Semantika

- **Značenje** (semantika) programskog jezika se opisuje na neformalan način

Semantička pravila za mC

- Standardni identifikatori su:
  - rezervisane reči: **`int`**, **`if`**, **`else`**, **`return`**
  - identifikator **`main`** je ime funkcije, za koju se podrazumeva da je definisana u izvršivom mC programu. Izvršavanje mC programa započne izvršavanjem **`main`** funkcije
  - definicija **`main`** funkcije izgleda:

```
int main ()
{
   ...
}
```

Telo **`main`** funkcije definiše korisnik. Ako telo **`main`** (i svake druge) funkcije ne sadrži **`return`** iskaz, podrazumeva se da je povratna vrednost funkcije nedefinisana i da do povratka iz funkcije dolazi po izvršavanju poslednjeg iskaza iz njenog tela.

Semantička pravila za mC

- Područje važenja (doseg, domašaj, vidljivost) identifikatora (*lexical-scope*, *static-scope*)
  - identifikatori se razvrstavaju u globalne i lokalne
  - globalni identifikatori su imena globalnih promenljivih i imena funkcija. Oni su definisani na nivou programa (van funkcija)
  - lokalni identifikatori su imena lokalnih promenljivih. Oni su definisani u okviru funkcija (tu spadaju i parametri funkcija)
  - područje važenja globalnih identifikatora je od mesta njihove definicije do kraja programskog teksta
  - područje važenja lokalnih identifikatora je od mesta njihove definicije do kraja tela funkcije u kojoj su definisani (znači svaka funkcija poseduje svoje lokalne promenljive)
  - identifikatori mogu biti korišćeni samo iza njihove definicije (to proizlazi iz područja njihovog važenja)

Semantička pravila za mC

- Jednoznačnost identifikatora
  - svi globalni identifikatori moraju biti međusobno različiti
  - svi lokalni identifikatori iste funkcije moraju biti međusobno različiti
  - ako postoje identični globalni identifikatori i lokalni identifikatori neke funkcije, tada van te funkcije važe globalni, a unutar nje lokalni identifikatori
  - lokalni identifikatori raznih funkcija mogu biti identični
  - rezervisane reči smeju da se koriste samo u skladu sa svojom ulogom i na globalnom i na lokalnom nivou
  - standardni identifikator **`main`** je rezervisan samo na globalnom nivou

Semantička pravila za mC

- Ispravnost korišćenja identifikatora
  - identifikatori smeju biti korišćeni samo u skladu sa njihovom definicijom:
    - imenu funkcije ne sme biti pridružena vrednost
- Provera tipova
  - na upotrebu identifikatora utiče i njihov tip
    - na primer, tip identifikatora s leve strane iskaza pridruživanja određuje tip izraza sa desne strane ovog iskaza tj. leva i desna strana iskaza pridruživanja moraju imati isti tip
  - isto važi i za konstante

Semantička pravila za mC

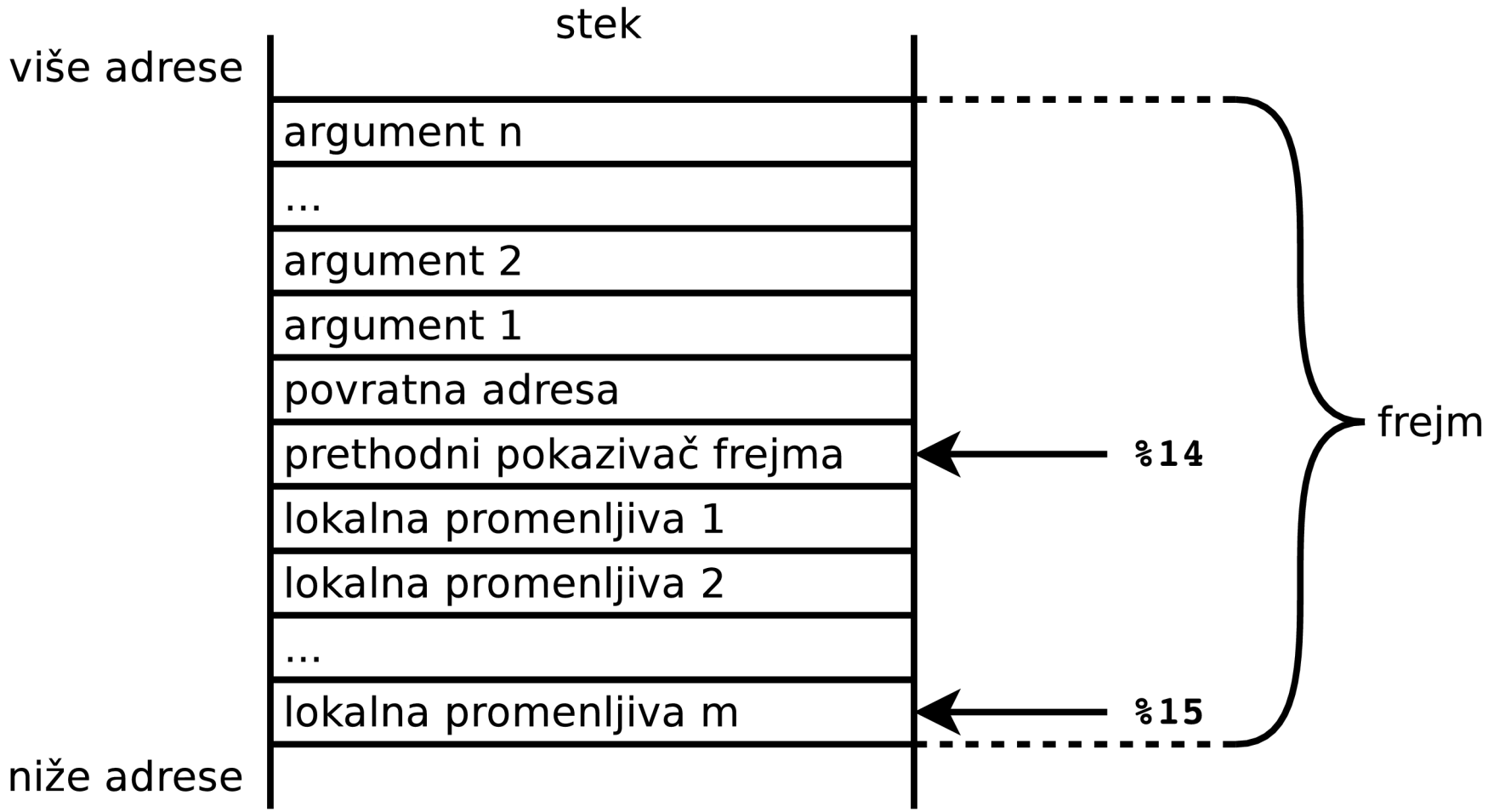
- tip izraza iz **`return`** iskaza neke funkcije i tip ove funkcije moraju biti identični
- tipovi korespondentnih parametara funkcije i argumenata iz njenog poziva moraju biti identični
- argumenti poziva funkcije moraju da se slažu po broju sa parametrima funkcije
- u istom relacionom izrazu smeju biti samo identifikatori istog tipa. Isto važi i za konstante
- podrazumeva se da je tip konstante **`int`** ako nije eksplicitno naznačeno da je tip konstante **`unsigned`**

Organizacija memorije za mC

- Globalni identifikatori su statični – postoje za sve vreme izvršavanja programa
  - za njih se mogu rezervisati memorijske lokacije u toku kompajliranja
- Lokalni identifikatori su dinamični – postoje samo za vreme izvršavanja funkcija
  - za njih se zauzimaju memorijske lokacije na početku izvršavanja funkcija
  - ove lokacije se oslobađaju na kraju izvšavanja funkcija
  - zato se lokalnim identifikatorima dodeljuju memorijske lokacije sa steka
- Deo steka koji se zauzima za izvršavanje neke funkcije se zove (stek) **frejm**.

Organizacija memorije za mC

- Izgled tipičnog frejma:



Organizacija memorije za mC

- Argumenti su vrednosti parametara koji se adresiraju u odnosu na **%14** (pokazivač frejma). Tako

  parametru$_1$ odgovara operand **8(%14)**

  parametru$_2$ odgovara operand **12(%14)**

  ...

  parametru$_n$ odgovara operand **4+n*4(%14)**

- I vrednosti lokalnih promenljivih se adresiraju u odnosu na **%14**. Tako

  lokalnoj promenljivoj$_1$ odgovara operand **-4(%14)**

  lokalnoj promenljivoj$_2$ odgovara operand **-8(%14)**

  ...

  lokalnoj promenljivoj$_m$ odgovara operand **-m*4(%14)**

- Prethodno je važno jer se u vreme kompilacije ne znaju apsolutne adrese parametara i lokalnih promenljivih

Tabela simbola

- Pronađeni globalni i lokalni identifikatori (simboli) se čuvaju u tabeli simbola

ime (char*)	vrsta (unsigned)	tip (unsigned)	atributi (int)
main	1 (FUNCTION)	1 (int)	2 (2 parametra)
x	3 (PARAMETER)	1 (int)	1 (redni broj)
y	3 (PARAMETER)	1 (int)	2 (redni broj)
...	...	...	...

Tabela simbola

- Operacije za rukovanje tabelom simbola
  - ubacivanje novog elementa u tabelu simbola
  - pretraga tabele simbola
  - ispis sadržaja tabele simbola (za dibagiranje)
  - brisanje (dela) tabele simbola

Tabela simbola

- Skraćenje vremena pretraživanja *(lookup)* tabele simbola
  - za veliki broj simbola u tabeli simbola srednje vreme pretraživanja postaje predugačko
  - srednje vreme pretraživanja tabele simbola se može smanjiti pomoću heš *(hash)* tabele:

Tabela simbola

- Operacije za rukovanje heš tabelom
  - izračunavanje heš indeksa na osnovu simbola
  - ubacivanje novog simbola u tabelu
  - pretraživanje tabele na osnovu simbola
  - brisanje postojećeg elementa iz tabele

Tabela simbola

- I globalni i lokalni identifikatori mogu biti smešteni u istu tabelu simbola
  - globalni su prisutni u tabeli simbola sve vreme kompilacije
  - lokalni su prisutni u tabeli simbola samo u toku kompajliranja njihove funkcije
- Pošto u tabeli simbola istovremeno mogu postojati identični globalni i lokalni identifikatori, radi njihovog razlikovanja u tabeli simbola mora biti naznačena vrsta identifikatora (simbola)
- Za svaki identifikator mora postojati i oznaka njegovog tipa
- Za parametre i lokalne promenljive mora postojati njihova relativna pozicija na steku u odnosu na **%14**

Tabela simbola

- Za svaku funkciju (radi provere ispravnosti njenih poziva) mora postojati
  - broj njenih parametara
  - tipovi pojedinih parametara
- Radi uniformnosti, u tabeli simbola mogu biti čuvane i konstante
  - one su lokalne za funkciju i
  - moraju biti jedinstvene
- Takođe, radi uniformnosti, u tabeli simbola mogu biti smeštene i oznake radnih registara
  - one se smeštaju u tabelu simbola u vreme njene inicijalizacije
    - **%0** u element sa indeksom **0**, .., a
    - **%12** u element sa indeksom **12**
  - prisutne su u tabeli simbola sve vreme kompilacije

Tabela simbola

- Opis mogućih značenja elemenata tabele simbola

STRING SIMBOLA	VRSTA SIMBOLA	TIP SIMBOLA	ATRIBUT SIMBOLA
	FUNKCIJA	DA	parametri
	LOKALNA PROMENLJIVA	DA	redni broj promenljive
	GLOBALNA PROMENLJIVA	DA	-
	PARAMETAR	DA	redni broj parametra
	KONSTANTA	DA	-
	RADNI REGISTAR	DA	-

Tabela simbola

- Uzroci neuspešne kompilacije
  - broj elemenata u tabeli simbola ograničava najveći mogući broj identifikatora u programu
  - broj elemenata niza sa tipovima parametara ograničava najveći mogući broj parametara u funkciji
  - broj radnih registara takođe uvodi ograničenje u radu kompajlera
- primer – **`symtab.h`**

Semantička analiza

- U okviru semantičke analize proverava se poštovanje semantičkih pravila
- Radi provere semantičkih pravila u tabelu simbola se smeštaju odgovarajući atributi identifikatora, čim sintaksna analiza nedvosmisleno ukaže na vrednost atributa
  - provera poštovanja semantičkih pravila se zasniva na korišćenju ovih atributa
- Pojedini koraci provere se obavljaju nakon prepoznavanja pojedinih sintaksnih pravila u toku sintaksne analize. Ovi koraci se nazivaju **semantičke akcije** i vezuju se za pomenuta sintaksna pravila
- Uporedni prikaz sintaksnih pravila i za njih vezanih semantičkih akcija obrazuje **sintaksno usmeravanu definiciju** (*syntax-directed definition*)

Semantička analiza

- provera globalnih i lokalnih identifikatora (pojam *variable*)
  - područje važenja
    - proveriti da li je promenljiva već definisana
      - ako nije: upisati u tabelu simbola sa atributima
      - ako jeste: prijaviti grešku - duplikat
  - svi globalni identifikatori moraju biti međusobno različiti
  - svi lokalni identifikatori jedne funkcije moraju biti međusobno različiti
- identični globalni i lokalni identifikatori ?
- identični lokalni identifikatori u različitim funkcijama ?

Semantička analiza

```
int x;
unsigned y;
int main() {
    int a;
    int x;
}
```

name	*kind*	*type*	*attribute*	*param_types*
x	GLOBAL_VAR	INT_TYPE	NO_ATTRIBUTE	NULL
y	GLOBAL_VAR	UNSIGNED_TYPE	NO_ATTRIBUTE	NULL
main	FUNCTION	INT_TYPE	0	NULL
a	LOCAL_VAR	INT_TYPE	1	NULL
x	LOCAL_VAR	INT_TYPE	2	NULL

Semantička analiza

- provera upotrebe identifikatora
  - leva strana iskaza dodele: ime lokalne promenljive, globalne promenljive ili parametra
  - desna strana iskaza dodele: ime se mora deklarisati pre upotrebe (pojam *exp*)

```
int main() {
    int a;
    a = a + b;   //error zbog b
}
```

name	*kind*	*type*	*attribute*	*param_types*
main	FUNCTION	INT_TYPE	0	NULL
a	LOCAL_VAR	INT_TYPE	1	NULL

Semantička analiza

- provera tipova
  - leva i desna strana iskaza dodele moraju imati isti tip
    - (pojam *assignment_statement*)
  - operandi u numeričkim izrazima moraju imati isti tip
    - (pojam *mul_exp*, *num_exp*)
  - operandi u relacionom izrazu moraju imati isti tip
    - (pojam *rel_exp*)
  - tip izraza iz return iskaza neke funkcije mora biti istog tipa kao i tip njene povratne vrednosti

```
check_types(idx1, idx2);
```

idx1	a	LOCAL_VAR	INT_TYPE	1	NULL
idx2	b	LOCAL_VAR	INT_TYPE	2	NULL

Semantička analiza

- provera identifikatora **`main`** (pojam *program*)
  - da li u tabeli simbola postoji **`main`** funkcija
  - da li je tip povratne vrednosti **`int`**

```
int main() {
    return 0;
}
```

name	*kind*	*type*	*attribute*	*param_types*
main	FUNCTION	INT_TYPE	0	NULL
0	CONSTANT	INT_TYPE	NO_ATTRIBUTE	NULL

Semantička analiza

- provera poziva funkcije (pojam *function_call* i *argument*)
  - proveriti da li se poziva postojeća funkcija
  - izbrojati argumente (?= broj parametara funkcije)
  - proveriti tip svakog argumenta (?= tipu odgovarajućeg parametra funkcije)

Semantička analiza

```
int f(int x, int y) {
    int a;
    a = x + y;
    return a;
}
```

Izgled tabele simbola neposredno pre kraja obrade funkcije **`f`**:

name	*kind*	*type*	*attribute*	*param_types*
f	FUNCTION	INT_TYPE	2	INT_TYPE, INT_TYPE, –, –, –
x	PARAMETER	INT_TYPE	1	NULL
y	PARAMETER	INT_TYPE	2	NULL
a	LOCAL_VAR	INT_TYPE	1	NULL

Semantička analiza

```
int f(int x, int y) {
    return x;
}
int main() {
    int a;
    a = f(1, a);  //error:  a = f(1u, a, 7);  a = g();
}
```

Izgled tabele simbola nakon obrade obe funkcije:

name	*kind*	*type*	*attribute*	*param_types*				
f	FUNCTION	INT_TYPE	2	INT_TYPE	INT_TYPE	–	–	–
main	FUNCTION	INT_TYPE	0	NULL				

Semantička analiza - primer

- primer semantičke analize – za razumevanje je važno pratiti redosled redukcija!
- primer - **`semantic`**