Języki formalne i techniki translacji

Laboratorium - Projekt

Termin oddania: ostatnie zajęcia przed 17 stycznia 2016
Wysłanie do wykładowcy: przed 23:59 28 stycznia 2016

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 (jeśli nie ma słowa `DOWN`) lub w odstępie -1 jeśli użyto słowa `DOWN`;
6. instrukcja `GET`, czyta wartość z zewnątrz i podstawia pod zmienną, a `PUT`, wypisuje wartość zmiennej/liczby na zewnątrz,
7. pozostałe instrukcje są zgodne z ich znaczeniem w większości języków programowania;
8. `pidentifier` jest opisany wyrażeniem regularnym `[_a-z]+`;
9. `num` jest liczbą naturalną w zapisie dziesiętnym (nie ma ograniczeń na wielkość liczby);
10. 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;
11. rozróżniamy małe i duże litery;

```
program       -> DECLARE vdeclarations IN commands END

vdeclarations -> vdeclarations pidentifier
               | vdeclarations pidentifier(num)
               |

commands    -> commands command
             |

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

12. w programie można użyć komentarzy postaci: `[ komentarz ]`, które nie mogą być zagnieżdżone.

Maszyna rejestrowa Maszyna rejestrowa składa się z 10 rejestrów $r_0, \ldots, r_9$, 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. Poniżej jest lista rozkazów wraz z ich interpretacją i czasem wykonania:

Rozkaz	Interpretacja	Czas
`READ i`	pobraną liczbę zapisuje w rejestrze r_i oraz $k \leftarrow k+1$	100
`WRITE i`	wyświetla zawartość rejestru r_i oraz $k \leftarrow k+1$	100
`LOAD i j`	$r_i \leftarrow p_{r_j}$ oraz $k \leftarrow k+1$	20
`STORE i j`	$p_{r_j} \leftarrow r_i$ oraz $k \leftarrow k+1$	20
`COPY i j`	$r_i \leftarrow r_j$ oraz $k \leftarrow k+1$	1
`ADD i j`	$r_i \leftarrow r_i + r_j$ oraz $k \leftarrow k+1$	5
`SUB i j`	$r_i \leftarrow \max\{r_i - r_j, 0\}$ oraz $k \leftarrow k+1$	5
`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
`RESET 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 $k \leftarrow j$, wpp. $k \leftarrow k+1$	1
`JODD i j`	jeśli r_i nieparzyste to $k \leftarrow j$, wpp. $k \leftarrow k+1$	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`.

Przykładowe kody programów i odpowiadające im kody maszyny rejestrowej

Przykład 1

```
1  DECLARE
2      a b
3  IN
4      GET a;
5      WHILE a > 0 DO
6          b := a / 2;
7          b := 2 * b;
8          IF a > b THEN PUT 1;
9          ELSE PUT 0;
10         ENDIF
11         a := a / 2;
12     ENDWHILE
13 END
```

```
0  READ 0
1  JZERO 0 10
2  COPY 1 0
3  SHR 1
4  SHL 1
5  COPY 2 0
6  SUB 2 1
7  WRITE 2
8  SHR 0
9  JUMP 1
10 HALT
```

Przykład 2

```
1  [ sito Eratostenesa ]
2  DECLARE
3      n j sito(100)
4  IN
5      n := 100-1;
6      FOR i DOWN FROM n TO 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             PUT i;
17         ENDIF
18     ENDFOR
19 END
```

```
0  RESET 0
1  INC 0
2  SHL 0
3  INC 0
4  SHL 0
5  SHL 0
6  SHL 0
7  SHL 0
8  INC 0
9  SHL 0
10 INC 0
11 RESET 1
12 INC 1
13 COPY 2 0
14 COPY 3 0
15 DEC 3
16 JZERO 3 21
17 STORE 1 2
18 DEC 2
19 DEC 3
20 JUMP 16
21 RESET 1
22 RESET 2
23 INC 2
24 INC 2
25 COPY 3 0
26 INC 3
27 SUB 3 2
28 JZERO 3 45
29 LOAD 4 2
30 JZERO 4 42
31 COPY 5 2
32 ADD 5 2
33 COPY 6 0
34 INC 6
35 SUB 6 5
36 JZERO 6 41
37 STORE 1 5
38 ADD 5 2
39 SUB 6 2
40 JUMP 36
41 WRITE 2
42 INC 2
43 DEC 3
44 JUMP 28
45 HALT
```

Optymalność wykonywania mnożenia i dzielenia Dla następującego programu

```
1  [ Rozklad liczby na czynniki pierwsze ]
2  DECLARE
3      n m reszta potega dzielnik
4  IN
5      GET 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             PUT dzielnik;
18             PUT potega;
19         ELSE
20             dzielnik := dzielnik + 1;
21             m := dzielnik * dzielnik;
22         ENDIF
23     ENDWHILE
24     IF n != 1 THEN [ ostatni dzielnik ]
25         PUT n;
26         PUT 1;
27     ENDIF
28 END
```

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: *********).
```