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

Termin oddania: ostatnie zajęcia przed 24 stycznia 2026
Wysłanie do wykładowcy (MS TEAMS): przed 23:45 1 lutego 2026

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, nieznana nazwa 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[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.

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

1 Prosty język imperatywny

Język powinien być zgodny z gramatyką zamieszczoną w Tabeli 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 `tab[10:30]` oznacza zadeklarowanie tablicy o 21 elementach indeksowanych od 10 do 30; identyfikator `tab[i]` oznacza odwołanie do `i`-tego elementu tablicy `tab`; deklaracja zawierająca pierwszą liczbę większą od drugiej powinna być zgłaszana jako błąd.
3. Procedury nie mogą zawierać wywołań rekurencyjnych, parametry formalne przekazywane są przez referencje (parametry IN-OUT, tryb domyślny), zmienne używane w procedurze muszą być jej parametrami formalnymi lub być zadeklarowane wewnątrz procedury; nazwa tablicy w parametrach formalnych powinna być poprzedzona literą *T*.
4. 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.
5. Poprzedzenie w parametrach formalnych zmiennej literą `I` oznacza, że ma być traktowana w procedurze jak stała (nie wolno jej modyfikować, może być przekazana do podprocedury tylko w miejscu także oznaczonym przez `I`). Poprzedzenie przez `O` oznacza, że zmienna ma nieokreśloną wartość (nie wolno jej czytać przed podstawieniem wartości, nie może być przekazana do podprocedury w miejscu oznaczonym przez `I`).
6. Pętla `FOR` ma iterator lokalny, przyjmujący wartości od wartości stojącej po `FROM` do wartości stojącej po `TO/DOWNTO` kolejno w odstępie +1 lub odpowiednio w odstępie -1; 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); iterator pętli `FOR` nie może być modyfikowany wewnątrz pętli (kompilator w takim przypadku powinien zgłaszać błąd).
7. Pętla `REPEAT-UNTIL` kończy pracę kiedy warunek napisany za `UNTIL` jest spełniony (pętla wykona się przynajmniej raz).
8. Instrukcja `READ` czyta wartość z zewnątrz i podstawia pod zmienną, a `WRITE` wypisuje wartość zmiennej/liczby na zewnątrz.
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 (w kodzie wejściowym liczby podawane jako stałe są ograniczone do typu 64 bitowego, na maszynie wirtualnej nie ma ograniczeń na wielkość liczb, obliczenia mogą generować dowolną liczbę naturalną).
12. Małe i duże litery są rozróżniane.
13. W programie można użyć komentarzy zaczynających się od # i obowiązujących do końca linii.

```
program_all   -> procedures main

procedures    -> procedures PROCEDURE proc_head IS declarations IN commands END
               | procedures PROCEDURE proc_head IS IN commands END
               |

main          -> PROGRAM IS declarations IN commands END
               | PROGRAM IS IN commands END

commands      -> commands command
               | command

command       -> identifier := expression;
               | IF condition THEN commands ELSE commands ENDIF
               | IF condition THEN commands ENDIF
               | WHILE condition DO commands ENDWHILE
               | REPEAT commands UNTIL condition;
               | FOR pidentifier FROM value TO value DO commands ENDFOR
               | FOR pidentifier FROM value DOWNTO value DO commands ENDFOR
               | proc_call;
               | READ identifier;
               | WRITE value;

proc_head     -> pidentifier ( args_decl )

proc_call     -> pidentifier ( args )

declarations  -> declarations, pidentifier
               | declarations, pidentifier[num:num]
               | pidentifier
               | pidentifier[num:num]

args_decl     -> args_decl, type pidentifier
               | type pidentifier

type          -> T | I | O |

args          -> args, pidentifier
               | pidentifier

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]
```

Tabela 1: Gramatyka języka

2 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 $i = 0, 1, 2, ...$ (z przyczyn technicznych $i \leqslant 2^{62}$). Maszyna pracuje na liczbach naturalnych.

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 Tabeli 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` j	$r_a \leftarrow p_j$ oraz $k \leftarrow k+1$	50
`STORE` j	$p_j \leftarrow r_a$ oraz $k \leftarrow k+1$	50
`RLOAD` x	$r_a \leftarrow p_{r_x}$ oraz $k \leftarrow k+1$	50
`RSTORE` 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
`SWP` x	$r_a \leftrightarrow r_x$ oraz $k \leftarrow k+1$	5
`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
`CALL` j	$r_a \leftarrow k+1$ oraz $k \leftarrow j$	1
`RTRN`	$k \leftarrow r_a$	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 plikuach `labor4.zip` i `mw2025.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).

3 Przykładowe kody programów

3.1 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
-1 # n -> P0, p -> P1
 0 READ     # READ n
 1 STORE 0
 2 LOAD 0   # p:=n/2
 3 SHR a
 4 STORE 1
 5 LOAD 1   # p:=2*p
 6 SHL a
 7 STORE 1
 8 LOAD 1   # IF n>p
 9 SWP b
10 LOAD 0
11 SUB b
12 JZERO 17# skok ELSE
13 RST a    # WRITE 1
14 INC a
15 WRITE
16 JUMP 19 # skok ENDIF
17 RST a    # WRITE 0
18 WRITE
19 LOAD 0   # n:=n/2
20 SHR a
21 STORE 0
22 LOAD 0   # skok R-U n>0
23 JPOS 2
24 HALT
```

```
-1 # kod zoptymalizowany
 0 READ
 1 SWP b
 2 RST a
 3 ADD b
 4 SHR b
 5 SHL b
 6 SUB b
 7 WRITE
 8 SHR b
 9 RST a
10 ADD b
11 JPOS 4
12 HALT
```

3.2 GCD

```
1  PROCEDURE gcd(I a,I b,O 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 43 # skok do programu
 1  STORE 0 # gcd
 2  LOAD 1  # x:=a
 3  RLOAD a
 4  STORE 4
 5  LOAD 2  # y:=b
 6  RLOAD a
 7  STORE 5
 8  LOAD 5  # WHILE
 9  JZERO 37        # skok za ENDWHILE
10  LOAD 4  # IF x>=y (y-x)
11  SWP b
12  LOAD 5
13  SUB b
14  JPOS 21 # skok ELSE
15  LOAD 5  # THEN x:=x-y
16  SWP b
17  LOAD 4
18  SUB b
19  STORE 4
20  JUMP 36 # skok za ENDIF
21  LOAD 5  # ELSE x:=x+y
22  SWP b
23  LOAD 4
24  ADD b
25  STORE 4
26  LOAD 5  # y:=x-y
27  SWP b
28  LOAD 4
29  SUB b
30  STORE 5
31  LOAD 5  # x:=x-y
32  SWP b
33  LOAD 4
34  SUB b
35  STORE 4
36  JUMP 8  # ENDWHILE
37  LOAD 3  # c:=x
38  SWP b
39  LOAD 4
40  RSTORE b
41  LOAD 0  # return
42  RTRN
43  READ    # READ a
44  STORE 6
45  READ    # READ b
46  STORE 7
47  READ    # READ c
48  STORE 8
49  READ    # READ d
50  STORE 9
51  RST a   # call gcd(a,b,x)
52  INC a
53  SHL a
54  INC a
55  SHL a
56  STORE 1
57  INC a
58  STORE 2
59  INC a
60  INC a
61  INC a
62  STORE 3
63  CALL 1
64  RST a   # call gcd(c,d,y)
65  INC a
66  SHL a
67  SHL a
68  SHL a
69  STORE 1
70  INC a
71  STORE 2
72  INC a
73  INC a
74  STORE 3
75  CALL 1
76  RST a   # call gcd(x,y,z)
77  INC a
78  SHL a
79  SHL a
80  INC a
81  SHL a
82  STORE 1
83  INC a
84  STORE 2
85  INC a
86  STORE 3
87  CALL 1
88  LOAD 12 # write z
89  WRITE
90  HALT
```

3.3 Sito Eratostenesa

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

```
0  JUMP 94 # skok do programu
1  STORE 0 # licz
2  LOAD 1  # r[b]:=&s[0]
3  SWP b
4  RST a   # i:=2
5  INC a
6  SHL a
7  STORE 4
8  LOAD 2  # i':=n+1-2
9  RLOAD a
10 DEC a
11 STORE 5
12 LOAD 5  # FOR
13 JZERO 26# skok za FOR
14 DEC a   # i'--
15 STORE 5
16 LOAD 4  # s[i]:=1
17 ADD b
18 SWP c
19 RST a
20 INC a
21 RSTORE c
22 LOAD 4  # i++
23 INC a
24 STORE 4
25 JUMP 12 # ENDFOR
26 RST a   # i:=2
27 INC a
28 SHL a
29 STORE 4
30 LOAD 2  # i':=n+1-2
31 RLOAD a
32 DEC a
33 STORE 5
```

```
34  LOAD 5  # FOR
35  JZERO 66# skok za FOR
36  DEC a   # i'--
37  STORE 5
38  LOAD 4  # IF s[i]>0
39  ADD b
40  RLOAD a
41  JZERO 62# skok za IF
42  LOAD 4  # j:=i+i;
43  ADD a
44  STORE 3
45  LOAD 2  # WHILE j<=n
46  RLOAD a
47  SWP c
48  LOAD 3
49  SUB c
50  JPOS 62 # skok za WHILE
51  LOAD 3  # s[j]:=0
52  ADD b
53  SWP c
54  RST a
55  RSTORE c
56  LOAD 4  # j:=j+i
57  SWP c
58  LOAD 3
59  ADD c
60  STORE 3
61  JUMP 45 # ENDWHILE
62  LOAD 4  # i++
63  INC a
64  STORE 4
65  JUMP 34 # ENDFOR
66  LOAD 0  # return
67  RTRN
68  STORE 6 # wypisz
69  LOAD 7  # r[b]:=&s
70  SWP b
71  LOAD 8  # i:=n
72  RLOAD a
73  STORE 9
74  LOAD 8  # i':=n-2+1
75  RLOAD a
76  DEC a
77  STORE 10
78  LOAD 10 # FOR
79  JZERO 92# skok za FOR
80  DEC a   # i'--
81  STORE 10
82  LOAD 9  # IF s[i]>0
83  ADD b
84  RLOAD a
85  JZERO 88# ENDIF
86  LOAD 9  # WRITE i
87  WRITE
88  LOAD 9
89  DEC a   # i--
90  STORE 9
91  JUMP 78 # ENDFOR
92  LOAD 6  # return
93  RTRN
94  RST a   # program n:=100
95  INC a
96  SHL a
97  INC a
98  SHL a
99  SHL a
100 SHL a
101 INC a
102 SHL a
103 SHL a
104 STORE 11
105 RST a   # call licz(sito,n)
106 INC a
107 SHL a
108 SHL a
109 INC a
110 SHL a
111 STORE 1
112 INC a
113 STORE 2
114 CALL 1
115 RST a   # call wypisz(sito,n)
116 INC a
117 SHL a
118 SHL a
119 INC a
120 SHL a
121 STORE 7
122 INC a
123 STORE 8
124 CALL 68
125 HALT
```

3.4 Optymalność wykonywania mnożenia i dzielenia

Operacje mnożenia, dzielenia i liczenia reszty powinny być zaimplementowane tak, aby wykonywały się w czasie logarytmicznym względem wielkości argumentów (nie może to być proste dodawanie czy odejmowanie w pętli).

```
1  # Rozkład na czynniki pierwsze
2  PROCEDURE check(n,I d,O 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).
```