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

Termin oddania: ostatnie zajęcia przed 19 stycznia 2019
Wysłanie do wykładowcy: przed 23:45 27 stycznia 2019

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 oraz zawierającym dane autora. W przypadku użycia innych języków niż C/C++ należy także zamieścić dokładne instrukcje co należy doinstalować dla systemu Ubuntu. Wywołanie programu powinno wyglądać następująco[1]

```
kompilator <nazwa pliku wejściowego> <nazwa pliku wyjściowego>
```

czyli dane i wynik są podawane przez nazwy plików (nie przez strumienie). 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 w tablicy 1 i spełniać następujące warunki:

1. 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;
2. `+ - * / %` oznaczają odpowiednio dodawanie, odejmowanie, mnożenie, dzielenie całkowitoliczbowe i obliczanie reszty na liczbach naturalnych;
3. `= != < > <= >=` oznaczają odpowiednio relacje $=$, $\neq$, $<$, $>$, $\leqslant$ i $\geqslant$ na liczbach naturalnych;
4. `:=` oznacza przypisanie;
5. deklaracja `tab(10:100)` oznacza zadeklarowanie tablicy `tab` o 91 elementach indeksowanych od 10 do 100, identyfikator `tab(i)` oznacza odwołanie do `i`-tego elementu tablicy `tab`, deklaracja zawierająca pierwszą liczbę większą od drugiej powinna być zgłaszana jako błąd;
6. 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`;
7. 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);
8. iterator pętli `FOR` nie może być modyfikowany wewnątrz pętli (kompilator w takim przypadku powinien zgłaszać błąd);
9. instrukcja `READ` czyta wartość z zewnątrz i podstawia pod zmienną, a `WRITE` wypisuje wartość zmiennej/liczby na zewnątrz;
10. pozostałe instrukcje są zgodne z ich znaczeniem w większości języków programowania;

[1] Dla innych niektórych języków programowania należy napisać w pliku `README` że jest inny sposób wywołania kompilatora, np. `java kompilator` lub `python kompilator`

```
program      -> DECLARE declarations IN commands END

declarations -> declarations pidentifier;
              | declarations pidentifier(num: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
              | DO commands WHILE condition ENDDO
              | 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)
```

Tablica 1: Gramatyka języka

11. `pidentifier` jest opisany wyrażeniem regularnym `[_a-z]+`;
12. `num` jest liczbą naturalną w zapisie dziesiętnym (w kodzie wejściowym liczby są ograniczone do typu `long long` (64 bitowy), na maszynie rejestrowej nie ma ograniczeń na wielkość liczb, obliczenia mogą generować dowolną liczbę naturalną);
13. małe i duże litery są rozróżniane;
14. w programie można użyć komentarzy postaci: `[ komentarz ]`, które nie mogą być zagnieżdżone.

Maszyna rejestrowa Maszyna rejestrowa składa się z 8 rejestrów (r_A, r_B, r_C, r_D, r_E, r_F, r_G, r_H), 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

Rozkaz	Interpretacja	Czas
`GET` X	pobraną liczbę zapisuje w rejestrze r_X oraz $k \leftarrow k+1$	100
`PUT` X	wyświetla zawartość rejestru r_X oraz $k \leftarrow k+1$	100
`LOAD` X	$r_X \leftarrow p_{r_A}$ oraz $k \leftarrow k+1$	50
`STORE` X	$p_{r_A} \leftarrow r_X$ oraz $k \leftarrow k+1$	50
`COPY` X Y	$r_X \leftarrow r_Y$ oraz $k \leftarrow k+1$	5
`ADD` X Y	$r_X \leftarrow r_X + r_Y$ oraz $k \leftarrow k+1$	5
`SUB` X Y	$r_X \leftarrow \max\{r_X - r_Y, 0\}$ oraz $k \leftarrow k+1$	5
`HALF` X	$r_X \leftarrow \lfloor r_X/2 \rfloor$ oraz $k \leftarrow k+1$	1
`INC` X	$r_X \leftarrow r_X + 1$ oraz $k \leftarrow k+1$	1
`DEC` X	$r_X \leftarrow \max(r_X - 1, 0)$ oraz $k \leftarrow k+1$	1
`JUMP` j	$k \leftarrow j$	1
`JZERO` X j	jeśli $r_X = 0$ to $k \leftarrow j$, w p.p. $k \leftarrow k+1$	1
`JODD` X j	jeśli r_X nieparzyste to $k \leftarrow j$, w p.p. $k \leftarrow k+1$	1
`HALT`	zatrzymaj program	0

Tablica 2: Rozkazy maszyny rejestrowej ($X, Y \in \{A, B, C, D, E, F, G, H\}$)

`HALT`. Początkowa zawartość rejestrów i komórek pamięci jest nieokreślona, a licznik rozkazów k ma wartość 0. W tablicy 2 jest podana lista rozkazów wraz z ich interpretacją i kosztem wykonania. W programie można zamieszczać komentarze zaczynające się od znaku `#` i obowiązujące do końca linii. Białe znaki w kodzie są pomijane. Przejście do nieistniejącego rozkazu lub wywołanie nieistniejącego rejestru jest traktowane jako błąd.

Wszystkie przykłady oraz kod maszyny rejestrowej napisany w `C+` zostały zamieszczone w pliku `labor4.zip` (kod maszyny jest w dwóch wersjach: podstawowej na liczbach typu `long long` oraz w wersji `cln` na dowolnych liczbach naturalnych, która jest jednak wolniejsza w działaniu ze względu na użycie biblioteki dużych liczb).

Przykładowe kody programów i odpowiadające im przykładowe kody maszyny rejestrowej

Przykład 1 – binarny zapis liczby

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

```
-1  # zapis binarny
 0  GET A
 1  JZERO A 10
 2  COPY B A
 3  HALF B
 4  ADD B B
 5  COPY C A
 6  SUB C B
 7  PUT C
 8  HALF A
 9  JUMP 1
10  HALT
```

Przykład 2 – sito Eratostenesa

```
1  [ sito Eratostenesa ]
2  DECLARE
3      n; j; sito(2:100);
4  IN
5      n := 100;
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  SUB B B          # generowanie n=100
1  INC B
2  ADD B B
3  INC B
4  ADD B B
5  ADD B B
6  ADD B B
7  INC B
8  ADD B B
9  ADD B B
10 SUB C C          # generowanie 1
11 INC C
12 COPY D B         # i:=n
13 COPY E B         # licznik pętli
14 DEC E            # licznik--
15 JZERO E 20       # wyjście z pętli
16 COPY A D         # sito(i):=1
17 STORE C
18 DEC D            # i--
19 JUMP 14          # powrót do pętli
20 DEC C            # generowanie 0
21 SUB D D          # i:=2
22 INC D
23 INC D
24 COPY E B         # licznik pętli
25 DEC E            # licznik--
26 JZERO E 42       # wyście z pętli
27 COPY A D         # czytanie a(i)
28 LOAD F
29 JZERO F 40       # sito(i)=0
30 COPY G D         # j:=i
31 ADD G D          # j+=i
32 COPY H B         # j<=n ?
33 INC H
34 SUB H G
35 JZERO H 39       # wyjście z while
36 COPY A G         # sito(j):=0
37 STORE C
38 JUMP 31          # powrót do pętli
39 PUT D            # write i
40 INC D            # i++
41 JUMP 25          # powrót do pętli
42 HALT             # koniec
```

Optymalność wykonywania mnożenia i dzielenia

```
1  [ Rozklad liczby na czynniki pierwsze ]
2  DECLARE
3      n; m; reszta; potega; dzielnik;
4  IN
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 koszt działania kodu wynikowego na załączonej maszynie powinien być porównywalny do poniższych wyników (mniej więcej tego samego rzędu wielkości - liczba cyfr):

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

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

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