Języki formalne i techniki translacji
Laboratorium - Projekt (wersja β)

Termin oddania: ostatnie zajęcia przed 17 stycznia 2018
Wysłanie do wykładowcy: przed 23:59 24 stycznia 2018

Używając BISON-a i FLEX-a napisz kompilator prostego języka imperatywnego do kodu maszyny rejestrowej. Specyfikacja języka i maszyny jest zamieszczona poniżej. Kompilator powinien sygnalizować miejsce i rodzaj błędu (np. druga deklaracja zmiennej, użycie niezadeklarowanej zmiennej, niewłaściwe użycie nazwy tablicy,...), a w przypadku braku błędów zwracać kod na maszynę rejestrową. Kod wynikowy powinien wykonywać się jak najszybciej (w miarę optymalnie, mnożenie i dzielenie powinny być wykonywane w czasie logarytmicznym w stosunku do wartości argumentów).

Program powinien być oddany z plikiem `Makefile` kompilującym go oraz z plikiem `README` opisującym dostarczone pliki i sposób użycia kompilatora. (Przy przesyłaniu do wykładowcy program powinien być spakowany programem zip a archiwum nazwane numerem indeksu studenta.)

Prosty język imperatywny Język powinien być zgodny z gramatyką zamieszczoną na rysunku 1 i spełniać następujące warunki:

1. `+ - * / %` oznaczają odpowiednio dodawanie, odejmowanie, mnożenie, dzielenie całkowitoliczbowe i obliczanie reszty na liczbach naturalnych;
2. `= <> < > <= >=` oznaczają odpowiednio relacje $=$, $\neq$, $<$, $>$, $\leqslant$ i $\geqslant$ na liczbach naturalnych;
3. `:=` oznacza przypisanie;
4. deklaracja `tab[100]` oznacza zadeklarowanie tablicy `tab` o 100 elementach indeksowanych od 0 do 99, identyfikator `tab[i]` oznacza odwołanie do `i`-tego elementu tablicy `tab`;
5. pętla `FOR` ma iterator lokalny, przyjmujący wartości od wartości stojącej po `FROM` do wartości stojącej po `TO` kolejno w odstępie +1 lub w odstępie -1 jeśli użyto słowa `DOWNTO`;
6. liczba iteracji pętli `FOR` jest ustalana na początku i nie podlega zmianie w trakcie wykonywania pętli (nawet jeśli zmieniają się wartości zmiennych wyznaczających początek i koniec pętli);
7. iterator pętli `FOR` nie może być modyfikowany wewnątrz pętli (kompilator w takim przypadku powinien zgłaszać błąd);
8. instrukcja `READ`, czyta wartość z zewnątrz i podstawia pod zmienną, a `WRITE`, wypisuje wartość zmiennej/liczby na zewnątrz,
9. pozostałe instrukcje są zgodne z ich znaczeniem w większości języków programowania;
10. `pidentifier` jest opisany wyrażeniem regularnym `[_a-z]+`;
11. `num` jest liczbą naturalną w zapisie dziesiętnym (nie ma ograniczeń na wielkość liczby);
12. działania arytmetyczne są wykonywane na liczbach naturalnych, w szczególności $a-b = \max\{a-b, 0\}$, dzielenie przez zero daje wynik 0 i resztę także 0;
13. małe i duże litery są rozróżniane;
14. w programie można użyć komentarzy postaci: `( komentarz )`, które nie mogą być zagnieżdżone.

```
program       -> VAR vdeclarations BEGIN commands END

vdeclarations -> vdeclarations pidentifier
               | vdeclarations pidentifier[num]
               |

commands    -> commands command
             | command

command     -> identifier := expression;
             | IF condition THEN commands ELSE commands ENDIF
             | IF condition THEN commands ENDIF
             | WHILE condition DO commands ENDWHILE
             | FOR pidentifier FROM value TO value DO commands ENDFOR
             | FOR pidentifier FROM value DOWNTO value DO commands ENDFOR
             | READ identifier;
             | WRITE value;

expression  -> value
             | value + value
             | value - value
             | value * value
             | value / value
             | value % value

condition   -> value = value
             | value <> value
             | value < value
             | value > value
             | value <= value
             | value >= value

value       -> num
             | identifier

identifier  -> pidentifier
             | pidentifier[pidentifier]
             | pidentifier[num]
```

Rysunek 1: Gramatyka języka

Maszyna rejestrowa Maszyna rejestrowa składa się z rejestru a, licznika rozkazów k oraz ciągu komórek pamięci p_i, dla $i = 0, 1, 2, \ldots$. Maszyna pracuje na liczbach naturalnych (wynikiem odejmowania większej liczby od mniejszej jest 0). Program maszyny składa się z ciągu rozkazów, który niejawnie numerujemy od zera. W kolejnych krokach wykonujemy zawsze rozkaz o numerze k aż napotkamy instrukcję `HALT`. Początkowa zawartość rejestrów i komórek pamięci jest nieokreślona, a licznik rozkazów k ma wartość 0. Lista rozkazów wraz z ich interpretacją i czasem wykonania zamieszczone są tabeli poniżej.

Rozkaz	Interpretacja	Czas
`GET`	pobraną liczbę zapisuje w rejestrze a oraz $k \leftarrow k+1$	100
`PUT`	wyświetla zawartość rejestru a oraz $k \leftarrow k+1$	100
`LOAD i`	$a \leftarrow p_i$ oraz $k \leftarrow k+1$	10
`LOADI i`	$a \leftarrow p_{p_i}$ oraz $k \leftarrow k+1$	10
`STORE i`	$p_i \leftarrow a$ oraz $k \leftarrow k+1$	10
`STOREI i`	$p_{p_i} \leftarrow a$ oraz $k \leftarrow k+1$	10
`ADD i`	$a \leftarrow a + p_i$ oraz $k \leftarrow k+1$	10
`ADDI i`	$a \leftarrow a + p_{p_i}$ oraz $k \leftarrow k+1$	10
`SUB i`	$a \leftarrow \max\{a - p_i, 0\}$ oraz $k \leftarrow k+1$	10
`SUBI i`	$a \leftarrow \max\{a - p_{p_i}, 0\}$ oraz $k \leftarrow k+1$	10
`SHR`	$a \leftarrow \lfloor a/2 \rfloor$ oraz $k \leftarrow k+1$	1
`SHL`	$a \leftarrow 2 * a$ oraz $k \leftarrow k+1$	1
`INC`	$a \leftarrow a+1$ oraz $k \leftarrow k+1$	1
`DEC`	$a \leftarrow \max(a-1, 0)$ oraz $k \leftarrow k+1$	1
`ZERO`	$a \leftarrow 0$ oraz $k \leftarrow k+1$	1
`JUMP j`	$k \leftarrow j$	1
`JZERO j`	jeśli $a = 0$ to $k \leftarrow j$, w p.p.. $k \leftarrow k+1$	1
`JODD j`	jeśli a nieparzyste to $k \leftarrow j$, w p.p. $k \leftarrow k+1$	1
`HALT`	zatrzymaj program	0

Przejście do nieistniejącego rozkazu lub wywołanie nieistniejącego rejestru jest traktowane jako błąd.
Kod maszyny rejestrowej napisany w `C++` *znajduje się w pliku* `interpreter.cc` *(wersja dla dużych liczb z użyciem biblioteki* `cln` *w pliku* `interpreter-cln.cc`*).*

Przykładowe kody programów i odpowiadające im kody maszyny rejestrowej

```
1  VAR
2      a b
3  BEGIN
4      READ a;
5      WHILE a > 0 DO
6          b := a / 2;
7          b := 2 * b;
8          IF a > b THEN WRITE 1;
9          ELSE WRITE 0;
10         ENDIF
11         a := a / 2;
12     ENDWHILE
13 END
```

```
0  GET
1  STORE 0
2  JZERO 13
3  SHR
4  SHL
5  STORE 1
6  LOAD 0
7  SUB 1
8  PUT
9  LOAD 0
10 SHR
11 STORE 0
12 JUMP 2
13 HALT
```

```
1  ( sito Eratostenesa )
2  VAR
3      n j sito[100]
4  BEGIN
5      n := 100-1;
6      FOR i FROM n DOWNTO 2 DO
7          sito[i] := 1;
8      ENDFOR
9      FOR i FROM 2 TO n DO
10         IF sito[i] <> 0 THEN
11             j := i + i;
12             WHILE j <= n DO
13                 sito[j] := 0;
14                 j := j + i;
15             ENDWHILE
16             WRITE i;
17         ENDIF
18     ENDFOR
19 END
```

```
0  ZERO
1  INC
2  SHL
3  SHL
4  SHL
5  STORE 5
6  ZERO
7  INC
8  SHL
9  INC
10 SHL
11 SHL
12 SHL
13 SHL
14 INC
15 SHL
16 INC
17 STORE 0
18 STORE 2
19 DEC
20 STORE 3
21 JZERO 35
22 LOAD 5
23 ADD 2
24 STORE 4
25 ZERO
26 INC
27 STOREI 4
28 LOAD 2
29 DEC
30 STORE 2
31 LOAD 3
32 DEC
33 STORE 3
34 JUMP 21
35 ZERO
36 INC
37 SHL
38 STORE 2
39 LOAD 0
40 DEC
41 STORE 3
42 JZERO 73
43 LOAD 5
44 ADD 2
45 STORE 4
46 LOADI 4
47 JZERO 66
48 LOAD 2
49 ADD 2
50 STORE 1
51 LOAD 0
52 INC
53 SUB 1
54 JZERO 64
55 LOAD 5
56 ADD 1
57 STORE 4
58 ZERO
59 STOREI 4
60 LOAD 1
61 ADD 2
62 STORE 1
63 JUMP 51
64 LOAD 2
65 PUT
66 LOAD 2
67 INC
68 STORE 2
69 LOAD 3
70 DEC
71 STORE 3
72 JUMP 42
73 HALT
```

Optymalność wykonywania mnożenia i dzielenia

```
1  ( Rozklad liczby na czynniki pierwsze )
2  VAR
3      n m reszta potega dzielnik
4  BEGIN
5      READ n;
6      dzielnik := 2;
7      m := dzielnik * dzielnik;
8      WHILE n >= m DO
9          potega := 0;
10         reszta := n % dzielnik;
11         WHILE reszta = 0 DO
12             n := n / dzielnik;
13             potega := potega + 1;
14             reszta := n % dzielnik;
15         ENDWHILE
16         IF potega > 0 THEN ( czy znaleziono dzielnik )
17             WRITE dzielnik;
18             WRITE potega;
19         ELSE
20             dzielnik := dzielnik + 1;
21             m := dzielnik * dzielnik;
22         ENDIF
23     ENDWHILE
24     IF n <> 1 THEN ( ostatni dzielnik )
25         WRITE n;
26         WRITE 1;
27     ENDIF
28 END
```

Dla powyższego programu kod wynikowy na załączonej maszynie powinien działać w czasie porównywalnym (mniej więcej tego samego rzędu wielkości - ilość cyfr) do poniższych wyników:

```
Uruchamianie programu.
? 1234567890
> 2
> 1
> 3
> 2
> 5
> 1
> 3607
> 1
> 3803
> 1
Skończono program (czas: *******).

Uruchamianie programu.
? 12345678901
> 857
> 1
> 14405693
> 1
Skończono program (czas: *******).

Uruchamianie programu.
? 12345678903
> 3
> 1
> 4115226301
> 1
Skończono program (czas: *********).
```