Chapter 1

Leksička analiza

Leksički analizator (skener) je program namenjen za leksičku analizu (skeniranje) teksta. Zadatak skenera je da pročita ulazni tekst i da ga podeli na simbole. Skener radi tako što čita tekst, karakter po karakter, i pokušava da prepozna neku od zadatih reči. Ukoliko naiđe na simbol čija pojava nije predviđena, treba da prijavi leksičku grešku.

Na primer, ako se u ulaznoj miniC datoteci nalazi kod:

```
if(a == 6)
  a = -b;
```

skener bi trebao da prepozna sledeće simbole:

`if` ključna reč

`(` leva mala zagrada

`a` identifikator

`==` relacioni operator

`6` celobrojni literal

`)` desna mala zagrada

`a` identifikator

`=` operator dodele

`-` operator unarni minus

`b` identifikator

`;` separator

Beline (u koje spadaju prazno mesto, tabulator i karakter za novi red) se preskaču, odnosno prepoznaju i ignorišu, jer imaju ulogu separatora simbola.

1.1 Implementacija skenera

Skener se može napraviti (programirati) ručno ili korišćenjem alata za generisanje skenera. Tokom ovog kursa se za generisanje skenera koristi program `flex`. `Flex` generiše skener

na osnovu pravila zadatih u `flex` specifikaciji. Specifikaciju kreira korisnik u posebnoj datoteci koja po konvenciji ima ekstenziju `.l`. Ova datoteka sadrži pravila koja opisuju reči koje skener treba da prepozna. Pravila se zadaju u obliku regularnih izraza. Svakom regularnom izrazu moguće je pridružiti akciju (u obliku `C` koda) koja će se izvršiti kada skener prepozna dati regularni izraz. Uobičajena akcija je vraćanje oznake simbola, odnosno tokena za taj simbol. Token je numerička oznaka klase (vrste) simbola. Na primer: token `NUMBER` označava bilo koji broj (i broj `2` i broj `359`), dok token `IF` označava jedino ključnu reč `if`.

`.l` datoteka se prosleđuje `flex`-u koji generiše skener na jeziku C, u funkciji `yylex()` u datoteci `lex.yy.c`. Akcije, pridružene regularnim izrazima u `.l` datoteci, su delovi `C` koda koji se direktno prenose (kopiraju) u `lex.yy.c`. Ovako generisan C kod se dalje prevodi C kompajlerom (tj. može da se uključi u C program). Ovakav program predstavlja leksički analizator koji transformiše ulazni tekst u niz tokena (u skladu sa pravilima zadatim u specifikaciji).

Korišćenje programa `flex` je prikazano na slici 1.1:

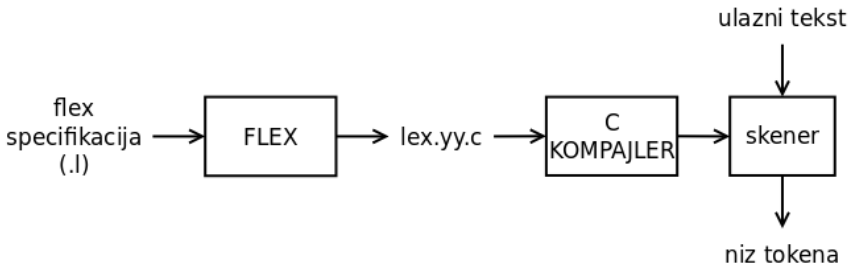

Figure 1.1: Korišćenje `flex`-a

Prilikom izvršavanja, skener će tražiti pojavu stringa u ulaznom tekstu koji odgovara nekom regularnom izrazu. Ako ga pronađe, izvršiće akciju (kod) koju je korisnik pridružio tom regularnom izrazu. U suprotnom, podrazumevana akcija za tekst koji ne odgovara ni jednom regularnom izrazu je kopiranje teksta na standardni izlaz.

Skener se generiše, kompajlira i pokreće naredbama:

```
$ flex scanner.l
$ gcc -o scanner lex.yy.c
$ ./scanner
```

U prvoj liniji se pokreće `flex` koji preuzima specifikaciju iz datoteke `scanner.l` i generiše skener u datoteci `lex.yy.c`.

U drugoj liniji se poziva `gcc` kompajler sa prosleđenom datotekom `lex.yy.c` (u kojoj se u funkciji `yylex()` nalazi skener). Svič `--o` prosleđen `gcc`-u omogućava korisniku da iza sviča navede ime programa koji će biti izgenerisan. Ukoliko se ne upotrebi ovaj svič, kompajler će napraviti izvršnu datoteku sa imenom `a.out`.

U trećoj liniji naredbom `./scanner` se izgenerisani skener pokreće u interaktivnom režimu, što znači da ulaz očekuje preko tastature, a da se program završava kada se na ulazu pojavi kombinacija `^D` (`CTRL+D`).

Ukoliko skener treba da skenira neku datoteku (na primer datoteku sa imenom `test.txt`), prilikom pokretanja programa treba upotrebiti redirekciju standardnog ulaza:

```
$ ./scanner <test.txt
```

1.2 miniC skener

Skener za miniC jezik je implementiran u datoteci `scanner.l` i dat na listinzima 1.1, 1.2 i 1.3.

Prvi deo specifikacije započinje uključivanjem dve opcije koje utiču na to kako će *flex* napraviti skener (listing 1.1):

```
%option noyywrap yylineno
```

`noyywrap` opcija znači da će skener skenirati samo jednu datoteku, i

`yylineno` opcija znači da će skener koristiti globalnu promenljivu *flex-a* `yylineno` koja sadrži broj trenutno skenirane linije.

Za neke simbole se, osim tokena, prosleđuje još neka vrednost koja bliže opisuje simbol (npr. za simbol *broj* se uz token `NUMBER` šalje i vrednost konkretnog broja, recimo `123`). U ovakvim situacijama, za smeštanje vrednosti, koja može biti različitog tipa - za različite simbole, se koristi unija. Unija se definiše u prvom delu specifikacije (listing 1.1), a koristi u drugom delu specifikacije (listing 1.2). Za miniC skener, potrebna je unija koja sadrži jedan celobrojni tip (`int i`) i jedan pokazivač na karakter (`char *s`). Promenljiva tipa unije, preko koje se, uz token, prosleđuje vrednost, se u *flex-u* zove `yylval`.

Prvi deo specifikacije sadrži enumeraciju `token`s sa definicijama tokena i niz `token_strings` koji sadrži imena tokena u obliku stringova koji se koriste prilikom ispisa tokena. U prvom delu je definisana i enumeracija `values` sa konstantama koje opisuju razne operatore i, opet, niz imena tih konstanti `value_strings` u obliku stringova (listing 1.1). Svi stringovi će se koristiti u `main` funkciji, prilikom prikazivanja prepoznatih tokena i vrednosti.

Listing 1.1: `scanner.l` - tokeni

```
%option noyywrap yylineno
%{
  union {
    int i;
    char *s;
  } yylval;

  enum tokens { _TYPE = 1, _IF, _ELSE, _RETURN,
                _LPAREN, _RPAREN, _LBRACKET, _RBRACKET,
                _SEMICOLON, _ASSIGN, _AROP, _RELOP, _ID,
                _INT_NUMBER, _UINT_NUMBER };
  char *token_strings[] = { "NONE", "_TYPE", "_IF",
                "_ELSE", "_RETURN", "_LPAREN","_RPAREN",
                "_LBRACKET", "_RBRACKET", "_SEMICOLON",
                "_ASSIGN", "_AROP", "_RELOP", "_ID",
                "_INT_NUMBER", "_UINT_NUMBER" };

  enum values { ADD, SUB, LT, EQ, INT, UINT};
  char *value_strings[] = { "ADD", "SUB",
                            "LT",  "EQ",
                            "INT", "UINT" };
```

Prvo pravilo preskače beline (listing 1.2). Sledeća tri pravila prepoznaju ključne reči `if`, `else` i `return`. Za ključne reči dovoljno je vezati samo tokene (nisu potrebne dodatne vrednosti koje bi ih detaljnije definisale).

Sledeća dva pravila prepoznaju označene i neoznačene celobrojne tipove. Za oba simbola se vezuje isti token (`_TYPE`), ali različite vrednosti uz token. Vrednosti su konstante (`INT` i `UINT`) koje opisuju prepoznate tipove (listing 1.2). Vrednost se prosleđuje kroz promenljivu `yylval` koja je definisana u prvom delu specifikacije kao unija (listing 1.1). Kako je konstanta celobrojnog tipa, njena vrednost se smešta u polje `i` promenljive `yylval`.

Dalje su navedena pravila koja prepoznaju delimitere (`(`, `)`, `{`, `}`), separatore (`;`) i operator dodele (`=`), koji su potputno opisani samo tokenima (listing 1.2).

Zatim slede pravila za prepoznavanje aritmetičkih i relacionih operatora (listing 1.2). Za ove operatore se, osim tokena, prosleđuje još i odgovarajuća konstanta kao vrednost simbola. Konstanta je potrebna da bi se operatori razlikovali međusobno, jer imaju isti token (na primer: konstanta `ADD` za operator `+`).

Zatim slede pravila za prepoznavanje identifikatora i literala. U ovim slučajevima se, pored tokena, prosleđuje još i string simbola, koji se smešta u polje `s` (tipa `char*`) promenljive `yylval` (listing 1.2). String prepoznatog simbola flex čuva u promenljivoj `yytext`. Regularni izraz za identifikator opisuje string koji počinje slovom (malim ili velikim: `[a-zA-Z]`), a iza njega se može naći slovo (malo ili veliko) ili cifra, 0 ili više puta: `[a-zA-Z0-9]*`. Minimalni identifikator se, znači, sastoji od jednog slova. Regularni izraz za označeni celobrojni literal opisuje string od minimalno jedne do maksimalno deset cifara: `[0-9]{1,10}`, ispred kojih se može, a ne mora, naći predznak plus ili minus: `[+-]?`. Regularni izraz za neoznačeni celobrojni literal ne prepoznaje predznak, već dozvoljava da se iza broja navede malo ili veliko slovo "u": `[uU]`.

U skeneru nam je bitan redosled pravila, tako da pravila za prepoznavanje ključnih reči treba staviti ispred pravila za identifikator (listing 1.2).

Linijski komentari se prepoznaju pomoću regularnog izraza koji prepoznaje dva znaka *slash* na početku stringa: `\/\/` (može i: `"//"`), a zatim bilo koji znak osim `\n` znaka, 0 ili više puta: `.*`.

Poslednje pravilo koje sadrži operator `.` (bilo koji znak osim `\n` znaka) služi da prepozna leksičku grešku. Bilo koji znak koji ne pripada nijednom prethodnom pravilu predstavlja grešku, koju treba saopštiti korisniku.

Listing 1.2: `scanner.l` - pravila

```
%%

[ \t\n]+        { /* skip */ }

"if"            { return _IF; }
"else"          { return _ELSE; }
"return"        { return _RETURN; }

"int"           { yylval.i = INT;  return _TYPE; }
"unsigned"      { yylval.i = UINT; return _TYPE; }

"("             { return _LPAREN; }
")"             { return _RPAREN; }
"{"             { return _LBRACKET; }
"}"             { return _RBRACKET; }
";"             { return _SEMICOLON; }
"="             { return _ASSIGN; }

"+"             { yylval.i = ADD; return _AROP; }
```

```
"-"             { yylval.i = SUB; return _AROP; }

"<"             { yylval.i = LT; return _RELOP; }
"=="            { yylval.i = EQ; return _RELOP; }

[a-zA-Z][a-zA-Z0-9]*  { yylval.s = yytext;
                        return _ID; }
[+-]?[0-9]{1,10}      { yylval.s = yytext;
                        return _INT_NUMBER;}
[0-9]{1,10}[uU]       { yylval.s = yytext;
                        return _UINT_NUMBER;}

\/\/.*          { /* skip */ }
.               { printf("line␣%d:␣LEXICAL␣ERROR␣"
                  "on␣char␣%c\n",yylineno, *yytext);}
```

Funkcija `main()` preuzima od skenera (funkcije `yylex()`) tokene i vrednosti simbola i prikazuje ih korisniku (listing 1.3). Za ispis se koriste stringovi iz niza `token_strings` i `value_strings` koji su definisani u prvom delu *flex* specifikacije (listing 1.1).

Listing 1.3: `scanner.l - main() funkcija`

```
%%

int main() {
  int tok;
  while(tok = yylex()) {
    printf("%s", token_strings[tok]);
    switch(tok) {
      case _ID:
      case _INT_NUMBER:
      case _UINT_NUMBER:
        printf(":␣%s", yylval.s); break;
      case _AROP:
      case _RELOP:
      case _TYPE:
        printf(":␣%s", value_strings[yylval.i]);break;
    }
    printf("\n");
  }
}
```

1.2.1 Primer upotrebe skenera

Ako bismo ovaj program pokrenuli sa redirekcijom standardnog ulaza na datoteku `test.c` sa listinga 1.4 i redirekcijom standardnog izlaza na datoteku `out.c`, njen sadržaj bi bio kao na listingu 1.5:

```
$ ./scanner <test.c >out.c
```

Listing 1.4: `test.c`

```
int boolval(int x) {
  if(x == 0)
    return 0;
  else
    return 1;
}
```

Listing 1.5: `out.c`

```
_TYPE: INT_TYPE
_ID: boolval
_LPAREN
_TYPE: INT_TYPE
_ID: x
_RPAREN
_LBRACKET
_IF
_LPAREN
_ID: x
_RELOP: EQ
_INT_NUMBER: 0
_RPAREN
_RETURN
_INT_NUMBER: 0
_SEMICOLON
_ELSE
_RETURN
_INT_NUMBER: 1
_SEMICOLON
_RBRACKET
```

Skener je ulaznu datoteku podelio na simbole, pri čemu je zanemario beline i komentare.

Ako bismo napisali uslov `if` iskaza sa nepodržanim relacionim operatorom `!=` umesto `==` (listing 1.6), skener bi nam prijavio leksičku grešku (listing 1.7):

Listing 1.6: `test2.c`

```
if(x != 0)
  return 0;
```

Listing 1.7: `out2.c`

```
_IF
_LPAREN
_ID: x
line 1: LEXICAL ERROR on char !
_ASSIGN
_INT_NUMBER: 0
_RPAREN
_RETURN
_INT_NUMBER: 0
_SEMICOLON
```

1.3 Vežbe

1. Proširiti miniC jezik i miniC skener `while` iskazom.
2. Proširiti miniC jezik i miniC skener `break` iskazom.
3. Proširiti miniC jezik i miniC skener `for` iskazom.
4. Proširiti miniC jezik i miniC skener `switch` iskazom.
5. Proširiti miniC jezik i miniC skener nizovima.
6. Proširiti relacione izraze miniC jezika i proširiti miniC skener.

7. Proširiti miniC jezik logičkim izrazima `&&` i `||` i proširiti miniC skener.

8. Proširiti miniC jezik i miniC skener definicijama funkcija sa više parametara, i pozivima funkcija sa više argumenata.