# Języki formalne i techniki translacji
## Laboratorium - Projekt (wersja $\alpha$)

**Termin oddania: ostatnie zajęcia przed 25 stycznia 2020**
**Wysłanie do wykładowcy: przed 23:45 31 stycznia 2020**

Używając BISON-a i FLEX-a napisz kompilator prostego języka imperatywnego do kodu maszyny wirtualnej. 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 wirtualną. 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. Archiwum nie powinno zawierać żadnych zbędnych plików.

**Prosty język imperatywny** Język powinien być zgodny z gramatyką zamieszczoną w tablicy 1 i spełniać następujące warunki:

1. działania arytmetyczne są wykonywane na liczbach całkowitych, ponadto dzielenie przez zero powinno dać wynik 0 i resztę także 0;
2. `PLUS MINUS TIMES DIV MOD` oznaczają odpowiednio dodawanie, odejmowanie, mnożenie, dzielenie całkowitoliczbowe i obliczanie reszty na liczbach całkowitych, reszta z dzielenia powinna mieć taki sam znak jak dzielnik;
3. `EQ NEQ LE GE LEQ GEQ` oznaczają odpowiednio relacje $=$, $\neq$, $<$, $>$, $\leqslant$ i $\geqslant$ na liczbach całkowitych;
4. `ASSIGN` oznacza przypisanie;
5. deklaracja `tab(-10:100)` oznacza zadeklarowanie tablicy `tab` o 111 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;

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

```
1  program       -> DECLARE declarations BEGIN commands END
2                 | BEGIN commands END
3
4  declarations  -> declarations, pidentifier
5                 | declarations, pidentifier(num:num)
6                 | pidentifier
7                 | pidentifier(num:num)
8
9  commands      -> commands command
10                | command
11
12 command       -> identifier ASSIGN expression;
13                | IF condition THEN commands ELSE commands ENDIF
14                | IF condition THEN commands ENDIF
15                | WHILE condition DO commands ENDWHILE
16                | DO commands WHILE condition ENDDO
17                | FOR pidentifier FROM value TO value DO commands ENDFOR
18                | FOR pidentifier FROM value DOWNTO value DO commands ENDFOR
19                | READ identifier;
20                | WRITE value;
21
22 expression    -> value
23                | value PLUS value
24                | value MINUS value
25                | value TIMES value
26                | value DIV value
27                | value MOD value
28
29 condition     -> value EQ value
30                | value NEQ value
31                | value LE value
32                | value GE value
33                | value LEQ value
34                | value GEQ value
35
36 value         -> num
37                | identifier
38
39 identifier    -> pidentifier
40                | pidentifier(pidentifier)
41                | pidentifier(num)
```

Tablica 1: Gramatyka języka

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

**Maszyna wirtualna** Maszyna wirtualna składa się z licznika rozkazów $k$ oraz ciągu komórek pamięci $p_i$, dla $i = 0, 1, 2, ...$ (z przyczyn technicznych $i \leqslant 2^{62}$). Maszyna pracuje na liczbach całkowitych. 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 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 adresu pamięci jest traktowane jako błąd.

| Rozkaz | Interpretacja | Czas |
|---|---|---|
| `GET` | pobraną liczbę zapisuje w komórce pamięci $p_0$ oraz $k \leftarrow k+1$ | 100 |
| `PUT` | wyświetla zawartość komórki pamięci $p_0$ oraz $k \leftarrow k+1$ | 100 |
| `LOAD` $i$ | $p_0 \leftarrow p_i$ oraz $k \leftarrow k+1$ | 10 |
| `STORE` $i$ | $p_i \leftarrow p_0$ oraz $k \leftarrow k+1$ | 10 |
| `LOADI` $i$ | $p_0 \leftarrow p_{p_i}$ oraz $k \leftarrow k+1$ | 20 |
| `STOREI` $i$ | $p_{p_i} \leftarrow p_0$ oraz $k \leftarrow k+1$ | 20 |
| `ADD` $i$ | $p_0 \leftarrow p_0 + p_i$ oraz $k \leftarrow k+1$ | 10 |
| `SUB` $i$ | $p_0 \leftarrow p_0 - p_i$ oraz $k \leftarrow k+1$ | 10 |
| `SHIFT` $i$ | $p_0 \leftarrow \lfloor 2^{p_i} \cdot p_0 \rfloor$ oraz $k \leftarrow k+1$ | 5 |
| `INC` | $p_0 \leftarrow p_0 + 1$ oraz $k \leftarrow k+1$ | 1 |
| `DEC` | $p_0 \leftarrow p_0 - 1$ oraz $k \leftarrow k+1$ | 1 |
| `JUMP` $j$ | $k \leftarrow j$ | 1 |
| `JPOS` $j$ | jeśli $p_0 > 0$ to $k \leftarrow j$, w p.p. $k \leftarrow k+1$ | 1 |
| `JZERO` $j$ | jeśli $p_0 = 0$ to $k \leftarrow j$, w p.p. $k \leftarrow k+1$ | 1 |
| `JNEG` $j$ | jeśli $p_0 < 0$ to $k \leftarrow j$, w p.p. $k \leftarrow k+1$ | 1 |
| `HALT` | zatrzymaj program | 0 |

Tablica 2: Rozkazy maszyny wirtualnej

Wszystkie przykłady oraz kod maszyny wirtualnej 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 całkowitych, która jest jednak wolniejsza w działaniu ze względu na użycie biblioteki dużych liczb).

## Przykładowe kody programów

**Przykład 1** – Binarny zapis liczby.

```
1  DECLARE
2      a, b
3  BEGIN
4      READ a;
5      IF a GEQ 0 THEN
6          WHILE a GE 0 DO
7              b ASSIGN a DIV 2;
8              b ASSIGN 2 TIMES b;
9              IF a GE b THEN
10                 WRITE 1;
11             ELSE
12                 WRITE 0;
13             ENDIF
14             a ASSIGN a DIV 2;
15         ENDWHILE
16     ENDIF
17 END
```

```
-1 # zapis binarny liczby
 0 SUB 0
 1 DEC
 2 STORE 3          # p(3) = 1
 3 GET
 4 JNEG 16          # nie if
 5 STORE 1          # p(1) = a
 6 JZERO 16         # !  warunek while
 7 SHIFT 3
 8 STORE 2          # p(2) = a/2
 9 LOAD 1
10 SUB 2
11 SUB 2
12 PUT              # wypisz a-2(a/2)
13 LOAD 2
14 STORE 1          # a = a/2
15 JUMP 6           # koniec while
16 HALT
```

**Przykład 2** – Sito Eratostenesa.

```
1  [ sito Eratostenesa ]
2  DECLARE
3      n, j, sito(2:100)
4  BEGIN
5      n ASSIGN 100;
6      FOR i FROM n DOWNTO 2 DO
7          sito(i) ASSIGN 1;
8      ENDFOR
9      FOR i FROM 2 TO n DO
10         IF sito(i) NEQ 0 THEN
11             j ASSIGN i PLUS i;
12             WHILE j LEQ n DO
13                 sito(j) ASSIGN 0;
14                 j ASSIGN j PLUS i;
15             ENDWHILE
16             WRITE i;
17         ENDIF
18     ENDFOR
19 END
```

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

### Optymalność wykonywania mnożenia i dzielenia

```
1  [ Rozklad liczby na czynniki pierwsze ]
2  DECLARE
3      n, m, reszta, potega, dzielnik
4  BEGIN
5      READ n;
6      dzielnik ASSIGN 2;
7      m ASSIGN dzielnik TIMES dzielnik;
8      WHILE n GEQ m DO
9          potega ASSIGN 0;
10         reszta ASSIGN n MOD dzielnik;
11         WHILE reszta EQ 0 DO
12             n ASSIGN n DIV dzielnik;
13             potega ASSIGN potega PLUS 1;
14             reszta ASSIGN n MOD dzielnik;
15         ENDWHILE
16         IF potega GE 0 THEN [ czy znaleziono dzielnik ]
17             WRITE dzielnik;
18             WRITE potega;
19         ELSE
20             dzielnik ASSIGN dzielnik PLUS 1;
21             m ASSIGN dzielnik TIMES dzielnik;
22         ENDIF
23     ENDWHILE
24     IF n NEQ 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 oznaczonych przez $*$):

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