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

Termin oddania: ostatnie zajęcia przed 22 stycznia 2017
Wysłanie do wykładowcy: przed 23:59 29 stycznia 2017

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. iterator pętli `FOR` nie może być modyfikowany wewnątrz pętli (kompilator w takim przypadku powinien zgłaszać błąd);
7. instrukcja `READ`, czyta wartość z zewnątrz i podstawia pod zmienną, a `WRITE`, wypisuje wartość zmiennej/liczby na zewnątrz,
8. instrukcja `SKIP` nic nie robi;
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
             | 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;
             | SKIP;

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 5 rejestrów $r_0, \ldots, r_4$, 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 i`	pobraną liczbę zapisuje w rejestrze r_i oraz $k \leftarrow k+1$	100
`PUT i`	wyświetla zawartość rejestru r_i oraz $k \leftarrow k+1$	100
`LOAD i`	$r_i \leftarrow p_{r_0}$ oraz $k \leftarrow k+1$	10
`STORE i`	$p_{r_0} \leftarrow r_i$ oraz $k \leftarrow k+1$	10
`ADD i`	$r_i \leftarrow r_i + p_{r_0}$ oraz $k \leftarrow k+1$	10
`SUB i`	$r_i \leftarrow \max\{r_i - p_{r_0}, 0\}$ oraz $k \leftarrow k+1$	10
`COPY i`	$r_0 \leftarrow r_i$ oraz $k \leftarrow k+1$	1
`SHR i`	$r_i \leftarrow \lfloor r_i/2 \rfloor$ oraz $k \leftarrow k+1$	1
`SHL i`	$r_i \leftarrow 2 * r_i$ oraz $k \leftarrow k+1$	1
`INC i`	$r_i \leftarrow r_i + 1$ oraz $k \leftarrow k+1$	1
`DEC i`	$r_i \leftarrow \max(r_i - 1, 0)$ oraz $k \leftarrow k+1$	1
`ZERO i`	$r_i \leftarrow 0$ oraz $k \leftarrow k+1$	1
`JUMP j`	$k \leftarrow j$	1
`JZERO i j`	jeśli $r_i = 0$ to $k \leftarrow j$, w p.p.. $k \leftarrow k+1$	1
`JODD i j`	jeśli r_i 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  ZERO 0
1  GET 1
2  STORE 1
3  JZERO 1 18
4  SHR 1
5  SHL 1
6  INC 0
7  STORE 1
8  DEC 0
9  LOAD 1
10 INC 0
11 SUB 1
12 PUT 1
13 DEC 0
14 LOAD 1
15 SHR 1
16 STORE 1
17 JUMP 3
18 HALT
```

```
{ sito Eratostenesa }
VAR
    n j sito[100]
BEGIN
    n := 100-1;
    FOR i FROM n DOWNTO 2 DO
        sito[i] := 1;
    ENDFOR
    FOR i FROM 2 TO n DO
        IF sito[i] <> 0 THEN
            j := i + i;
            WHILE j <= n DO
                sito[j] := 0;
                j := j + i;
            ENDWHILE
            WRITE i;
        ELSE
          SKIP;
        ENDIF
    ENDFOR
END
```

```
ZERO 1
INC 1
SHL 1
INC 1
SHL 1
SHL 1
SHL 1
SHL 1
INC 1
SHL 1
INC 1
ZERO 0
STORE 1
ZERO 1
INC 1
LOAD 4
DEC 4
COPY 4
INC 0
INC 0
INC 0
JZERO 4 26
STORE 1
DEC 0
DEC 4
JUMP 21
ZERO 1
ZERO 2
INC 2
INC 2
ZERO 0
INC 0
STORE 2
ZERO 0
LOAD 4
DEC 4
JZERO 4 84
ZERO 0
INC 0
LOAD 0
INC 0
INC 0
INC 0
LOAD 3
JZERO 3 77
ZERO 0
INC 0
LOAD 3
ADD 3
INC 0
STORE 3
ZERO 0
LOAD 3
INC 3
INC 0
INC 0
SUB 3
JZERO 3 73
LOAD 0
INC 0
INC 0
INC 0
ZERO 3
STORE 3
ZERO 0
INC 0
INC 0
LOAD 3
DEC 0
ADD 3
INC 0
STORE 3
JUMP 51
ZERO 0
INC 0
LOAD 3
PUT 3
ZERO 0
INC 0
LOAD 3
INC 3
STORE 3
DEC 4
JUMP 36
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     ELSE
28         SKIP;
29     ENDIF
30 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: *********).
```