INTERPRETER ATARI BASIC

Gabriela Ossowska

Interpreter jest przeznaczony dla języka bazującego na BASICU, z dodaną obsługą instrukcji, blokową instrukcją warunkową i przesłanianiem zmiennych. Jest napisany w Javie z wykorzystaniem ANTLRa do generacji leksera i bazowego wizytatora.

Przykład 1.

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

BEGIN
REM OBLICZANIE ILOCZYNU DWOCH LICZB
LET X=0, Y=0
PRINT "PROGRAM OBLICZA ILOCZYN DWOCH LICZB"
PRINT "PODAJ DWIE DOWOLNE LICZBY"
READ X,Y
LET M = MULTIPLY[X,Y]
? M
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.

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
FOR A=2 TO X
F=F*A
PRINT F
NEXT A
PRINT A
ENDIF
RETURN F

BEGIN
LET X=0
? "PODAJ LICZBE":READ X
LET RESULT = FACTORIAL[X]
? "SILNIA Z ";X;"WYNOSI ";RESULT
END
```

Przykład pętli for: oblicza silnię dla liczb >= 2. Jednocześnie jest definiowana zmienna A, jej zasięg nie wychodzi poza blok.

Przykład instrukcji warunkowej: jeżeli X>=2, wykonaj pętlę.

Przykład 3.

```
FUNCTION$ CONC[N,A$,B$]
DIM C$
C$=A$
C$(LEN(A$)-1)=B$(N)
RETURN C$

BEGIN
DIM A$,B$,RES$
LET N=0
? "PODAJ NAPIS 1 ":READ A$
? "PODAJ NAPIS 2 ":READ B$
? "PODAJ LICZBE ":READ N
RES$ = CONC[N,A$, B$]
PRINT RES$
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$,B$,C$ – zdeklarowanie zmiennych
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

Gramatyka

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

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
‘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 } | ‘(‘ condition ‘)’;

logTerm = artmExpr ;

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

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

intAssignment = id ‘=’ artmExpr;

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

len = ‘LEN’ ‘(‘ stringArg ‘)’;

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

read = ‘READ’ (stringId | id){ ‘,’ (stringId | 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: przy uruchomieniu interpretera należy podać nazwę pliku z kodem jako parametr wywołania. Programy mogą odczytywać wartość podaną przez użytkownika ze standardowego wejścia.
Wyjście i informacje o błędach są wypisawane w konsoli.

Tablice symboli

Zmienne trzymane są w stosie list map, gdzie każda pierwsza lista zawiera zmienne zdefiniowane w głównej części programu, każda kolejna lista obejmuje kontekst funkcji/procedury, każda nowa mapa kontekst bloku. Na funkcje/procedury jest oddzielna lista, zawierająca obiekty, które je reprezentują, a które zawierają nazwę funkcji, nazwy jej argumentów i listę instrukcji.

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, ENDIF, LET, DIM, LEN, , PRINT, PRINT, READ, GOSUB.

Parser przekazuje do analizatora semantycznego drzewo obiektów reprezentujących instrukcje. Obiekt zawiera nazwę instrukcji, listę argumentów, matodę execute.

Testowanie

Działanie interpretera było sprawdzone przy użyciu krótkich programów. Obejmowały sprawdzenie prawidłowości: deklarowania i przypisywania wartości do zmiennych, odczytywania wartości podanych przez użytkownika, drukowania, tworzenia kontekstu i wykonywania pętli, wykonywania instrukcji warunkowych z warunkami logicznymi z uwzględnieniem nawiasowania, tworzenia kontekstu funkcji, obliczania wartości arytmetycznych, manipulacji napisami i fragmentami napisów.

Testy jednostkowe obejmują sprawdzenie, czy lekser prawidłowo rozpoznaje idektyfikatory, czy rozróżnia identyfikator zmiennej napisowej od identyfikatora zmiennej całkowitej, czy omija komentarze i czy prawidłowo rozpoznaje tokeny w nierówności.