CA4003 Compiler Construction Assignment Language Definition

David Sinclair

2017-2018

1 Overview

The language is not case sensitive. A nonterminal, X, is represented by enclosing it in angle brackets, e.g. $\langle X \rangle$. A terminal is represented without angle brackets. A **bold typeface** is used to represent terminal symbols in the language and reserved words, whereas a non-bold typeface is used for symbols that are used to group terminals and nonterminals together. Source code is kept in files with the .ccl extension, e.g. hello_world.ccl .

2 Syntax

The reserved words in the language are **var**, **const**, **return**, **integer**, **boolean**, **void**, **main**, **if**, **else**, **true**, **false**, **while**, **begin**, **end**, **is** and **skip**.

The following are tokens in the language: **, ; : = ( ) + - ~ || && == != < <= > >=**

Integers are represented by a string of one or more digits ('0'-'9') and may start with a minus sign ('-'), e.g. 123, -456. Unless it is the number '0', numbers may not start with leading '0's. For example, 0012 is illegal.

Identifiers are represented by a string of letters, digits or underscore character ('_') beginning with a letter. Identifiers cannot be reserved words.

Comments can appear between any two tokens. There are two forms of comment: one is delimited by /* and */ and can be nested; the other begins with // and is delimited by the end of line and this type of comments may not be nested.

$$\langle \text{program} \rangle \models \langle \text{decl\_list} \rangle \ \langle \text{function\_list} \rangle \ \langle \text{main} \rangle \tag{1}$$

$$\langle \text{decl\_list} \rangle \models (\langle \text{decl} \rangle \ ; \ \langle \text{decl\_list} \rangle \ | \ \epsilon) \tag{2}$$

$$\langle \text{decl} \rangle \models \langle \text{var\_decl} \rangle \ | \ \langle \text{const\_decl} \rangle \tag{3}$$

$$\langle \text{var\_decl} \rangle \models \textbf{var} \ \text{identifier:}\langle \text{type} \rangle \tag{4}$$

$$\langle \text{const\_decl} \rangle \models \textbf{const} \ \text{identifier:}\langle \text{type} \rangle = \langle \text{expression} \rangle \tag{5}$$

$$\langle \text{function\_list} \rangle \models (\langle \text{function} \rangle \ \langle \text{function\_list} \rangle \ | \ \epsilon) \tag{6}$$

$$\langle \text{function} \rangle \models \langle \text{type} \rangle \ \text{identifier} \ (\langle \text{parameter\_list} \rangle) \ \textbf{is} \tag{7}$$
$$\langle \text{decl\_list} \rangle$$
$$\textbf{begin}$$
$$\langle \text{statement\_bock} \rangle$$
$$\textbf{return} \ (\ \langle \text{expression} \rangle \ | \ \epsilon \ ) \ ;$$
$$\textbf{end}$$

$$\langle \text{type} \rangle \models \textbf{integer} \ | \ \textbf{boolean} \ | \ \textbf{void} \tag{8}$$

$$\langle \text{parameter\_list} \rangle \models \langle \text{nemp\_parameter\_list} \rangle \ | \ \epsilon \tag{9}$$

$$\langle \text{nemp\_parameter\_list} \rangle \models \text{identifier:}\langle \text{type} \rangle \ | \ \text{identifier:}\langle \text{type} \rangle \ , \ \langle \text{nemp\_parameter\_list} \rangle$$

$$\langle \text{main} \rangle \models \textbf{main} \tag{10}$$
$$\textbf{begin}$$
$$\langle \text{decl\_list} \rangle$$
$$\langle \text{statement\_block} \rangle$$
$$\textbf{end}$$

$$\langle \text{statement\_block} \rangle \models (\langle \text{statement} \rangle \ \langle \text{statement\_block} \rangle) \ | \ \epsilon \tag{11}$$

$$\langle \text{statement} \rangle \models \text{identifier} = \langle \text{expression} \rangle \ ; \ | \tag{12}$$
$$\text{identifier} \ (\ \langle \text{arg\_list} \rangle \ ) \ ; \ |$$
$$\textbf{begin} \ \langle \text{statement\_block} \rangle \ \textbf{end} \ |$$
$$\textbf{if} \ \langle \text{condition} \rangle \ \textbf{begin} \ \langle \text{statement\_block} \rangle \ \textbf{end}$$
$$\textbf{else begin} \ \langle \text{statement\_block} \rangle \ \textbf{end} \ |$$
$$\textbf{while} \ \langle \text{condition} \rangle \ \textbf{begin} \ \langle \text{statement\_block} \rangle \ \textbf{end} \ |$$
$$\textbf{skip} \ ;$$

$$\langle \text{expression} \rangle \models \langle \text{fragment} \rangle \ \langle \text{binary\_arith\_op} \rangle \ \langle \text{fragment} \rangle \ | \tag{13}$$
$$(\ \langle \text{expression} \rangle \ ) \ |$$
$$\text{identifier} \ (\langle \text{arg\_list} \rangle \ ) \ |$$

$$\langle\text{fragment}\rangle$$

$$
\begin{aligned}
\langle\text{binary\_arith\_op}\rangle &\models\ \textbf{+} \mid \textbf{-} & (14)\\
\langle\text{fragment}\rangle &\models\ \text{identifier} \mid \textbf{-}\ \text{identifier} \mid \text{number} \mid \textbf{true} \mid \textbf{false} \mid & (15)\\
&\quad\ \langle\text{expression}\rangle \\
\langle\text{condition}\rangle &\models\ \sim \langle\text{condition}\rangle \mid & (16)\\
&\quad\ \textbf{(}\ \langle\text{condition}\rangle\ \textbf{)}\ \mid \\
&\quad\ \langle\text{expression}\rangle\ \langle\text{comp\_op}\rangle\ \langle\text{expression}\rangle\ \mid \\
&\quad\ \langle\text{condition}\rangle(\ \textbf{||}\ \mid\ \textbf{\&\&}\ )\langle\text{condition}\rangle \\
\langle\text{comp\_op}\rangle &\models\ \textbf{==} \mid \textbf{!=} \mid \textbf{<} \mid \textbf{<=} \mid \textbf{>} \mid \textbf{>=} & (17)\\[2ex]
\langle\text{arg\_list}\rangle &\models\ \langle\text{nemp\_arg\_list}\rangle \mid \epsilon & (18)\\
\langle\text{nemp\_arg\_list}\rangle &\models\ \text{identifier} \mid \text{identifier}\ \textbf{,}\ \langle\text{nemp\_arg\_list}\rangle & (19)
\end{aligned}
$$

3 Semantics

Declaration made outside a function (including main) are global in scope. Declarations inside a function are local in scope to that function. Function arguments are *passed-by-value*. Variables or constants cannot be declared using the void type. The skip statement does nothing.

The operators in the language are:

Operator	Arity	Description
=	binary	assignment
+	binary	arithmetic addition
-	binary	arithmetic subtraction
-	unary	arithmetic negation
$\sim$	unary	logical negation
||	binary	logical disjunction (logical or)
&&	binary	logical conjunction (logical and)
==	binary	is equal to (arithmetic and logical)
!=	binary	is not equal to (arithmetic and logical)
<	binary	is less than (arithmetic)
<=	binary	is less than or equal to (arithmetic)
>	binary	is greater than (arithmetic)
>=	binary	is greater than or equal to (arithmetic)

The following table gives the precedence (from highest to lowest) and associativity of these operators.

3

Operator(s)	Associativity	Notes
~	right to left	logical negation
-	right to left	arithmetic negation
+ -	left to right	addition & subtraction
< <= > >=	left to right	arithmetic comparison operators
== !=	left to right	equality & inequality operators
&&	left to right	logical conjunction
||	left to right	logical disjunction
=	right to left	assignment

4 Examples

Three versions of the simplest non-empty file demonstrating that the language is case insensitive.

```
main              Main              MAIN
begin             begin             begin
end               end               end
```

A simple file demonstrating comments.

```
main
begin
  // a simple comment
  /* a comment /* with /* several */ nested */ comments */
end
```

The simplest program that uses functions.

```
void func () is
begin
  return ();
end
```

```
main
begin
  func ();
end
```

A simple file demonstrating the different scopes.

```
var i:integer;

integer test_fn (x:integer) is
  var i:integer;
begin
  i = 2;
  return (x);
end

main
begin
  var i:integer;

  i = 1;
  i = test_fn (i);
end
```

A file demonstrating the use of functions.

```
integer multiply (x:integer, y:integer) is
  var result:integer;
  var minus_sign : boolean;
begin
  // figure out sign of result and convert args to absolute values

  if (x < 0 && y >= 0)
  begin
    minus_sign = true;
    x = -x;
  end
  else
  begin
```

```
    if  y  <  0  &&  x  >=  0
    begin
      minus_sign  =  true ;
      y  =  −y ;
    end
    else
    begin
      if  (x  <  0)  &&  y  <  0
      begin
        minus_sign  =  false ;
        x  =  −x ;
        y  =  −y ;
      end
      else
      begin
        minus_sign  =  false ;
      end
    end
  end

  result  =  0;

  while  (y  >  0)
  begin
    result  =  result  +  x ;
    y  =  y  −  1;
  end

  if  minus_sign  ==  true
  begin
    result  =  −result ;
  end
  else
  begin
    skip ;
  end

    return  ( result );
end
```

```
main
begin
  var arg_1:integer;
  var arg_2:integer;
  var result:integer;
  const five:integer = 5;

  arg_1 = -6;
  arg_2 = five;

  result = multiply (arg_1, arg_2);
end
```