TKOM

Tomasz Główka

Dokumentacja wstępna

**Temat**: Język do operacji na listach.

**Opis**

Celem projektu jest stworzenie języka, który skupia się na strukturach listowych i operacjach zwiazanych z nimi.

**Funkcjonalość**

Język udostępnia:

* Definiowanie i wykonywanie funkcji
* Deklarację oraz definicję liczb oraz list
* Instrukcje warunkowe
* Pętle
* Filtrowanie list
* Tworzeni podlist
* Wyświetlanie zawartości list

Uwzględniona jest kolejność operatorów. Komentarze są poprzedzane znakiem „#” Główna część programu musi się zaczynać słowem kluczowym „begin” i kończyć słowem kluczowym „end” (można zauważyć podobieństwo do języka PL/SQL).

**Definicja i deklaracja obiektów**

Możliwymi do zadeklarowania obiektami są: lista z elementami double, string albo zmienna typu double.

Typ danych w listach jest określony przy deklaracji :

* string MyList;
* number MyList

Można również zdefiniować listy:

* string MyList = {‘jeden’, ‘dwa’, ‘trzy’};
* number MyList = {2.4, 5.500, 1000.23};

Zmienne typu double są poprzedzone słowem “var”:

* var number MyVar;

Można również zdefiniować zmienne:

* var number MyVar = 5;

**Operatory**

Operator przypisania „=” pozwala podmienić zawartość listy na zawartość drugiej listy(Uwaga! Gdy definiujemy listę wartościami pamiętać należy o braku spacji pomiędzy kolejnymi elementami i przecinkami. ):

number MyList1 = {1,2}

number MyList2 = {4,5}

MyList1 = MyList2 #MyList1 zawiera teraz element: 4, 5

Operator dodawania „+” sprawia połączenie elementów dodawanych list.

number MyList1 = {1, 2}

number MyList2 = {4, 5}

number MyList3 = MyList1 + MyList2

#MyList3 ma teraz elementy: 1, 2, 4, 5

Operator mnożenia „\*” pozwala przemnożyć ze sobą dwie listy. Polega to na tym, że w wyniku mamy listę, której elementami są odpowiednio przemnożone elementy list(pierwszy = pierwszy element pierwszej listy \* pierwszy element drugiej listy etc. W przypadku gdy mamy string następuje konkatenacja odpowiadających elementów):

number MyList1 = {1,2}

number MyList2 = {2,2}

number MyList3 = MyList1 \* MyList2

#MyList3 ma teraz elementy: 2, 4

Kolejność operatorów jest uwzględniona. Operator mnożenia ma zawsze pierszeństwo przed operatorem dodawania:

number MyList1 = {1, 2}

number MyList2 = {3, 4}

MyList1 = MyList1 + MyList1\*MyList2   
#MyList1 ma teraz elementy: 1, 2, 3, 8

**Pętle**

Język ma w swoim zanadrzu pętle „foreach”, która przechodzi tyle razy ile wskazana lista ma elementów:

number MyList1 = {1, 2}

foreach x in MyList1 loop

show(MyList1)

end loop

#instrukcja show wykona sie dwa razy

**Instrukcje warunkowe**

Język posiada również instrukcje warunktowe “if”, w środku których uzywa się operatorów :

* równy „==”
* większy „>”
* mniejszy „<”
* nierówny „!=”

Przy użyciu tych operatorów do list sprawdzana jest ilość elementów listy:

number MyList1 = {1,2,3}

number MyList2 = {1,2,3}

number MyList3 = {1,2}

if MyList1 == MyList2 and MyList2 != MyList3 then

MyList1 = MyList1 + MyList2

end if

**Funkcje**

Język pozwala na definiowanie własnych funkcji I wywoływaniu ich (dopuszczalna jest rekurencja). Słowo kluczowe „function” zaznacza, że tu zaczyna się funkcja a słowo „return” kończy działanie funkcji. Funkcje definiuje sie na końcu programu. Mogą one tylko operować na zmiennych zdefiniowanych wyżej(globalnych).

begin

number MyList1 = {1,3}

number MyList2 = {1,3,3,3,3,3,3,3,3}

MyFunction()

show (MyList1)

function number MyFunction()

begin

if MyList1 <= MyList2 then

MyList1 = MyList1 + MyList1

MyFunction()

end if

end function

end

**Podlisty**

Mamy możliwość pobrania podlisty podając indeks pierwszego pobranego elementu oraz indeks ostatniego pobranego elementu:

number MyList1 = {1, 3, 5, 6, 7, 9,10}

number MyList2 = MyList1[2,5]

#MyList2 zawiera teraz elementy: 5, 6, 7

**Filtrowanie**

Możemy utworzyć listę poprzez przefiltrowanie innej. W nawiasach kwadratowych podajemy warunek filtrowania „>”, „<”, „==”, „!=” i wartość z którą filtrujemy:

number MyList1 = {1, 3, 5, 6, 7, 9, 10}

number MyList2 = MyList1[>5] #elementy większe od 5

#MyList2 zawiera teraz elementy: 6, 7, 9, 10

**Wyświetlanie**

Do wyświetlania służy nam formuła „show”:

number MyList1 = {1, 3, 5, 6, 7, 9, 10}

show(MyList1)

#otrzymamy na standardowym wyjściu kolejne elementy listy MyList1 #czyli: 1, 3, 5, 6, 7, 9, 10,

**Błędy**

Ewentualne błędy będą prezentowane w konsoli w postaci prostych komunikatów.

Przykładowe komunikaty:

* Error: division by zero !!!
* Undefined variable: varrrr
* Incorrect lower bound: 20
* Lower bound is greater than upper
* Incorrect upper bound: 100
* Function: fun1, does not exists

**Gramatyka**

**Opis gramatyki realizowanego projektu**

* OR = “or”
* THEN = “then”
* AND = “and”
* SHOW = “show”
* LOOP = “loop”
* FOREACH = “foreach”
* IF = “if”
* BEGINN = “begin”
* END = “end”
* IN = “in”
* FUNCTION = “function”
* RETURN = “return”
* VAR = “var”
* STRING = “string”
* NUMBER\_TYPE = “number”
* LEQUAL = ”<=”
* GEQUAL = “>=”
* EQUAL = “==”
* NOTEQUAL = “!=”
* NEWLINE = ‘\n’
* ARG = “arg”
* VARIABLE = [a-zA-Z][a-zA-Z0-9]\*
* COMMENT = #.+
* DOUBLE = [0-9]+(\.[0-9]+)?
* DOUBLE\_LISTDEF = \{[0-9]+(\.[0-9]+)?(,[0-9]+(\.[0-9]+)?)\*\}
* STRING\_LISTDEF = \{(\'.+\')(,(\'.+\'))\*\}
* program: BEGINN NEWLINE stmnt\_list function\_list END NEWLINE
* function\_list: function\_list NEWLINE | function\_list function NEWLINE
* function: FUNCTION NUMBER\_TYPE VARIABLE '(' function\_arguments\_list ')' NEWLINE BEGINN NEWLINE stmnt\_list END FUNCTION | FUNCTION STRING VARIABLE '(' function\_arguments\_list ')' NEWLINE BEGINN NEWLINE stmnt\_list END FUNCTION
* stmnt\_list: stmnt\_list NEWLINE | stmnt\_list statement NEWLINE |
* function\_arguments\_list: function\_arguments\_list | function\_arguments\_list function\_argument |
* function\_argument: ARG VAR NUMBER\_TYPE VARIABLE | ARG NUMBER\_TYPE VARIABLE | ARG STRING VARIABLE
* statement: var\_initialization | var\_declaration | print\_statement| foreach\_statement | if\_statement | return | fun\_execution | comment
* comment: COMMENT
* print\_statement: SHOW '(' VARIABLE ')'
* foreach\_statement: FOREACH VARIABLE IN VARIABLE LOOP NEWLINE stmnt\_list END LOOP
* if\_statement: IF condition THEN NEWLINE stmnt\_list END IF
* return: RETURN expression
* fun\_execution: VARIABLE '('function\_arguments\_list')'
* condition: expression | expression OR condition | expression AND condition
* expression: DOUBLE | STRING\_LISTDEF | DOUBLE\_LISTDEF | VARIABLE | VARIABLE NOTEQUAL VARIABLE | VARIABLE EQUAL VARIABLE | VARIABLE GEQUAL VARIABLE | VARIABLE LEQUAL VARIABLE | VARIABLE ‘<’ VARIABLE | VARIABLE ‘>’ VARIABLE | expression '\*' expression | expression '/' expression | expression '+' expression | expression '-' expression | ‘(‘ expression ‘)’ | VARIABLE '[' VARIABLE ',' DOUBLE ']' | VARIABLE '[' DOUBLE ',' DOUBLE ']' | VARIABLE '[' VARIABLE ',' VARIABLE '] | VARIABLE '[' DOUBLE ',' VARIABLE ']' | VARIABLE '[' NOTEQUAL expression']' | VARIABLE '[' EQUAL expression']' | VARIABLE '[' LEQUAL expression']' | VARIABLE '[' GEQUAL expression']' | VARIABLE '[' '>' expression']' | VARIABLE '[' '<' expression']'
* var\_declaration: VAR NUMBER\_TYPE VARIABLE | NUMBER\_TYPE VARIABLE | STRING VARIABLE
* var\_initialization: VAR NUMBER\_TYPE VARIABLE '=' expression | VARIABLE '=' expression | NUMBER\_TYPE VARIABLE '=' expression | STRING VARIABLE '=' expression

**Założenia Techniczne**

Program będzie dostawał na wejście plik tekstowy „test.txt” z programem napisanym w utworzonym języku.

Poprawność będzie sprawdzana na podstawie standardowego wyjścia, które zostanie przekierowane do pliku „output.txt” i zawartości prezentowanych list po operacjach wykonanych przez program.

Testy zostały zrealizowane za pomocą biblikoteki BOOST i wskazują poprawność generowania logiki programu.

Projekt realizowany będzie w języku c++ z pomocą narzędzi BISON i FLEX

**Uruchomienie programu**

1. Do zbudowania projektu nalezy uzyc kolejno polecen:

-make clean

-make

2. Zostanie utworzony plik wynikowy " uruchamiany w nastepujacy sposob:

./lexer test.txt - to wywoanie spowoduje analize przykladowego programu

3. Program „wypluje” plik „output.txt”, w którym to zostanie umieszczone wykonanie programu

4.Testowanie:

-wejsc do katalogu Test

-odpalic "make clean"

-odpalic "make"