INTERPRETER ATARI BASIC

Gabriela Ossowska

Przykład 1.

```
FUNCTION MULTIPLY[A,B]
LET C=A*B
RETURN C

BEGIN
REM OBLICZANIE ILOCZYNU DWOCH LICZB
PRINT "PROGRAM OBLICZA ILOCZYN DWOCH LICZB"
PRINT "PODAJ DWIE DOWOLNE LICZBY";:INPUT X,Y
? X;"*";Y;"=";X*Y;MULTIPLY[X,Y]
END
```

Instrukcje są wykonywane po kolei od BEGIN do END. Powyżej można definiować swoje funkcje i procedury. Funkcja zaczyna się słowem FUNCTION, a kończy RETURN [zwracana zmienna]. Każdą instrukcję kończy znak nowej linii lub „:”.

REM rozpoczyna linię zawierającą komentarz.
„:” rozdziela instrukcje zawarte w jednej linii - jest równoważne przejściu do nowej linii.
„;” powoduje, że kursor nie przechodzi do nowej linii po wykonaniu PRINT i łączy ciągi do wydrukowania.

PRINT i ? są sobie równoważne i oznaczają „drukuj”.
Zmienne całkowite X i Y są definiowane we wbudowanej funkcji INPUT. Funkcja czeka na podanie przez użytkownika wartości, które zostaną przypisane zmiennym. LET w MULTIPLY służy do zdefiniowania zmiennej całkowitej C.

Zmienne mogą być typu całkowitego lub być napisami. Napis od liczby całkowitej odróżnia „$”:
W – liczba całkowita,
W$ - napis.

Funkcja MULTIPLY oczekuje dwóch zmiennych całkowitych i zwraca liczbę całkowitą.

Zasięg zmiennych obejmuje blok funkcji lub procedury albo główną część programu – zależnie od tego, gdzie są zdefiniowane. Zasięg zmiennych A, B i C obejmuje funkcję MULTIPLY, zasięg X i Y – część od BEGIN do END.

Przykład 2.

```
REM OBLICZANIE SILNI

FUNCTION FACTORIAL[X]
LET F=1
IF X<2 THEN
```

```
GOTO RET
ENDIF
FOR A=2 TO X;F=F*A;NEXT A
RET# RETURN F

BEGIN
? „PODAJ LICZBE”;:INPUT[X]
? „SILNIA Z ”;X;„WYNOSI ”;FACTORIAL[X]
END
```

„FOR A=2 TO X;F = F * A;NEXT A” - przykład pętli for: oblicza silnię dla liczb >= 2. Jednocześnie jest definiowana zmienna A.

„IF X<2 THEN GOTO RET ENDIF” - przykład instrukcji warunkowej: jeżeli X<2, idź do linii oznaczonej etykietą „RET”. Etykiety kończy „#”.

Przykład 3.

Wersja z IF-ELSIF-ELSE-ENDIF i śledzeniem iteracji:

```
REM OBLICZANIE SILNI

FUNCTION FACTORIAL[X]
LET F=1
IF X==0 THEN
? „ZERO!”
ELSIF X==1 THEN
? „JEDEN!”
ELSE
FOR A=2 TO X
F = F * A
? „F = ”;F
NEXT A
ENDIF
RETURN F

BEGIN
? „PODAJ LICZBE”;:INPUT[X]
? „LICZENIE SILNI: ”
FACTORIAL[X]
END
```

Instrukcja warunkowa może zawierać dowolnie wiele ‘ELSE IF’; po ‘ELSE IF’ musi nastąpić ‘ELSE’. Instrukcja warunkowa kończy się ‘ENDIF’. W głównym bloku (od ‘BEGIN’ do ‘END’) i w bloku funkcji FACTORIAL używane są zmienne o tej samej nazwie ‘X’, które są jednak dwiema różnymi zmiennymi – każdy blok ma swoją tablicę symboli.

Przykład 4.

```
FUNCTION CONC[A$,B$,N]$
DIM C$(LEN(A$)+LEN(B$)-N)
C$=A$
C$(LEN(A$))=B$(N)
RETURN C$

BEGIN
DIM A$(10),B$(10)
? „PODAJ NAPIS 1 ”;:READ A$
? „PODAJ NAPIS 2 ”;:READ B$
? „PODAJ LICZBE ”;:INPUT N
? CONC[A$, B$,N]
END
```

Program łączy w jeden napis cały NAPIS1 i część napisu NAPIS2, zaczynając od znaku na pozycji N. Pozycje w napisach są numerowanie od 0. „$” na końcu nagłówka funkcji oznacza, że zwraca napis.

DIM A$(10),B$(10) – zdeklarowanie pustych napisów z ich maksymalną długością.
LEN(A$) - wykorzystanie wbudowanej funkcji zwracającej długość napisu A
DIM C$(LEN(A$)+LEN($B)-N) – deklaracja napisu o maksymalnej długości równej długości A + długości B zaczynając od pozycji N
C$=A$ - przepisanie A do C
C$(LEN(A$))=B$(N) – przepisanie znaków z B z pozycji od N do końca B za znakami przepisanymi z A

Przykład 5.

```
SUB _SUM[X,Y]
LET Z=0
Z=X+Y
? X;"+";Y;"=";Z
RETURN

BEGIN
REM OBLICZANIE SUMY DWOCH LICZB
PRINT "PODAJ DWIE DOWOLNE LICZBY ";:INPUT X,Y
LET Z=0
GOSUB SUM
? „Z=”;Z
PRINT "PODAJ DWIE DOWOLNE LICZBY ";:INPUT W,V
_SUM[W,V]
? „Z=”;Z
END

REM SUMA
SUM# Z=X+Y
? X;"+";Y;"=",Z
RETURN
```

Program pokazuje wykorzystanie procedur na dwa sposoby. Procedura może być nienazwana i zapisana za END, zaczynając od etykiety, a kończąc na RETURN. Wywołuje się ją przez GOSUB z nazwą etykiety. Dla procedur anonimowych nie jest tworzona nowa tablica symboli, co oznacza, że zmienne za i przed END w głównej części programu mają ten sam zasięg. Można też zdefiniować procedurę podobnie, jak funkcję, z SUB zamiast FUNCTION i bez wartości zwracanej. Zasięg zmiennych nie wychodzi wtedy poza blok procedury.

Przykładowy przebieg:
PODAJ DWIE DOWOLNE LICZBY 1
2
1+2=3
3
PODAJ DWIE DOWOLNE LICZBY 20
11
20+11=31
3

Gramatyka

program = { predefined } ‘BEGIN’ { instructions } ‘END’ { instructions ‘RETURN’ } ;

predefined = stringFunction | intFunction | procedure ;

intFunction = ‘FUNCTION’ funSignature
instructions
‘RETURN’ artmExpr ;

stringFunction = ‘FUNCTION$’ funSignature
instructions
‘RETURN’ stringArg ;

procedure = ‘SUB’ funSignature
instructions
‘RETURN’ ;

for = ‘FOR’ intAssignment ‘TO’ artmExpr
instructions
‘NEXT’ id ;

if = ‘IF’ condition ‘THEN’
instructions
[{ ‘ELSIF’ condition ‘THEN’
instructions }
‘ELSE’
instructions]
‘ENDIF’ ;

instructions = [label] instruction { [label] instruction } ;

instruction = intDefinition | stringDeclaration | intAssignment | stringAssignment | input | print | read | goto | gosub | funSignature | if | for ;

condition = comp { (‘AND’ | ‘OR’) comp } ;

comp = logTerm { (‘>’ | ‘<’ | ‘=<’ | ‘>=’ | ‘==’) logTerm } ;

logTerm = artmExpr | (‘(‘ condition ‘)’) ;

stringDeclaration = ‘DIM’ substringOrDecl { ‘,’ substringOrDecl } ;

intDefinition = [‘LET’] intAssignment { ‘,’ intAssignment } ;

intAssignment = id ‘=’ artmExpr;

stringAssignment = (stringId | substringOrDecl) ‘=’ stringArg ;

len = ‘LEN’ ‘(‘ stringArg ‘)’;

input = ‘INPUT’ id { ‘,’ id } ;

print = ‘PRINT’ arg (SEMICOLON arg)* SEMICOLON? ;

read = ‘READ’ stringId { ‘,’ stringId } ;

goto = ‘GOTO’ id ;

gosub = ‘GOSUB’ id ;

label = id ‘#’;

stringArg = string | stringId | substringOrDecl | funSignature;

funSignature = id callArgs ;

callArgs = ‘[‘ [{ (artmExpr | stringArg) ‘,’ } (artmExpr | stringArg)] ‘]’;

substringOrDecl = strgingId ‘(‘ artmExpr ‘)’ ;

artmExpr = additiveExpr { (‘+’ | ‘-‘) additiveExpr } ;

additiveExpr = multExpression { (‘*’ | ‘/’) multExpression } ;

multExpression = [‘-‘] term ;

term = number | id | funSignature | len | (‘(‘ artmExpr ‘)’) ;

stringId = id ‘$’

id = letter {alpha}

number = digit {digit};

digit = ‘0’ | ‘1’ | ‘... | ‘9’;

alpha = ? znak alfanumeryczny ?

letter = * litera *

nl = ? znak nowej linii ? | ':'

Wejście/wyjście, błędy

Wejście: interpreter jest uruchamiany w konsoli, po uruchomieniu czeka na polecenia. Może interpretować polecenia wpisywane bezpośrednio do konsoli lub program zapisany w pliku, uruchamiany poleceniem RUN [nazwa]. Wyniki domyślnie są wypisywane w konsoli, mogą być zapisywane w pliku przy uruchomieniu przez RUN [nazwa] | [nazwa_pliku].
W przypadku napotkania błędu interpreter informuje, na której linii się zatrzymał ('ERROR AT LINE …') i przestaje wykonywać instrukcje.

Tablice symboli

Zmienne zdefiniowane w głównej części programu będą trzymane w globalnej tablicy symboli, a w oddzielnej tablicy nazwy funkcji i procedur. Każdy blok funkcji i nazwanej procedury ma własną tablicę symboli. Interpreter najpierw będzie szukał zmiennej w tablicy danej funkcji albo procedury, a w drugiej kolejności w tablicy zmiennych globalnych.

Nazwa zmiennej	Typ zmiennej	Wartość
X	INTEGER	5
Y	INTEGER	10
NAPIS1$	STRING	ffffffff

Nazwa	Typ	Argumenty	Typ zwracanej wartości	Własna tablica symboli (zmiennych)
CONC	FUNC	A$,B$,N	STRING	Tab_zm1
MULTIPLY	FUNC	A,B	INTEGER	Tab_zm2
_SUM	PROC	X,Y		Tab_zm3

Komunikacja między modułami

Lekser rozpoznaje tokeny typów:

PLUS
MINUS
MULTI_OPERATOR (*, /)
LOG_OPERATOR (AND, OR)
ASSIGN (=)
COMP_OPERATOR (<, >=, ==, …)

SEMICOLON
COLON
NUMBER_SIGN
LEFT_PARENTHESES
RIGHT_PARENTHESES
RIGHT_BRACKET
LEFT_BRACKET
DOLLAR
STRING_ID (ID '$' – nazwy zmiennych string i funkcji zwracających napisy)
ID (nazwy zmiennych liczbowych i funkcji zwracających liczby oraz procedur)
STRING
NUMBER
LETTER
DIGIT
COMMENT (pomijany w interpretacji)

oraz tokeny odpowiadające słowom kluczowym:
BEGIN, END, RETURN, FUNCTION, FUNCTION$ (funkcja zwracająca string), SUB, FOR, TO, NEXT, IF, THEN, ELSIF, ELSE, ENDIF, LET, DIM, LEN, INPUT, PRINT, PRINT, READ, GOTO, GOSUB.

Parser przekazuje do analizatora semantycznego drzewo obiektów reprezentujących instrukcje. Obiekt zawiera nazwę instrukcji, listę argumentów, fragment drzewa instrukcji, gdzie dana instrukcja jest korzeniem i metodę, która przejdzie po drzewie, wykonując instrukcje-dzieci.