TKOM – Projekt wstępny
Język wspierający operacje na listach

Aleksander Jedynak
/293784/

Opis funkcjonalny

Język będzie umożliwiał łatwe definiowanie list i operowanie na nich. Zapewni sortowanie i filtrowanie stworzonych list.

Elementy:

- Język oferuje dwa typy danych: var oraz list. Typ var może przechowywać liczbę całkowitą lub zmiennoprzecinkową albo ciąg tekstowy w podwójnym cudzysłowiu. Typ list przechowuje zbiór zmiennych typu var.
- Instrukcja warunkowa if else.
- Pętla while.
- Możliwość definiowania funkcji za pomocą słowa function. Funkcja może przyjmować parametry i zwracać wartości dowolnego typu (bez deklaracji wartości zwracanej).
- Operacje wbudowane (przedstawione w tabeli).
- Operatory (przedstawione w tabeli).

Operacja	Argumenty	Przeznaczenie
Print	var zmienna	Wypisanie na ekran wartości argumentu. Brak wartości zwracanej.
PrintList	list lista	
Sort	list lista, var mode	Sortuje listę rosnąco (mode = 0) lub malejąco (mode != 0). W posortowanej liście w pierwszej kolejności znajdą się wartości numeryczne, potem tekstowe. Zwraca posortowaną listę.
Filter	list lista, var od, var do, var ciąg	Filtruje listę na podstawie podanych parametrów. Zwraca elementy listy znajdujące się w zakresie podanym w argumentach lub zawierających podany ciąg. Do pominięcia jednego z parametrów filtrowania należy użyć słowa kluczowego null.
Element	list lista, var indeks	Zwraca element na określonej pozycji.
Erase	list lista, var od, var do	Zwraca listę pozbawioną zbioru elementów określonych parametrami od i do.
Sum	list lista	Zwraca liczbę będącą sumą wszystkich elementów w liście. Ciągi tekstowe są traktowane jako 0.
Sublist	list lista, var od, var do	Zwraca listę zawierającą elementy o indeksach określonych parametrami od i do.
PopFirst	list lista	Usuwa i zwraca pierwszy element.
PopLast	list lista	Usuwa i zwraca ostatni element.
Count	list lista	Zwraca długość listy.

Operatory	Lista <operator> lista	Lista <operator> zmienna i odwrotnie	Zmienna <operator> zmienna
nawiasy	Ustalenie kolejności działań		
***,/**	„*” wykonuje operację iloczynu skalarnego na listach typu liczbowego o jednakowej długości. „/” nie ma tu zastosowania.	„*” – mnożenie wszystkich elementów listy – tylko dla listy i zmiennej liczbowych.	Standardowe działanie dla liczb. Dla ciągów tekstowych dostępny tylko operator „+” (obsługuje konkatenację).
+, -	Złączenie list dla operatora „+”. Operator „-” odejmuje wszystkie elementy drugiej listy od pierwszej.	Dodanie i usuwanie elementu. „-” usuwa wszystkie pasujące elementy.	
<, <=, >=, >	-	-	Standardowe działanie dla zmiennych liczbowych. Zmienne tekstowe porównywane z liczbowymi zawsze mniejsze. Dla dwóch zmiennych tekstowych porównanie alfabetyczne.
==, !=	Równość, gdy elementy takie same i tak samo ułożone.	Równość, gdy lista zawiera jeden element i jest on taki sam jak porównywany z nią.	Równość, gdy liczby lub ciągi tekstowe takie same.
&	Operatory „i” oraz „lub”. Działanie jak w języku C dla && i ||. Listy oraz zmienne tekstowe są interpretowane jako fałsz. Wartość liczbowa 0 to fałsz, inne wartości są prawdziwe. Operatory zwracają 0 (fałsz) lub 1 (prawda).		
|			
=	Przypisanie	-	przypisanie

Przykład 1. Deklaracja i definicja zmiennych

```
var liczba = 10;
var ciag = "abcd";
list lista = {10, 4, 5, 3, 2+3, "a"};
```

Przykład 2. Działania na listach

```
list lista = {10, 4, 5};
lista = lista + 10;
Print(lista);
//Wynik: {10, 4, 5, 10}

lista = lista - 10;
Print(lista);
//Wynik: {4, 5}
```

```
lista = {"A", 10, 4, 5};
Sort(Lista, 0);
Print(lista);
//Wynik: {4, 5, 10, "A"}

list lista2 = {"B"};
lista = lista + lista2;

Print(lista)
//Wynik: {4, 5, 10, "A", "B"}
```

Przykład 3. Konstrukcje językowe

```
function max(list lista)
{
    var tmp = Element(lista, 0);
    var it = 0;
    while(it < Count(lista))
    {
        if(Element(lista, it) > tmp)
        {
            tmp = Element(lista, it);
        }
        it = it + 1;
    }
    return tmp;
}
list l1 = {0,1,2,3,4,5};
Print(max(l1));
//Wynik: 5
list l2 = {"a", "b", 'c'};
Print(max(l2));
//Wynik: c

list l3 = {"a", "b", '5'};
Print(max(l3));
//Wynik: 5
```

Przykład 4. Działanie operatorów

```
list l1 = {1, 2, 3};
list l2 = {4, 5, 6};
Print(l1*l2);
//Wynik: 32
list l3 = {2, 3, 4};
Print(l1-l3);
//Wynik: {1}

Print(l3-2);
```

```
//Wynik: {3, 4}

Print(l3*5);
//Wynik: {10, 15, 20}
```

Opis gramatyki

Symbole terminalne są pogrubione i ujęte w pojedyncze apostrofy.

program	=	{ function | instruction };
func	=	**'function'** name '**(**' params '**)**' '**{**' { instruction } '**}**';
instruction	=	**'if'** '(' expression ')' instruction [else instruction]
		| **'while'** '**(**' expression '**)**' instruction
		| **'return'** [expression] '**;**'
		| [type] name = expression '**;**'
		| [expression] '**;**'
		| type name '**;**'
		| '**{**' { instruction } '**}**';
list_init	=	'**{**' { expression { '**,**' expression } } '**}**';
expression	=	'–' expression
		| '!' expression
		| expression **('*' |'/')** expression
		| expression **('+' |'–')** expression
		| expression **('+' |'–')** expression
		| expression **('<' | '>' | '>=' | '<=')** expression
		| expression **('==' | '!=')** expression
		| expression '**&**' expression
		| expression '**|**' expression
		| (operation|name) ['**(**' [expression { '**;**' expression }] '**)**']
		| '**(**' expression '**)**'
		| number
		| null
		| string;
params	=	[type name] {'**,**' type name};
name	=	letter { letter | digit };
number	=	digit {digit} ['.' digit {digit}] ;
type	=	**'var'** | **'list'** ;
null	=	**'null';**
string	=	'"' {***any ASCII character***} '"' ;
operation	=	**' Print ' | ' PrintList ' | ' Sort ' |' Filter ' | ' Element ' | ' Erase ' | ' Sum ' | ' Sublist ' | ' PopFirst ' | ' PopLast ' | ' Count ' ;**
digit	=	**'a' | …. | 'Z';**
letter	=	**'0' | …. | '9';**

Opis realizacji

- Językiem implementacji będzie C#.
- Obsługiwany będzie zarówno tryb wsadowy, jak i interaktywny.
- Program będzie uruchamiany za pomocą terminala. Na wyjście przekazywany będzie wynik działania operacji Print i PrintList.