

Projekat iz predmeta programski prevodioci
Makro funkcije i *conditional inclusion* u miniC jeziku

Mentor:

Dunja Vrbaški
Milena Laketić

Student:

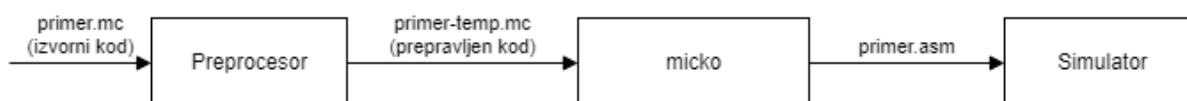
Uroš Prijović

Sadržaj

Uvod	3
Sintaksna analiza	4
Definisanje makroa	4
Conditional inclusion proverom da li je makro definisan	5
Conditional inclusion proverom da li makro nije definisan	6
Semantička analiza	7
Definisanje makroa	7
Generisanje međukoda	8
Testiranje	9
Zaključak	10

Uvod

U ovom radu će biti obuhvaćeno idejno rešenje problema makro funkcija, makroa i uslovnog uključivanja(eng. *conditional inclusion*) u miniC jeziku. Navedeni elementi C jezika su deo preprocesora koji predstavlja korak pre kompajlera, čiji kod koristi kompajler da izvrši prevođenje. S toga, za pravilnu implementaciju prevodioca koji će podržavati iskaze kao što su `#define`, `#ifdef`, `#ifndef` itd. potrebno je napraviti parser koji će generisati međukod koji će se proslediti micko kompajleru.



Slika 1 - tok prevođenja miniC programa sa dodatkom preprocesora

Definisani i objašnjeni iskazi u ovom radu će biti: `#define`, `#ifdef`, `#ifndef`, `#if`, `#else`, `#endif`. Ideja je da se vrednosti makroa i makro funkcija obrade pre ulasku u kompajler, kako bi se eventualno eliminisao kod koji ne ispunjava *conditional inclusion* i kako bi se upravljalo tokom programa i njegovor prevođenja sa višeg i bržeg nivoa.

Sintaksna analiza

Definisanje makroa

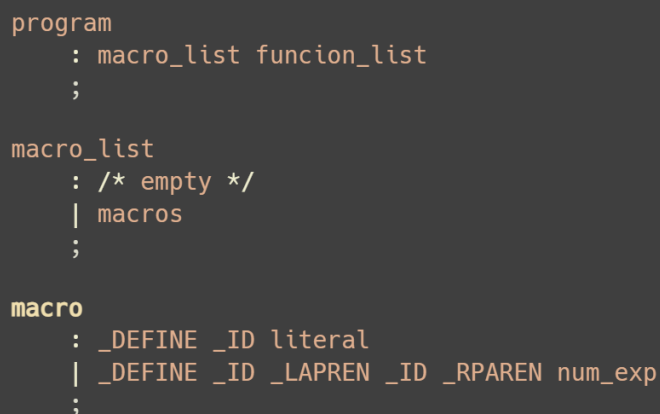
Za definisanje makroa i kreiranje sintaksnog analizatora koji će da podrži ovaj iskaz potrebno je definisati novu vrstu simbola MACRO, tako što će se ubaciti `MACRO = 0x16` na kraj enumeracije `kinds` (u datoteci `defs.h`), kao što je to urađeno za globalne promenljive na sedmom terminu vežbi. Za dopunu sintakse potrebno je da se podrže sledeće alternative definisanja makroa:

1. `"#define" <id1> <num_exp>`
2. `"#define" <id1>(<id2>)<num_exp>`

gde `<id1>` identifikator makroa, `<num_exp>` može biti literal, identifikator, numerički izraz ili poziv (makro) funkcije, dok `<id2>` predstavlja parametar funkcije.



```
"#define"      { return _DEFINE; }
```



```
program
: macro_list funcion_list
;

macro_list
: /* empty */
| macros
;

macro
: _DEFINE _ID literal
| _DEFINE _ID _LAPREN _ID _RPAREN num_exp
;
```

Conditional inclusion proverom da li je makro definisan

Uslovno uključivanje ili *conditional inclusion* može se primeniti na bilo koje parče koda, dok će u ovom primeru biti obuhvaćene samo funkcije zbog kompleksnosti celokupnog problema. Uzimajući ovo u obzir sintaksa će biti sledeća:

`"#ifdef" <id> <function> ["#else" <function>] "#endif"`

gde je `<id>` identifikator makroa, `<function>` funkcija i `"#else" <function>` opcioni blok.

```
"#else"      { return _DEF_ELSE; }
"#define"    { return _DEFINE; }
"#ifdef"     { return _IFDEF; }
"#endif"     { return _ENDIF; }
```

```
function_list
: if_function
| function_list if_function
;

if_function
: function
| def_if_part _ENDIF
| def_if_part _DEF_ELSE function _ENDIF
;

function
: type _ID _LPAREN parameter _RPAREN body
;

def_if_part
: _IFDEF _ID function
;
```

Conditional inclusion proverom da li makro nije definisan

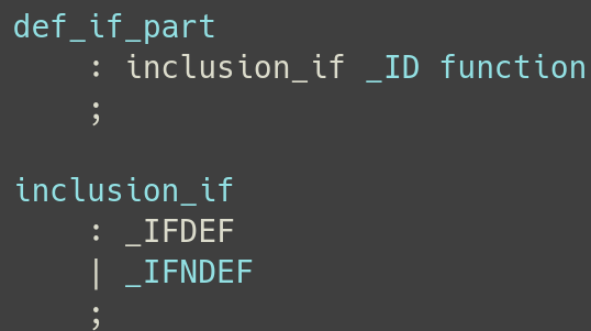
Uzimaju se isti uslovi kao iz prethodnog potpoglavlja s tim da se ovde proverava da li makro nije definisan pa se na osnovu toga ulazi u blok koji sledi iskaz i sintaksa je sledeća:

```
"#ifndef" <id> <function> ["#else" <function>] "#endif"
```

gde je <id> identifikator makroa, <function> funkcija i "#else" <function> opcioni blok.



```
"#ifndef"      { return _IFNDEF; }
```



```
def_if_part
: inclusion_if _ID function
;

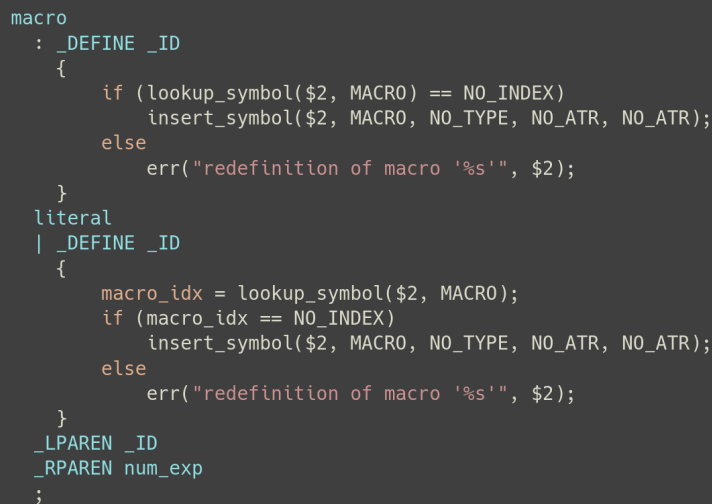
inclusion_if
: _IFDEF
| _IFNDEF
;
```

Semantička analiza

Definisanje makroa

Jedna od odlika makro funkcija u odnosu na klasične funkcije jeste nedostatak tipova prilikom definisanja i prevođenja. Za makro funkciju se ne definišu povratni tip i tip parametara kao što se ne definiše tip makro promenljive ali se zato svi argumenti prenose direktno u formi u kojoj su prosleđeni.

Jedina semantička provera koju je potrebno obaviti za definisanje makro promenljive ili makro funkcije jeste da li ona već postoji, tj. da li je u pitanju ponovna deklaracija.



```
macro
: _DEFINE _ID
{
    if (lookup_symbol($2, MACRO) == NO_INDEX)
        insert_symbol($2, MACRO, NO_TYPE, NO_ATR, NO_ATR);
    else
        err("redefinition of macro '%s'", $2);
}
literal
| _DEFINE _ID
{
    macro_idx = lookup_symbol($2, MACRO);
    if (macro_idx == NO_INDEX)
        insert_symbol($2, MACRO, NO_TYPE, NO_ATR, NO_ATR);
    else
        err("redefinition of macro '%s'", $2);
}
_LPAREN _ID
_RPAREN num_exp
;
```

Za uslovno uključivanje proverama definisanosti makroa nije potrebno praviti dodatne semantičke provere.

Generisanje međukoda

Međukod predstavlja miniC kod koji uključuje prethodno definisane makroe iz izvornog miniC koda i delove koda za koje su uslovi uključanja zadovoljeni. Tok kojim je potrebno razvijati naš prevodilac je sledeći ([Slika 1](#)): potrebno je propustiti izvorni kod kroz parser koji će vršiti sintaksnu i semantičku proveru, zatim generisati međukod koji će se dalje proslediti micko kompajleru i prevesti ga na hipotetski asemblerski kod. Za navedeni tok, potrebno je implementirati sintaksni analizator opisan u poglavlju [Sintaksna analiza](#), zatim semantički analizator po uputstvu iz poglavlja [Semantička analiza](#) i na kraju izmeniti Makefile koja će regulisati tok prevođenja.

Testiranje

Sintaksna i semantička analiza su testirane pomoću pet testova sa pravilnim primerima, dva testa sa testovima sa sintaksnim greškama i jednog testa sa semantičkom greškom.

Ok testovi

Prvi test sa pravilnim primerima proverava da li je moguća deklaracija konstante definišući jednostavnu konstantu `#define VAR 10`. Drugi test proverava mogućnost definisanja više uzastopnih konstanti, konkretno u primeru 3. Treći test proverava mogućnost definisanja makro funkcija `#define A(X) 1 + X` i `#define B(Y) Y + 3`. Četvrti i peti test proveravaju mogućnost upotrebe `#ifdef`, `#else` i `#endif` tokena.

Sintaksni testovi

Prvi test sa sintaksnom greškom proverava da li će naš "preprocesor" javiti grešku prilikom pokušaja deklarisanja konstante bez vrednosti, dok drugi test proverava ispravnost *conditional inclusion*-a izostavljanjem `#endif` tokena.

Semantički testovi

Skup testova sadrži samo jedan test za grešku u semantici, a taj test proverava da li je moguće deklarirati istoimeni makro više puta.

Zaključak

Upotreba konstanti i makro funkcija nije moguća ovom implementacijom zbog sledećih ne završenih uslova u implementaciji: izmena *Makefile* skripte, generisanje međukoda, semantička provera međukoda.

Izmena skripte nije urađena zbog kompleksnosti, jer bi trebalo da se kreiraju privremeni fajlovi za svaki fajl na ulazi, pa da se ti privremeni fajlovi prosleđuju micko kompajleru i na kraju je potrebno i obrisati te privremene fajlove.

Generisanje međukoda, takođe, nije urađeno zbog kompleksnosti samog zadatka. Za uspešno generisanje koda potrebno je svesti broj makro funkcija na minimum standardnom zamenom delova koda, potrebno je vršiti provere uslova uključivanja i na osnovu njih selektovati koji deo koda će se naći u okviru međukoda.

Postoje sintaksne i semantičke provere za naš "preprocesor", međutim, trebalo bi dopuniti sintaksni analizator `#define` iskazom kao što je to učinjeno za preprocesor i trebalo bi dodati pravilo u semantičkom analizatoru samog kompajler koje bi proveravala da li su sve pozvane makro funkcije definisane. Ovaj problem nije rešen zbog moje nemogućnosti da pronađem dokumentaciju hipotetskog asemblera i vidim način definisanja makro funkcija u kodu.