

# Języki formalne i techniki translacji

## Laboratorium - Projekt (wersja $\alpha$ )

**Termin oddania: ostatnie zajęcia przed 20 stycznia 2024**

**Wysłanie do wykładowcy (MS TEAMS): przed 23:45 30 stycznia 2024**

Używając BISON-a i FLEX-a, lub innych narzędzi o podobnej funkcjonalności, 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, nieznaną nazwą procedury, ...), a w przypadku braku błędów zwracać kod na maszynę wirtualną. Kod wynikowy powinien być jak najkrótszy i wykonywać się jak najszybciej (w miarę optymalnie, mnożenie i dzielenie powinny być wykonywane w czasie logarytmicznym w stosunku do wartości argumentów). Ocena końcowa zależy od obu wielkości.

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<sup>1</sup>

kompiletor <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 Tabelicy 1 i spełniać następujące warunki:

1. Działania arytmetyczne są wykonywane na liczbach naturalnych. Wynikiem odejmowania liczby większej od mniejszej jest 0, Ponadto dzielenie przez zero powinno dać wynik 0 i resztę także 0.
2. Deklaracja `t[100]` oznacza zadeklarowanie tablicy o 100 elementach indeksowanych od 0 do 99.
3. Procedury nie mogą zawierać wywołań rekurencyjnych, parametry formalne przekazywane są przez referencje (parametry IN-OUT), zmienne używane w procedurze muszą być jej parametrami formalnymi lub być zadeklarowane wewnątrz procedury, nazwa tablicy w parametrach formalnych powinna być poprzedzona literą *T*. W procedurze można wywołać tylko procedury zdefiniowane wcześniej w kodzie programu, a jako ich parametry formalne można podać zarówno parametry formalne procedury wywołującej, jak i jej zmienne lokalne.
4. Pętla REPEAT-UNTIL kończy pracę kiedy warunek napisany za UNTIL jest spełniony (pętla wykona się przynajmniej raz).
5. Instrukcja READ czyta wartość z zewnątrz i podstawia pod zmienną, a WRITE wypisuje wartość zmiennej/liczby na zewnątrz.
6. Pozostałe instrukcje są zgodne z ich znaczeniem w większości języków programowania;
7. `pidentifier` jest opisany wyrażeniem regularnym `[_a-z]+`;
8. `num` jest liczbą naturalną w zapisie dziesiętnym (w kodzie wejściowym liczby podawane jako stałe są ograniczone do typu `long long` (64 bitowy), na maszynie wirtualnej nie ma ograniczeń na wielkość liczb, obliczenia mogą generować dowolną liczbę naturalną);

---

<sup>1</sup>Dla niektórych języków programowania należy napisać w pliku README że jest inny sposób wywołania kompilatora, np. `java kompilator` lub `python kompilator`

```

1  program_all  -> procedures main
2
3  procedures   -> procedures PROCEDURE proc_head IS declarations IN commands END
4                | procedures PROCEDURE proc_head IS IN commands END
5                |
6
7  main         -> PROGRAM IS declarations IN commands END
8                | PROGRAM IS IN commands END
9
10 commands    -> commands command
11                | command
12
13 command      -> identifier := expression;
14                | IF condition THEN commands ELSE commands ENDIF
15                | IF condition THEN commands ENDIF
16                | WHILE condition DO commands ENDWHILE
17                | REPEAT commands UNTIL condition;
18                | proc_call;
19                | READ identifier;
20                | WRITE value;
21
22 proc_head     -> pidentifier ( args_decl )
23
24 proc_call     -> pidentifier ( args )
25
26 declarations -> declarations, pidentifier
27                | declarations, pidentifier[num]
28                | pidentifier
29                | pidentifier[num]
30
31 args_decl     -> args_decl, pidentifier
32                | args_decl, T pidentifier
33                | pidentifier
34                | T pidentifier
35
36 args         -> args, pidentifier
37                | pidentifier
38
39 expression    -> value
40                | value + value
41                | value - value
42                | value * value
43                | value / value
44                | value % value
45
46 condition     -> value = value
47                | value != value
48                | value > value
49                | value < value
50                | value >= value
51                | value <= value
52
53 value        -> num
54                | identifier
55
56 identifier    -> pidentifier
57                | pidentifier[num]
58                | pidentifier[pidentifier]

```

Tablica 1: Gramatyka języka

9. Małe i duże litery są rozróżniane;
10. W programie można użyć komentarzy zaczynających się od # i obowiązujących do końca linii.

## Maszyna wirtualna

**Maszyna wirtualna** Maszyna wirtualna 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  $i = 0, 1, 2, \dots$  (z przyczyn technicznych  $i \leq 2^{62}$ ). 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 rejestrów i komórek pamięci jest nieokreślona, a licznik rozkazów  $k$  ma wartość 0. W tabelicy 2 jest podana lista rozkazów wraz z ich interpretacją i kosztem wykonania. W programie można zamieszczać komentarze w postaci: # komentarz, które sięgają do końca linii. Białe znaki w kodzie są pomijane. Przejście do nieistniejącego rozkazu lub wywołanie nieistniejącego rejestru jest traktowane jako błąd.

Rozkaz	Interpretacja	Czas
READ	pobraną liczbę zapisuje w rejestrze $r_a$ oraz $k \leftarrow k + 1$	100
WRITE	wyświetla zawartość rejestru $r_a$ oraz $k \leftarrow k + 1$	100
LOAD $x$	$r_a \leftarrow p_{r_x}$ oraz $k \leftarrow k + 1$	50
STORE $x$	$p_{r_x} \leftarrow r_a$ oraz $k \leftarrow k + 1$	50
ADD $x$	$r_a \leftarrow r_a + r_x$ oraz $k \leftarrow k + 1$	5
SUB $x$	$r_a \leftarrow \max\{r_a - r_x, 0\}$ oraz $k \leftarrow k + 1$	5
GET $x$	$r_a \leftarrow r_x$ oraz $k \leftarrow k + 1$	1
PUT $x$	$r_x \leftarrow r_a$ oraz $k \leftarrow k + 1$	1
RST $x$	$r_x \leftarrow 0$ 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
SHL $x$	$r_x \leftarrow 2 * r_x$ oraz $k \leftarrow k + 1$	1
SHR $x$	$r_x \leftarrow \lfloor r_x / 2 \rfloor$ oraz $k \leftarrow k + 1$	1
JUMP $j$	$k \leftarrow j$	1
JPOS $j$	jeśli $r_a > 0$ to $k \leftarrow j$ , w p.p. $k \leftarrow k + 1$	1
JZERO $j$	jeśli $r_a = 0$ to $k \leftarrow j$ , w p.p. $k \leftarrow k + 1$	1
STRK $x$	$r_x \leftarrow k$ oraz $k \leftarrow k + 1$	1
JUMPR $x$	$k \leftarrow r_x$	1
HALT	zatrzymaj program	0

Tabela 2: Rozkazy maszyny wirtualnej ( $x \in \{a, b, c, d, e, f, g, h\}$  i  $j \in \mathbb{N}$ )

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 naturalnych, która jest jednak wolniejsza w działaniu ze względu na użycie biblioteki dużych liczb).

## Przykładowe kody programów

### Przykład 1 – Binarny zapis liczby.

---

```
1 # Binarna postać liczby
2 PROGRAM IS
3     n, p
4 IN
5     READ n;
6     REPEAT
7         p:=n/2;
8         p:=2*p;
9         IF n>p THEN
10            WRITE 1;
11        ELSE
12            WRITE 0;
13        ENDIF
14        n:=n/2;
15    UNTIL n=0;
16 END
```

---

-2 # prosta translacja z użyciem pamięci

-1 # n -> P0, p -> P1

```
0 READ
1 RST b
2 STORE b
3 RST b
4 LOAD b
5 SHR a
6 SHL a
7 RST b
8 INC b
9 STORE b
10 RST b
11 LOAD b
12 PUT c
13 RST b
14 INC b
15 LOAD b
16 PUT d
17 GET c
18 SUB d
19 JZERO 24
20 RST a
21 INC a
22 WRITE
23 JUMP 26
24 RST a
25 WRITE
26 RST b
27 LOAD b
28 SHR a
29 RST b
30 STORE b
31 RST b
32 LOAD b
33 JPOS 3
34 HALT
```

-1 # kod zoptymalizowany

```
0 READ
1 PUT b
2 SHR a
3 SHL a
4 PUT c
5 GET b
6 SUB c
7 WRITE
8 SHR b
9 GET b
10 JPOS 2
11 HALT
```

## Przykład 2 – GCD.

```
1  PROCEDURE gcd(a,b,c) IS
2    x,y
3  IN
4    x:=a;
5    y:=b;
6    WHILE y>0 DO
7      IF x>=y THEN
8        x:=x-y;
9      ELSE
10     x:=x+y;
11     y:=x-y;
12     x:=x-y;
13   ENDIF
14 ENDWHILE
15   c:=x;
16 END
17
18 PROGRAM IS
19   a,b,c,d,x,y,z
20 IN
21   READ a;
22   READ b;
23   READ c;
24   READ d;
25   gcd(a,b,x);
26   gcd(c,d,y);
27   gcd(x,y,z);
28   WRITE z;
29 END
```

```
0  JUMP 37
1  RST b      # gcd
2  LOAD b    # x:=a
3  LOAD a
4  PUT c
5  INC b
6  LOAD b    # y:=b
7  LOAD a
8  PUT d
9  GET d     # while
10 JZERO 29  # not y>0
11 GET c     # if x>=y
12 INC a
13 SUB d
14 JZERO 19
15 GET c     # then
16 SUB d
17 PUT c
18 JUMP 28
19 GET c     # else
20 ADD d
21 PUT c
22 GET c
23 SUB d
24 PUT d
25 GET c
26 SUB d
27 PUT c
28 JUMP 9    # endwhile
29 INC b
30 LOAD b
31 PUT e
32 GET c
33 STORE e
34 INC b     # return
35 LOAD b
36 JUMPR a  # end gcd
37 RST b     # program
38 INC b
39 SHL b
40 SHL b
41 READ
42 STORE b
43 INC b
44 READ
45 STORE b
46 INC b
47 READ
48 STORE b
49 INC b
50 READ
51 STORE b
52 RST a    # call gcd(a,b,x)
53 INC a
54 SHL a
55 SHL a
56 RST b
57 STORE b
58 INC a
59 INC b
60 STORE b
61 INC a
62 INC a
63 INC a
64 INC b
65 STORE b
66 INC b   # return set
67 RST a
68 INC a
69 SHL a
70 SHL a
71 STRK c
72 ADD c
73 STORE b
74 JUMP 1  # end call
75 RST a   # call gcd(c,d,y)
76 INC a
77 SHL a
78 INC a
79 SHL a
80 RST b
81 STORE b
82 INC a
83 INC b
84 STORE b
85 INC a
86 INC a
87 INC b
88 STORE b
89 INC b   # return set
90 RST a
91 INC a
92 SHL a
93 SHL a
94 STRK c
95 ADD c
96 STORE b
97 JUMP 1  # end call
98 RST a   # call gcd(x,y,z)
99 INC a
100 SHL a
101 SHL a
102 SHL a
103 RST b
104 STORE b
105 INC a
106 INC b
107 STORE b
108 INC a
109 INC b
110 STORE b
111 INC b   # return set
112 RST a
113 INC a
114 SHL a
115 SHL a
116 STRK c
117 ADD c
118 STORE b
119 JUMP 1  # end call
120 RST b   # write z
121 INC b
122 SHL b
123 SHL b
124 INC b
125 SHL b
126 LOAD b
127 WRITE
128 HALT
```

### Przykład 3 – Sito Eratostenesa.

```

1  PROCEDURE licz(T s, n) IS
2  i, j
3  IN
4  i:=2;
5  WHILE i<=n DO
6  s[i]:=1;
7  i:=i+1;
8  ENDWHILE
9  i:=2;
10 WHILE i<=n DO
11 IF s[i]>0 THEN
12 j:=i+i;
13 WHILE j<=n DO
14 s[j]:=0;
15 j:=j+i;
16 ENDWHILE
17 ENDIF
18 i:=i+1;
19 ENDWHILE
20 END
21 PROCEDURE wypisz(T s, n) IS
22 i
23 IN
24 i:=2;
25 WHILE i<=n DO
26 IF s[i]>0 THEN
27 WRITE i;
28 ENDIF
29 i:=i+1;
30 ENDWHILE
31 END
32 PROGRAM IS
33 n, sito[100]
34 IN
35 n:=99;
36 licz(sito,n);
37 wypisz(sito,n);
38 END

```

---

```

0  JUMP 104
1  RST b # licz
2  LOAD b
3  PUT d # &s -> r_d
4  INC b
5  LOAD b
6  PUT e # &n -> r_e
7  RST a
8  INC a
9  INC a
10 PUT f # i=2 -> r_f
11 RST a # 1 -> h
12 INC a
13 PUT f
14 GET e # while i<=n
15 PUT b
16 LOAD b
17 INC a
18 SUB f
19 JZERO 27 # not i<=n
20 GET d # s[i]:=1
21 ADD f
22 PUT b
23 GET h
24 STORE b
25 INC f # i:=i+1
26 JUMP 14 # endwhile
27 RST a
28 INC a
29 INC a
30 PUT f # i=2 -> r_f
31 GET e # while i<=n
32 PUT b
33 LOAD b
34 INC a
35 SUB f
36 JZERO 62 # not i<=n
37 GET d # if s[i]>0
38 ADD f
39 PUT b
40 LOAD b
41 JZERO 60 # not s[i]>0
42 GET f
43 ADD f
44 PUT g # j=i+i -> r_g
45 GET e # while j<=n
46 PUT b
47 LOAD b
48 INC a
49 SUB g
50 JZERO 60 # not j<=n
51 GET d # s[j]:=0
52 ADD g
53 PUT b
54 RST a
55 STORE b
56 GET g # j:=j+i
57 ADD f
58 PUT g
59 JUMP 45 # endwhile
60 INC f # i:=i+1
61 JUMP 31 # endwhile
62 RST b # return
63 INC b
64 SHL b
65 SHL b
66 LOAD b
67 JUMPR a # end licz
68 RST b # wypisz
69 INC b
70 SHL b
71 SHL b
72 INC b
73 LOAD b
74 PUT d # &s -> r_d
75 INC b
76 LOAD b
77 PUT e # &n -> r_e
78 RST a
79 INC a
80 INC a
81 PUT f # i=2 -> r_f
82 GET e # while i<=n
83 PUT b
84 LOAD b
85 INC a
86 SUB f
87 JZERO 97 # not i<=n
88 GET d # if s[i]>0
89 ADD f
90 PUT b
91 LOAD b
92 JZERO 95 # not s[i]>0
93 GET f
94 WRITE
95 INC f # i:=i+1
96 JUMP 82
97 RST b # return
98 INC b
99 SHL b
100 SHL b
101 SHL b
102 LOAD b
103 JUMPR a # end wypisz
104 RST b # program
105 INC b
106 SHL b
107 SHL b
108 SHL b
109 INC b
110 RST a # 99
111 INC a
112 SHL a
113 INC a
114 SHL a
115 SHL a
116 SHL a
117 SHL a
118 INC a
119 SHL a
120 INC a
121 STORE b # n:=99
122 RST c # call licz
123 GET b
124 INC a
125 STORE c
126 INC c
127 DEC a
128 STORE c
129 SHL c
130 SHL c
131 RST a # 4
132 INC a
133 SHL a
134 SHL a
135 STRK d
136 ADD d
137 STORE c
138 JUMP 1 # end call licz
139 RST b # call wypisz
140 INC b
141 SHL b
142 SHL b
143 SHL b
144 INC b
145 RST c
146 INC c
147 SHL c
148 SHL c
149 INC c
150 GET b
151 INC a
152 STORE c
153 INC c
154 DEC a
155 STORE c
156 INC c
157 INC c
158 RST a # 4
159 INC a
160 SHL a
161 SHL a
162 STRK d
163 ADD d
164 STORE c
165 JUMP 68 # end call wypisz
166 HALT

```

## Optymalność wykonywania mnożenia i dzielenia

```
1 # Rozkład na czynniki pierwsze
2 PROCEDURE check(n,d,p) IS
3   r
4   IN
5     p:=0;
6     r:=n%d;
7     WHILE r=0 DO
8       n:=n/d;
9       p:=p+1;
10      r:=n%d;
11    ENDWHILE
12  END
13
14 PROGRAM IS
15   n,m,potega,dzielnik
16   IN
17   READ n;
18   dzielnik:=2;
19   m:=dzielnik*dzielnik;
20   WHILE n>=m DO
21     check(n,dzielnik,potega);
22     IF potega>0 THEN # jest podzielna przez dzielnik
23       WRITE dzielnik;
24       WRITE potega;
25     ENDIF
26     dzielnik:=dzielnik+1;
27     m:=dzielnik*dzielnik;
28   ENDWHILE
29   IF n!=1 THEN # ostatni dzielnik różny od 1
30     WRITE n;
31     WRITE 1;
32   ENDIF
33 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: *****; w tym i/o: 1100).
...
Uruchamianie programu.
? 12345678901
> 857
> 1
> 14405693
> 1
Skończono program (koszt: *****; w tym i/o: 500).
...
Uruchamianie programu.
? 12345678903
> 3
> 1
> 4115226301
> 1
Skończono program (koszt: *****; w tym i/o: 500).
```