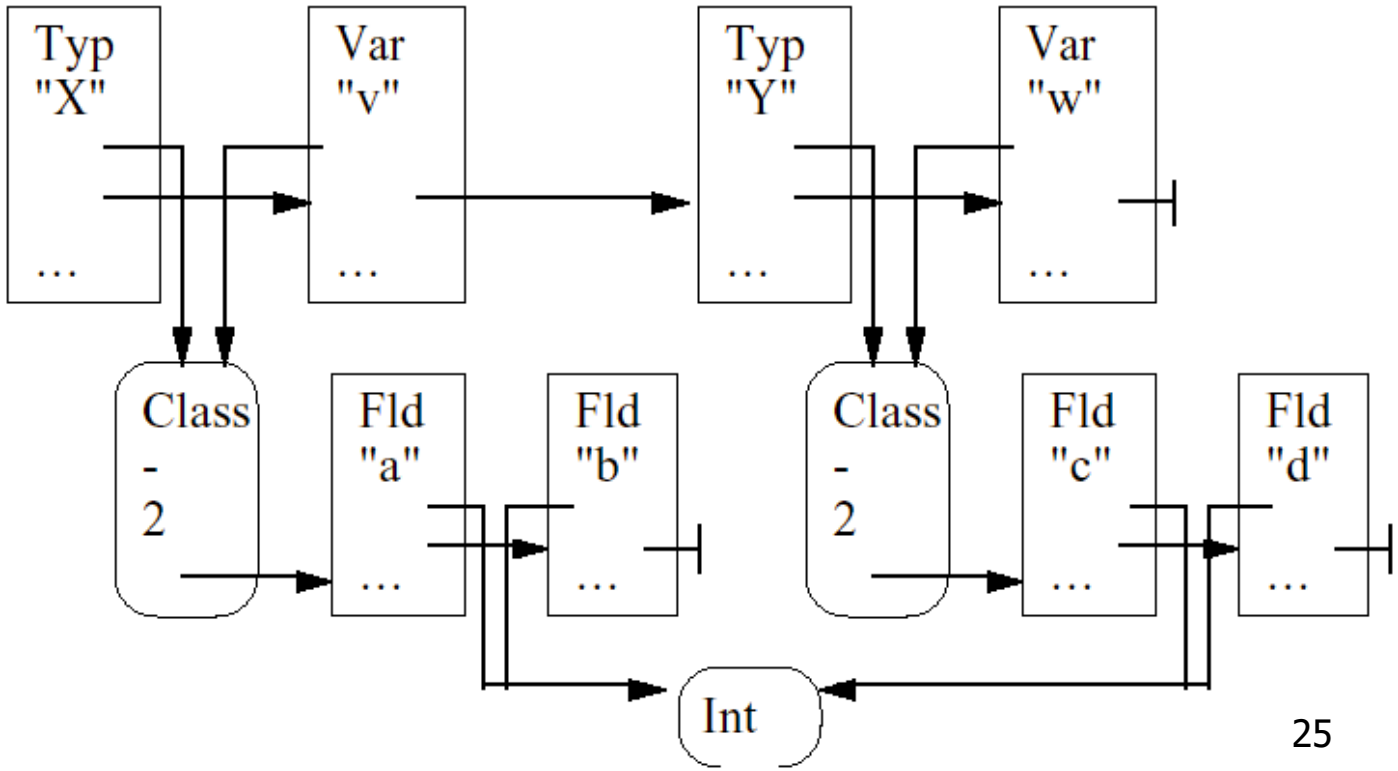
- Dva tipa su ekvivalentna ako su reprezentovana istim čvorom tipa u grafu (tj. imaju isto ime).
- Primer: a i b imaju isti tip.



Ekvivalentost tipova po strukturi

- Dva tipa su ekvivalentna ako imaju istu strukturu (topologiju grafa tipova).
- Primer: tipovi promenljivih **v** i **w** su ekvivalentni po strukturi (ne važi u mikrojavi). Ovaj bio bi ispravan u C-u (uz zamenu *class* sa *struct*) gde se svuda primenjuje ekvivalencija po imenu, osim kod zapisa gde se gleda struktura.

```
class X {int a, b;}
class Y {int c, d;}
X v;
Y w;
```



Provera tipova u mikrojavi

```
class Struct {
    ...
    // checks, if two types are compatible (e.g. in comparisons)
    public bool CompatibleWith (Struct other) {
        return this.Equals(other) ||
                this == Tab.nullType && other.IsRefType() ||
                other == Tab.nullType && this.isRefType();
    }
}
```

Provera tipova u mikrojavi

```
class Struct {
    ...
    // checks, if this can be assigned to dest (explicit. assignment, parameters)
    public bool AssignableTo (Struct dest) {
        if   (     this.Equals(dest) ||
                   this == Tab.nullType && dest.IsRefType() ||
                   kind == Kinds.Arr && dest.kind == Kinds.Arr &&   // for len()
                              dest.elemType == Tab.noType)
            return true;
        for (Struct s = elemType; s != null; s = s.elemType) {
            if (s.equals(dest))
                return true;
        }
        return false;
    }
}
```

Provera tipova u mikrojavi

```
class Struct {
    ...
    // checks, if two types are equal (structural equivalence for array,
    // name equivalence otherwise)
    public bool Equals (Struct other) {
        if (kind == Kinds.Arr)
            return other.kind == Kinds.Arr &&
                   elemType.Equals(other.elemType);
        return other == this;
    }

    public bool IsRefType() {
        return kind == Kinds.Class || kind == Kinds.Arr;
    }
}
```

Ostale stvari u vezi tipova (van Mikrojave)

- Konverzije tipova: na primer, u izrazima gde se koriste operandi različitih tipova, kao što je sabiranje celobrojne vrednosti sa realnom, mora se operand konvertovati u drugi tip (ceo broj u realni).
  - Ako konverziju eksplicitno navodi programer, radi se o eksplicitnoj konverziji (npr. operator cast u Cu (int)x).
  - Ako konverziju ne navodi programer, nego je automatski uradi kompajler, reč je o implicitnoj konverziji (engl. coercion)
- Preklapanje funkcija i operatora (engl. overloading): kada isto ime ima različita značenja u zavisnosti od konteksta. Na primer, u C++u:

```
int fja( int x ) { x = 1; }
int fja ( T& t ) { t.end = true; }
```

su dve funkcije istog imena.