INTERPRETER ATARI BASIC

Gabriela Ossowska

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, zaczynające się słowem FUNCTION, a kończące 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.

2.

```
REM OBLICZANIE SILNI

FUNCTION FACTORIAL[X]
LET F=1
IF X<2 THEN GOTO RET
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” - przykład instrukcji warunkowej: jeżeli X<2, idź do linii oznaczonej etykietą „RET”. Etykiety na początku linii kończy „#”.

Wersja z blokowym 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
```

Instrukcje warunkowe traktowane są w następujący sposób:
IF X>0 AND Y>0 THEN ? „DODATNIE” - po THEN nie było znaku nowej linii, więc jeżeli warunek jest spełniony, wydrukuj „DODATNIE” i idź do następnej linii. Konstrukcja nie przewiduje użycia ELSE, ELSIF.
Jeżeli po THEN jest znak nowej linii, to oczekiwane jest zakończenie przez ENDIF i można wykorzystać ELSIF i ELSE.

3.

```
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

4.

```
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. 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 = {function | procedure}, „BEGIN”, {instruction|block}, „END”, {{instruction|block}, „RETURN”};

function = intFunction | stringFunction;

intFunction = „FUNCTION”, name, args,
{instruction|block},
RETURN, int;

stringFunction = „FUNCTION”, name, args, „$”,
{instruction|block},
RETURN, string ;

procedure = „SUB”, name, args,
{instruction|block},
RETURN;

block = forBlock|ifBlock;

forBlock = „FOR”, intAssignment „TO”, wh, int,
{block | instruction},
„NEXT”, intId,

ifBlock = „IF”, condition, „THEN”,
{instruction|block},
{„ELSIF”, condition, „THEN”,
{instruction|block}},
[„ELSE”,
{instruction|block}],
„ENDIF”,;

instruction = (intDefinition | stringDeclaration | intAssignment | stringAssignment | arithmetic | len | input | print | read| goto | gosub | intCall | stringCall | procCall | if),
{„:” , (intDefinition | stringDeclaration | intAssignment | stringAssignment | arithmetic | len | input | print | read| goto | gosub | intCall | stringCall | procCall | if)},
nl;

intDefinition = „LET”, wh, intAssignment, {, wh, intAssignment};

stringDeclaration = „DIM”, wh, stringId, ”$”, ”(”, „number”, ”)”;

intAssignment = intId,”=”, (int | intCall | arithmetic);

stringAssignment = stringId, ”=”, (string | stringCall);

arithmetic = term, { ("+" | "-"), term };

term = factor, { ("*"|"/"), factor };

factor = int | intCall | ("(" arithmetic ")");

len = „LEN”, ”(”, string, „)”;

input = „INPUT”, intId, {„ , ”, wh, intId};

print = „PRINT”, printable, {„ ; ”, printable};

printable = string | int | stringCall | intCall;

read = „READ”, stringId, {„ , ”, wh, stringId};

goto = „GOTO”, name;

gosub = „GOSUB”, name;

label = name, „#”;

intCall = name, callArgs;

stringCall = name, callArgs, „$”;

procCall = name, callArgs;

if = „IF”, condition, THEN, {instruction};

condition = cond [{ („AND” | „OR”), cond}]

cond = (condTerm, (">" | "<" | "==" | "<>"), condTerm) | („(”, condition, „)”);

condTerm = atirhmetic | int | intCall;

args = „[”, (intId | stringId), { „ , ”, (intId | stringId)} „]”;

callArgs = „[”, (int | string | intCall | stringCall), { „ , ”, (int | string | intCall | stringCall)} „]”;

string = stringId | substring | „ " ”, alpha, {alpha}, „ " ”;

substring = stringId, „(”, (int | intCall), „)”;

int = inId | number;

stringId = name, „$”

intId = name;

name = alpha, {alpha}

number = digit, {digit};

digit = "0" | "1" | "..." | "9";

alpha = ? znak alfanumeryczny ?

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 wypisuje informuje, na której linii się zatrzymał 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	fffffff

Nazwa	**Typ**	**Argumenty**	**Typ zwracanej wartości**	**Wskaźnik na własną tablicę symboli**
CONC	FUNC	A$,B$,N	STRING	wsk1
MULTIPLY	FUNC	A,B	INTEGER	wsk2
_SUM	PROC	X,Y		wsk3

Komunikacja między modułami

Lekser rozpoznaje tokeny typów:

INTEGER,
STRING,
OPERATOR (operatory arytmetyczne i logiczne),
END_INSTR (nowa linia, „:”),
SEMICOLON,
COLON,
IDENTIFIER (nazwy zmiennych, funkcji, procedur),
LEFT_PARENTHESES
RIGHT_PARENTHESES
RIGHT_BRACKET
LEFT_BRACKET
COMMA
DOLLAR

NUMBER_SIGN

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

Struktura tokenu przekazywana przez lekser parserowi zawiera typ tokenu i jego wartość w przypadku tokenów IDENTIFIER, INTEGER, STRING.

Parser przekazuje do analizatora semantycznego drzewo obiektów reprezentujących: instrukcje odpowiadające użyciu słów kluczowych (np. obiekt klasy instr_LET, zawierający informacje, co i do czego ma być przypisane),
zdefiniowane funkcje (obiekt zawiera informacje o argumentach, typie zwracanej wartości, zawartych instrukcjach),
operatory,
zmienne,
liczby całkowite,
napisy.