

# Algol 60 grammar (tncy)

```
<empty>  
::=
```

## 1. Basic Symbols

```
<basic symbol>  
::= <letter>  
    <digit>  
    <logical value>  
    <delimiter>
```

### 1.1. Letters

```
<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

### 1.1.1. Digits

<i>&lt;digit&gt;</i>	
::=	0
	1
	2
	3
	4
	5
	6
	7
	8
	9

### 1.1.2. Logical values

<i>&lt;logical value&gt;</i>	
::=	true
	false

## 1.2. Delimiters

<i>&lt;delimiter&gt;</i>	
::=	<i>&lt;operator&gt;</i>
	<i>&lt;separator&gt;</i>
	<i>&lt;bracket&gt;</i>
	<i>&lt;declarator&gt;</i>
	<i>&lt;specificator&gt;</i>

<i>&lt;operator&gt;</i>	
::=	<i>&lt;arithmetic operator&gt;</i>
	<i>&lt;relational operator&gt;</i>
	<i>&lt;logical operator&gt;</i>
	<i>&lt;sequential operator&gt;</i>

<i>&lt;arithmetic operator&gt;</i>	
::=	+
	-
	x
	/
	÷

**<relational operator>**  
↑  
::= <  
      ≤  
      =  
      ≥  
      >  
      ≠

**<logical operator>**  
::= ≡  
      ⊃  
      ∨  
      ∧  
      ¬

**<sequential operator>**  
::= go to  
      if  
      then  
      else  
      for  
      do

**<separator>**  
::= ,  
      .  
      <sup>10</sup>  
      :  
      ;  
      :=  
      ⊥  
      step  
      until  
      while  
      comment

**<bracket>**  
::= (  
      )  
      [  
      ]  
      '  
      '  
      begin  
      end

**<declarator>**  
::= own  
      Boolean  
      integer  
      real  
      array  
      switch  
      procedure

**<specifier>**  
::= string  
      label

	value	
The sequence		is equivalent to
; comment <i>&lt;any sequence of zero or more characters not containing ;&gt;</i> ;		;
begin comment <i>&lt;any sequence of zero or more characters not containing ;&gt;</i> ;		begin

### 1.3. Identifiers

```

<identifier>
::=    <letter>
|      <identifier> <letter>
|      <identifier> <digit>

```

### 1.4. Numbers

```

<unsigned integer>
::=    <digit>
|      <unsigned integer> <digit>

<integer>
::=    <unsigned integer>
|      + <unsigned integer>
|      - <unsigned integer>

<decimal fraction>
::=    . <unsigned integer>

<exponential part>
::=    10 <integer>

<decimal number>
::=    <unsigned integer>
|      <decimal fraction>
|      <unsigned integer> <decimal fraction>

<unsigned number>
::=    <decimal number>
|      <decimal number> <exponential part>

<number>
::=    <unsigned number>
|      + <unsigned number>
|      - <unsigned number>

```

### 1.5. Strings

```

<proper string>
::=    <any sequence of characters not containing ' or '\>
|      <empty>

```

```

<open string>
::=    <proper string>
|      <proper string> <closed string> <open string>

<closed string>
::=    ' <open string> '

<string>
::=    <closed string>
|      <closed string> <string>

```

## 2. Expressions

```

<expression>
::=    <arithmetic expression>
|      <Boolean expression>
|      <designational expression>

```

### 2.1. Variables

```

<variable identifier>
::=    <identifier>

<simple variable>
::=    <variable identifier>

<subscript expression>
::=    <arithmetic expression>

<subscript list>
::=    <subscript expression>
|      <subscript list> , <subscript expression>

<array identifier>
::=    <identifier>

<subscripted variable>
::=    <array identifier> [ <subscript list> ]

<variable>
::=    <simple variable>
|      <subscripted variable>

```

### 2.2. Function designators

```

<procedure identifier>
::=    <identifier>

<actual parameter>
::=    <string>
|      <expression>
|      <array identifier>
|      <switch identifier>
|      <procedure identifier>

```

```

<letter string>
::=    <letter>
|      <letter string> <letter>

<parameter delimiter>
::=    ,
|      ) <letter string> : (

<actual parameter list>
::=    <actual parameter>
|      <actual parameter list> <parameter delimiter> <actual parameter>

<actual parameter part>
::=    <empty>
|      ( <actual parameter list> )

<function designator>
::=    <procedure identifier> <actual parameter part>

```

## 2.3. Arithmetic expressions

```

<adding operator>
::=    +
|      -

<multiplying operator>
::=    ×
|      /
|      ÷

<primary>
::=    <unsigned number>
|      <variable>
|      <function designator>
|      ( <arithmetic expression> )

<factor>
::=    <primary>
|      <factor> ↑ <primary>

<term>
::=    <factor>
|      <term> <multiplying operator> <factor>

<simple arithmetic expression>
::=    <term>
|      <adding operator> <term>
|      <simple arithmetic expression> <adding operator> <term>

<if clause>
::=    if <Boolean expression> then

<arithmetic expression>
::=    <simple arithmetic expression>
|      <if clause> <simple arithmetic expression> else <arithmetic
expression>

```

## 2.4. Boolean expressions

*<relational operator>*

*::=*  
|  
|  
|  
|  
|  
|

*<*  
*≡*  
*=*  
*≠*  
*>*  
*≠*

*<relation>*

*::=*     *<simple arithmetic expression>* *<relational operator>* *<simple arithmetic expression>*

*<Boolean primary>*

*::=*     *<logical value>*  
|        *<variable>*  
|        *<function designator>*  
|        *<relation>*  
|        ( *<Boolean expression>* )

*<Boolean secondary>*

*::=*     *<Boolean primary>*  
|        *¬* *<Boolean primary>*

*<Boolean factor>*

*::=*     *<Boolean secondary>*  
|        *<Boolean factor>* *∧* *<Boolean secondary>*

*<Boolean term>*

*::=*     *<Boolean factor>*  
|        *<Boolean term>* *∨* *<Boolean factor>*

*<implication>*

*::=*     *<Boolean term>*  
|        *<implication>* *⊃* *<Boolean term>*

*<simple Boolean>*

*::=*     *<implication>*  
|        *<simple Boolean>* *≡* *<implication>*

*<Boolean expression>*

*::=*     *<simple Boolean>*  
|        *<if clause>* *<simple Boolean>* **else** *<Boolean expression>*

## 2.5. Designational expressions

*<label>*

*::=*     *<identifier>*  
|        *<unsigned integer>*

*<switch identifier>*

*::=*     *<identifier>*

*<switch designator>*

*::=*     *<switch identifier>* [ *<subscript expression>* ]

```

<simple designational expression>
::=    <label>
      |    <switch designator>
      |    ( <designational expression> )

<designational expression>
::=    <simple designational expression>
      |    <if clause> <simple designational expression> else <designational
          expression>

```

## 3. Statements

### 3.1. Compound statements and blocks

```

<unlabelled basic statement>
::=    <assignment statement>
      |    <go to statement>
      |    <dummy statement>
      |    <procedure statement>

<basic statement>
::=    <unlabelled basic statement>
      |    <label> : <basic statement>

<unconditional statement>
::=    <basic statement>
      |    <compound statement>
      |    <block>

<statement>
::=    <unconditional statement>
      |    <conditional statement>
      |    <for statement>

<compound tail>
::=    <statement> end
      |    <statement> ; <compound tail>

<block head>
::=    begin <declaration>
      |    <block head> ; <declaration>

<unlabelled block>
::=    <block head> ; <compound tail>

<unlabelled compound>
::=    begin <compound tail>

<compound statement>
::=    <unlabelled compound>
      |    <label> : <compound statement>

<block>
::=    <unlabelled block>
      |    <label> : <block>

<program>

```



```
 ::= <block>
 | <compound statement>
```

## 3.2. Assignment statements

```
<destination>
 ::= <variable>
 | <procedure identifier>

<left part>
 ::= <destination> :=

<left part list>
 ::= <left part>
 | <left part list> <left part>

<assignment statement>
 ::= <left part list> <arithmetic expression>
 | <left part list> <Boolean expression>
```

## 3.3. Go to statements

```
<go to statement>
 ::= go to <designational expression>
```

## 3.4. Dummy statements

```
<dummy statement>
 ::= <empty>
```

## 3.5. Conditional statements

```
<if clause>
 ::= if <Boolean expression> then

<unconditional statement>
 ::= <basic statement>
 | <compound statement>
 | <block>

<if statement>
 ::= <if clause> <unconditional statement>

<conditional statement>
 ::= <if statement>
 | <if statement> else <statement>
 | <if clause> <for statement>
 | <label> : <conditional statement>
```

## 3.6. For statements

```
<for list element>
```

```

::=      <arithmetic expression>
|        <arithmetic expression> step <arithmetic expression> until
|        <arithmetic expression>
|        <arithmetic expression> while <Boolean expression>

<for list>
::=      <for list element>
|        <for list> , <for list element>

<for clause>
::=      for <variable> := <for list> do

<for statement>
::=      <for clause> <statement>
|        <label> : <for statement>

```

### 3.7. Procedure statements

```

<actual parameter>
::=      <string>
|        <expression>
|        <array identifier>
|        <switch identifier>
|        <procedure identifier>

<letter string>
::=      <letter>
|        <letter string> <letter>

<parameter delimiter>
::=      ,
|        ) <letter string> : (

<actual parameter list>
::=      <actual parameter>
|        <actual parameter list> <parameter delimiter> <actual parameter>

<actual parameter part>
::=      <empty>
|        ( <actual parameter list> )

<procedure statement>
::=      <procedure identifier> <actual parameter part>

```

## 4. Declarations

```

<declaration>
::=      <type declaration>
|        <array declaration>
|        <switch declaration>
|        <procedure declaration>

```

### 4.1. Type declarations

```

<type list>

```

```

::=      <simple variable>
|        <simple variable> , <type list>

<type>
::=      real
|        integer
|        Boolean

<local or own>
::=      <empty>
|        own

<type declaration>
::=      <local or own> <type> <type list>

```

## 4.2. Array declarations

```

<lower bound>
::=      <arithmetic expression>

<upper bound>
::=      <arithmetic expression>

<bound pair>
::=      <lower bound> : <upper bound>

<bound pair list>
::=      <bound pair>
|        <bound pair list> , <bound pair>

<array segment>
::=      <array identifier> [ <bound pair list> ]
|        <array identifier> , <array segment>

<array list>
::=      <array segment>
|        <array list> , <array segment>

<array declarer>
::=      <type> array
|        array

<array declaration>
::=      <local or own> <array declarer> <array list>

```

## 4.3. Switch declarations

```

<switch list>
::=      <designational expression>
|        <switch list> , <designational expression>

<switch declaration>
::=      switch <switch identifier> := <switch list>

```

## 4.4. Procedure declarations

```

<formal parameter>
::=    <identifier>

<formal parameter list>
::=    <formal parameter>
|      <formal parameter list> <parameter delimiter> <formal parameter>

<formal parameter part>
::=    <empty>
|      ( <formal parameter list> )

<identifier list>
::=    <identifier>
|      <identifier list> , <identifier>

<value part>
::=    value <identifier list> ;
|      <empty>

<specifier>
::=    string
|      <type>
|      <array declarer>
|      label
|      switch
|      procedure
|      <type> procedure

<specification part>
::=    <empty>
|      <specifier> <identifier list> ;
|      <specification part> <specifier> <identifier list> ;

<procedure heading>
::=    <procedure identifier> <formal parameter part> ; <value part>
|      <specification part>

<procedure body>
::=    <statement>

<procedure declaration>
::=    procedure <procedure heading> <procedure body>
|      <type> procedure <procedure heading> <procedure body>

```

## 5. Transcription of basic symbols

The UTF-8 symbol	is replaced by the ASCII symbol
(	(
)	)
[	[
]	]
,	,

The UTF-8 symbol	is replaced by the ASCII symbol
;	;
:	:
:=	:=
≡	⟷
⊃	⇒
∨	∖
∧	/\
¬	~
=	=
≠	⟷
∧	<
≧	>=
>	>
≦	<=
+	+
-	-
x	*
/	/
÷	//
↑	**
┐	`
┐	'
.	.
10	e
⌞	