Program ::= (Package-Declaration **;**)\* Command

Command ::= single-Command (; single-Command)\*

single-Command ::=V-name **::=** Expression

ILong-Identifier **(** Actual-Parameter-Sequence **)**

| **pass**

| **loop** Loop-Cases

| **let** Declaration **in** Command **end**

| **if** Expression **then** Command **else** Command **end**

| **choose** Expression **from** Cases **end**

Loop-Cases ::=  **while** Expression **do** Command **end**

| **until** Expression **do** Command **end**

| **do** Command Do-loop

| **for** Identifier **from** Expression **to** Expression For-loop

Do-loop ::=  **while** Expression **end**

| **until** Expression **end**

For-loop ::=  **do** Command **end**

| **while** Expression **do** Command **end**

| **until** Expression **do** Command **end**

Cases ::= (Case)+ [ ElseCase ]

Case ::= **when** Case-Literals **then** Command

ElseCase ::= **else** Command

Case-Literals ::= Case-Range ( **|** Case-Range)\*

Case-Range ::= Case-Literal [ **..** Case-Literal]

Case-Literal ::= Integer-Literal

| Character-Literal

Expression ::= secondary-Expression

I **let** Declaration **in** Expression

I **if** Expression **then** Expression **else** Expression

secondary-Expression ::= primary-Expression (Operator primary-Expression)\*

primary-Expression ::= Integer-Literal

I Character-Literal

I V-name

I Long-ldentifier **(** Actual-Parameter-Sequence **)**

I Operator primary-Expression

I **(** Expression **)**

I **{** Record-Aggregate **}**

I  **[** Array-Aggregate **]**

Record-Aggregate ::= Identifier **~** Expression [**,** Record-Aggregate]

Array-Aggregate ::= Expression [**,** Array-Aggregate]

V-name ::= [Package-Identifier **$**] Var-Name

Var-name ::= Identifier (Selector)\*

Selector ::= **.**Identifier

| **[** Expression **]**

Declaration ::= compound-Declaration (**;** compound-Declaration)\*

compound-Declaration ::= single-Declaration

| **recursive** Proc-Funcs **end**

| **private** Declaration **in** Declaration **end**

| **par** single-Declaration (**|** single-Declaration)⁺ **end**

Proc-Func ::= **proc** Identifier **(** Formal-Parameter-Sequence **)**

**~** Command **end**

| **func** Identifier **(** Formal-Parameter-Sequence **)**

**:** Type-denoter **~** Expression

Proc-Funcs

::= Proc-Func (**|** Proc-Func)+

single-Declaration ::= **const** Identifier **~** Expression

| **var** Identifier Var-Single-Declaration

| **proc** Identifier **(** Formal-Parameter-Sequence **)**

**~** Command **end**

| **func** Identifier **(** Formal-Parameter-Sequence **)**

**:** Type-denoter **~** Expression

| **type** Identifier **~** Type-denoter

Var-Single-Declaration ::= **:** Type-denoter

| **::=** Expression

Formal-Parameter-Sequence ::= [proper-Formal-Parameter-Sequence]

proper-Formal-Parameter-Sequence ::= 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 [, proper-Actual-Parameter-Sequence]

Actual-Parameter ::= Expression

| **var** V-name

| **proc** Identifier

| **func** Identifier

Type-denoter ::= Long-Identifier

I **array** Integer-Literal **of** Type-denoter

I **record** Record-Type-denoter **end**

Record-Type-denoter ::= ldentifier **:** Type-denoter[ **,** Record-Type-denoter]

Package-Declaration ::= **package** Package-Identifier **~**

Declaration **end**

**Microsintaxis**

Package-Identifier ::= Identifier

Long-Identifier ::= [Package-Identifier **$**] Identifier

Token ::= Integer-Literal | Character-Literal | Identifier | Operator | **array** | **choose** | **const** | **do** | **else** | **end** | **for** | **from** | **func** | **if** | **in** | **let** | **loop** | **of** | **package** | **par** | **pass** | **private** | **proc** | **record** | **recursive** | **then** | **to** | **type** | **until** | **var** | **when** | **while** | **.** | **:** | **;** | **,** | **:=** | **~** | **(** | **)** | **[** | **]** | **{** | **}** | **|** | **::=** | **$** | **. .**

Integer-Literal ::= Digit Digit\*

Character-Literal ::= **‘** Graphic **‘**

Identifier ::= Letter ( Letter | Digit )\*

Operator ::= Op-character Op-character\*

Comment ::= **!** Graphic\* end-of-line

Blank ::= space | tab | end-of-line

Graphic ::= Letter | Digit | Op-character | space | tab| **.** | **:** | **;** | **,** | **~** | **(** | **)** | **[** | **]** | **{** | **}** | **!** | **`** | **”** | **#** | **$**

Letter ::= **a | b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** | **j** | **k** | **l** | **m** | **n** | **o** | **p** | **q** | **r** | **s** | **t** | **u** | **v** | **w** | **x** | **y** | **z | A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** | **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **X** | **Y** | **Z**

Digit ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Op-character ::= **+** | **-** | **\*** | **/** | **=** | **<** | **>** | **\** | **&** | **@** | **%** | **^** | **?**

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