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

single-Command ::=

I V-name ::= Expression

I Identifier ( Actual-Parameter-Sequence )

I **begin** Command **end**

I **let** Declaration **in** single-Command

I **if** Expression **then** single-Command

**else** single-Command

I **while** Expression **do** single-Command

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 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 ::= Identifier

| V-name V-nameAux

V-nameAux ::= . Identifier

| [ Expression ]

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

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

| **var** Identifier : Type-denoter

| **proc** Identifier ( Formal-Parameter-Sequence ) ~ single-Command

| **func** Identifier ( Formal-Parameter-Sequence ) : Type-denoter ~ Expression

| **type** Identifier ~ Type-denoter

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 ::= Identifier

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

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

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

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

Token ::= Integer-Literal | Character-Literal | Identifier | Operator | **array** | **begin** | **const** | **do** | **else** | **end** | **func** | **if** | **in** | **let** | **of** | **proc** | **record** | **then** | **type** | **var** | **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

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

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

Program ::= Command