Funciones Anónimas

Comandos

Command	::=	single-Command
	|	Command ; single-Command
single-Command	::=	
	|	V-name := Expression
	|	Identifier (Actual-Parameter-Sequence)
	|	**begin** Command **end**
	|	**let** Declaration **in** single-Command
	|	**if** Expression
		then single-Command
		else single-Command
	|	**while** Expression **do** single-Command

(la primera forma de single-Command esta vacía.)

Expresiones

Expression	::=	secondary-Expression
	|	**let** Declaration **in** Expression
	|	**if** Expression **then** Expression
secondary-Expression	::=	primary-Expression
	|	secondary-Expression Operator primary-Expression
primary-Expression	::=	Integer-Literal
	|	Character-Literal
	|	V-name
	|	Identifier (Actual-Parameter-Sequence)
	|	Operator primary-Expression
	|	(Expression)
	|	{ Record-Aggregate }
	|	[Array-Aggregate]
	|	**fun** (Formal-Parameter-Sequence) : Type-denoter ~ Expression
Record-Aggregate	::=	Identifier ~ Expression
	|	Identifier ~ Expression , Record-Aggregate
Array-Aggregate	::=	Expression
	|	Expression , Array-Aggregate

Nombres de variables o Valores

V-name	::=	Identifier
	|	V-name . Identifier
	|	V-name [Expression]

Declaraciones

```
Declaration            ::=    single-Declaration
                       |      Declaration ; single-Declaration
single-Declaration     ::=    const Identifier ~ Expression
                       |      var Identifier : Type-denoter
                       |      proc Identifier ( Formal-Parameter-Sequece) ~
                                  single-Command
                       |      func Identifier ( Formal-Parameter-Sequence )
                                  : Type-denoter ~Expression
                       |      type Identifier ~ Type-denoter
```

```
let
  var i : integer,
  var f : fun(integer) : integer
in
  f := fun(n : integer) : integer ~ n + i
```

Parámetros

```
Formal-Parameter-Sequence
                       ::=
                       |      proper-Formal-Parameter-Sequence

proper-Formal-Parameter-Sequence
                       ::=    Formal-Parameter
                       ::=    Formal-Parameter , proper-Formal-Parameter-Sequence

Formal-Parameter       ::=    Identifier : Type-denoter
                       |      var Identifier : Type-denoter
                       |      proc Identifier ( Formal-Parameter-Sequence )
                       |      func Identifier ( Formal-Parameter-Sequence )
                                  :  Type-denoter

Actual-Parameter-Sequence
                       ::=
                       |      proper-Actual-Parameter-Sequence

proper-Actual-Parameter-Sequence
                       ::=    Actual-Parameter
                       |      Actual-Parameter, proper-Actual-Parameter-Sequence

Actual-Parameter       ::=    Expression
                       |      var V-name
                       |      proc Identifier
                       |      func Identifier
         NO HACE FALTA |  fun (Formal-Parameter-Sequence ) : Type-denoter ~ Expression
```

Denotadores de Tipos

Type-denoter ::= Identififer
| **array** Integer-Literal **of** Type-Denoter
| **record** Record-Type-denoter **end**
| **fun** (Formal-Parameter-Sequence (sin nombres de vars)) :
Type-denoter

Record-Type-denoter ::= Identifier : Type-denoter
Identifier :T ype-denoter , Record-Type-denoter

Sintaxis

Program ::= (Token | Comment | Blank)*

Token ::= Integer-Literal | Character-Literal | Indetifier | Operator |
array | **begin** | **const** | **do** | **else** | **end** | **func** | **if** | **in** |
let | **of** | **proc** | **record** | **then** | **type** | **while** |
. | : | ; | , | := | ~ | (|) | [|] | { | }

Integer-Literal ::= Digit Digit*

Character-Literal ::= 'Graphic'

Identifier ::= Letter (Letter | Digit)*

Operator ::= Op-character Op-character*

Comment ::= **!** Graphic*

Funciones Anónimas

1. poder declarar variables de tipo fun
2. Poder mandar expresiones fun a en los parámetros formales de un procedimiento
3. El tipo fun y func debe ser compatible.

TAM

```
let

   func fib(func f(integer) : integer),
   var v : array 10 of fun(integer) : integer


in
    for i := 1 to 10 do
        v[i] = fun(n : integer) : integer ~ n+i
    fib(fun(n : integer) : integer ~ n+1)
```