

Opis maszyny wirtualnej i wymagania kompilatora

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. $=$ \neq $<$ $>$ \leq \geq oznaczają odpowiednio relacje $=$, \neq , $<$, $>$, \leq i \geq 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 odstępnie $+1$ lub w odstępnie -1 jeśli użyto słowa DOWNT0;
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. pętla REPEAT-UNTIL kończy pracę kiedy warunek napisany za UNTIL jest spełniony (pętla wykona się przynajmniej raz);

¹Dla 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 := expression;
13             | IF condition THEN commands ELSE commands ENDIF
14             | IF condition THEN commands ENDIF
15             | WHILE condition DO commands ENDWHILE
16             | REPEAT commands UNTIL condition;
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 + value
24             | value - value
25             | value * value
26             | value / value
27             | value % value
28
29 condition    -> value = value
30             | value != value
31             | value < value
32             | value > value
33             | value <= value
34             | value >= value
35
36 value        -> num
37             | identifier
38
39 identifier    -> pidentifier
40             | pidentifier(pidentifier)
41             | pidentifier(num)

```

Tablica 1: Gramatyka języka

Rozkaz	Interpretacja	Czas
GET x	pobraną liczbę zapisuje w komórce pamięci p_{r_x} oraz $k \leftarrow k + 1$	100
PUT x	wyświetla zawartość komórki pamięci p_{r_x} oraz $k \leftarrow k + 1$	100
LOAD $x \ y$	$r_x \leftarrow p_{r_y}$ oraz $k \leftarrow k + 1$	20
STORE $x \ y$	$p_{r_y} \leftarrow r_x$ oraz $k \leftarrow k + 1$	20
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
RESET 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
SHR x	$r_x \leftarrow \lfloor r_x / 2 \rfloor$ oraz $k \leftarrow k + 1$	1
SHL x	$r_x \leftarrow r_x * 2$ oraz $k \leftarrow k + 1$	1
JUMP j	$k \leftarrow k + j$	1
JZERO $x \ j$	jeśli $r_x = 0$ to $k \leftarrow k + j$, w p.p. $k \leftarrow k + 1$	1
JODD $x \ j$	jeśli r_x nieparzyste to $k \leftarrow k + j$, w p.p. $k \leftarrow k + 1$	1
HALT	zatrzymaj program	0

Tablica 2: Rozkazy maszyny wirtualnej ($x, y \in \{a, b, c, d, e, f\}$ i $j \in \mathbb{Z} \setminus \{0\}$)

10. instrukcja READ czyta wartość z zewnątrz i podstawia pod zmienną, a WRITE wypisuje wartość zmiennej/liczby na zewnątrz;
11. pozostałe instrukcje są zgodne z ich znaczeniem w większości języków programowania;
12. `pidentifier` jest opisany wyrażeniem regularnym `[_a-z]+`;
13. `num` jest liczbą naturalną 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 naturalną);
14. małe i duże litery są rozróżniane;
15. w programie można użyć komentarzy postaci: `[komentarz]`, które nie mogą być zagnieżdżone.

Maszyna wirtualna Maszyna wirtualna składa się z 6 rejestrów ($r_a, r_b, r_c, r_d, r_e, r_f$), 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 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. W tablicy 2 jest podana lista rozkazów wraz z ich interpretacją i kosztem wykonania. W programie można zamieszczać komentarze w postaci: `(komentarz)`, które nie mogą być zagnieżdżone. Białe znaki w kodzie są pomijane. Przejście do nieistniejącego rozkazu lub wywołanie nieistniejącego rejestru jest traktowane jako błąd.

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 DECLARE
2     n,p
3 BEGIN
4     READ n;
5     REPEAT
6         p:=n/2;
7         p:=2*p;
8         IF n>p THEN
9             WRITE 1;
10        ELSE
11            WRITE 0;
12        ENDIF
13        n:=n/2;
14    UNTIL n=0;
15 END
```

```
-1 (zapis binarny liczby)
0 RESET a          (p[0] <- 0)
1 STORE a a
2 INC a            (p[1] <- 1)
3 STORE a a
4 INC a            (p[2] <- read n)
5 GET a
6 LOAD a a         (a <- p[2])
7 RESET b          (b <- 0)
8 ADD b a          (b <- a/2)
9 SHR b
10 SHL b           (b <- b*2)
11 RESET c          (c <- a-b [a>b])
12 ADD c a
13 SUB c b
14 JZERO c 5        (not a>b)
15 RESET d
16 INC d
17 PUT d            (write p[1])
18 JUMP 3
19 RESET d
20 PUT d            (write p[0])
21 SHR a            (a <- a/2)
22 JZERO a 2        (a=0 - until)
23 JUMP -16         (goto repeat)
24 HALT
```

```
-1 (zapis binarny liczby - zoptymalizowany)
0 RESET a          (p[0] <- 0)
1 STORE a a
2 INC a            (p[1] <- 1)
3 STORE a a
4 INC a            (p[2] <- read n)
5 GET a
6 LOAD a a         (a <- p[2])
7 RESET b
8 JODD a 3
9 PUT b            (write p[0])
10 JUMP 4
11 INC b
12 PUT b           (write p[1])
13 DEC b
14 SHR a            (a <- a/2)
15 JZERO a 2        (a=0 - until)
16 JUMP -8          (goto repeat)
17 HALT
```

Przykład 2 – Sito Eratostenesa.

```
1  [ sito Eratostenesa ]
2  DECLARE
3      n, j, sito(2:100)
4  BEGIN
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	RESET a	(generowanie 100)	23	INC c	
1	INC a		24	RESET d	(licznik dla for)
2	SHL a		25	ADD d a	
3	INC a		26	DEC d	(licznik--)
4	SHL a		27	JZERO d 18	(wyjście z for)
5	SHL a		28	LOAD e c	
6	SHL a		29	JZERO e 14	(sito[i]=0)
7	INC a		30	RESET f	(j:=i)
8	SHL a		31	ADD f c	
9	SHL a		32	ADD f c	(j+=i)
10	RESET b	(generowanie 1)	33	RESET b	(j<=n ?)
11	INC b		34	ADD b a	
12	RESET c	(i:=n)	35	INC b	
13	ADD c a		36	SUB b f	
14	RESET d	(licznik dla for)	37	JZERO b 4	(wyjście z while)
15	ADD d a		38	RESET b	(sito[j]:=0)
16	DEC d	(licznik--)	39	STORE b f	
17	JZERO d 4	(wyjście z for)	40	JUMP -8	(powrót do while)
18	STORE b c	(sito[i]:=1)	41	STORE c b	(write i)
19	DEC c	(i--)	42	PUT b	
20	JUMP -4	(powrót do for)	43	INC c	(i++)
21	RESET c	(i:=2)	44	JUMP -18	(powrót do for)
22	INC c		45	HALT	

Optymalność wykonywania mnożenia i dzielenia

```
1  [ Rozkład liczby na czynniki pierwsze ]
2  DECLARE
3      n, m, reszta, potega, dzielnik
4  BEGIN
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 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).
```