

Języki formalne i techniki translacji

Laboratorium - Projekt (wersja α)

Termin oddania: ostatnie zajęcia przed 21 stycznia 2023

Wysłanie do wykładowcy (MS TEAMS): przed 23:45 28 stycznia 2023

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 następująco¹

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 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. 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. 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.
3. Pętla REPEAT-UNTIL kończy pracę kiedy warunek napisany za UNTIL jest spełniony (pętla wykona się przynajmniej raz).
4. Instrukcja READ czyta wartość z zewnątrz i podstawia pod zmienną, a WRITE wypisuje wartość zmiennej/liczby na zewnątrz.
5. Pozostałe instrukcje są zgodne z ich znaczeniem w większości języków programowania;
6. `identyfikator` jest opisany wyrażeniem regularnym `[_a-z]+`;
7. `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);
8. Małe i duże litery są rozróżniane;
9. W programie można użyć komentarzy postaci: `[komentarz]`, które nie mogą być zagnieżdżone.

¹Dla 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 VAR declarations BEGIN commands END
4                | procedures PROCEDURE proc_head IS BEGIN commands END
5                |
6
7  main         -> PROGRAM IS VAR declarations BEGIN commands END
8                | PROGRAM IS BEGIN 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_head;
19                | READ identifier;
20                | WRITE value;
21
22 proc_head     -> identifier ( declarations )
23
24 declarations  -> declarations, identifier
25                | identifier
26
27 expression    -> value
28                | value + value
29                | value - value
30                | value * value
31                | value / value
32                | value % value
33
34 condition     -> value = value
35                | value != value
36                | value > value
37                | value < value
38                | value >= value
39                | value <= value
40
41 value         -> num
42                | identifier

```

Tablica 1: Gramatyka języka

Maszyna wirtualna Maszyna wirtualna składa się z licznika rozkazów k oraz ciągu komórek pamięci p_i , dla $i = 0, 1, 2, \dots$ (z przyczyn technicznych $i \leq 2^{62}$). Komórka p_0 pełni rolę akumulatora. 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 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 jest traktowane jako błąd.

Rozkaz	Interpretacja	Czas
GET i	pobraną liczbę zapisuje w komórce pamięci p_i oraz $k \leftarrow k + 1$	100
PUT i	wyświetla zawartość komórki pamięci p_i 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 $k \leftarrow k + 1$	10
STOREI i	$p_{p_i} \leftarrow p_0$ oraz $k \leftarrow k + 1$	10
ADD i	$p_0 \leftarrow p_0 + p_i$ oraz $k \leftarrow k + 1$	10
SUB i	$p_0 \leftarrow \max\{p_0 - p_i, 0\}$ oraz $k \leftarrow k + 1$	10
ADDI i	$p_0 \leftarrow p_0 + p_{p_i}$ oraz $k \leftarrow k + 1$	10
SUBI i	$p_0 \leftarrow \max\{p_0 - p_{p_i}, 0\}$ oraz $k \leftarrow k + 1$	10
SET x	$p_0 \leftarrow x$ oraz $k \leftarrow k + 1$	10
HALF	$p_0 \leftarrow \lfloor \frac{p_0}{2} \rfloor$ oraz $k \leftarrow k + 1$	5
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
JUMPI i	$k \leftarrow p_i$	1
HALT	zatrzymaj program	0

Tablica 2: Rozkazy maszyny wirtualnej

Zamieszczone poniżej przykłady oraz kod maszyny wirtualnej napisany w języku 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  VAR n, p
4  BEGIN
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
```

-1 [prosta translacja]

```
0 GET 1
1 LOAD 1
2 HALF
3 STORE 2
4 LOAD 2
5 ADD 0
6 STORE 2
7 LOAD 1
8 SUB 2
9 JZERO 13
10 SET 1
11 PUT 0
12 JUMP 15
13 SET 0
14 PUT 0
15 LOAD 1
16 HALF
17 STORE 1
18 LOAD 1
19 JPOS 1
20 HALT
```

-1 [kod zoptymalizowany]

```
0 GET 1
1 LOAD 1
2 HALF
3 ADD 0
4 STORE 2
5 LOAD 1
6 SUB 2
7 PUT 0
8 LOAD 1
9 HALF
10 STORE 1
11 JPOS 2
12 HALT
```

Przykład 2

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

0	JUMP	31		32	GET	12
1	LOADI	1	[swap]	33	GET	13
2	STORE	3		34	GET	14
3	LOADI	2		35	SET	11 [call gcd]
4	STOREI	1		36	STORE	5
5	LOAD	3		37	SET	12
6	STOREI	2		38	STORE	6
7	JUMPI	4		39	SET	15
8	LOADI	5	[gcd]	40	STORE	7
9	STORE	8		41	SET	44
10	LOADI	6		42	STORE	10
11	STORE	9		43	JUMP	8
12	LOAD	9	[while]	44	SET	13 [call gcd]
13	JZERO	28		45	STORE	5
14	SUB	8	[if]	46	SET	14
15	JPOS	20		47	STORE	6
16	LOAD	8	[then]	48	SET	16
17	SUB	9		49	STORE	7
18	STORE	8		50	SET	53
19	JUMP	27		51	STORE	10
20	SET	8	[else call swap]	52	JUMP	8
21	STORE	1		53	SET	15 [call gcd]
22	SET	9		54	STORE	5
23	STORE	2		55	SET	16
24	SET	27		56	STORE	6
25	STORE	4		57	SET	17
26	JUMP	1		58	STORE	7
27	JUMP	12	[endif endwhile]	59	SET	62
28	LOAD	8		60	STORE	10
29	STOREI	7		61	JUMP	8
30	JUMPI	10		62	PUT	17
31	GET	11	[program]	63	HALT	

Optymalność wykonywania mnożenia i dzielenia

```
1  [Rozkład na czynniki pierwsze]
2  PROCEDURE check(n,d,p) IS
3  VAR r
4  BEGIN
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 VAR n,m,potega,dzielnik
16 BEGIN
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 * (plus-minus 1)):

```
...
Uruchamianie programu.
? 1234567890
> 2
> 1
> 3
> 2
> 5
> 1
> 3607
> 1
> 3803
> 1
Skończono program (koszt: ** *** ***; w tym i/o: 1 100).
...
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).
```