CA4003 Compiler Construction
Assignment
Language Definition

David Sinclair

2022-2023

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 should be kept in files with the .cal extension, e.g. hello_world.cal .

2 Syntax

The reserved words in the language are **variable**, **constant**, **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.

⟨program⟩ ⊨ ⟨decl_list⟩ ⟨function_list⟩ ⟨main⟩ (1)

⟨decl_list⟩ ⊨ (⟨decl⟩ **;** ⟨decl_list⟩ | ϵ) (2)

⟨decl⟩ ⊨ ⟨var_decl⟩ | ⟨const_decl⟩ (3)

⟨var_decl⟩ ⊨ **variable** `identifier:`⟨type⟩ (4)

⟨const_decl⟩ ⊨ **constant** `identifier:`⟨type⟩ **:=** ⟨expression⟩ (5)

⟨function_list⟩ ⊨ (⟨function⟩ ⟨function_list⟩ | ϵ) (6)

⟨function⟩ ⊨ ⟨type⟩ `identifier` **(**⟨parameter_list⟩**) is** (7)
⟨decl_list⟩
begin
⟨statement_bock⟩
return (⟨expression⟩ | ϵ **) ;**
end

⟨type⟩ ⊨ **integer** | **boolean** | **void** (8)

⟨parameter_list⟩ ⊨ ⟨nemp_parameter_list⟩ | ϵ (9)

⟨nemp_parameter_list⟩ ⊨ `identifier:`⟨type⟩ | `identifier:`⟨type⟩ **,** ⟨nemp_parameter_list⟩

⟨main⟩ ⊨ **main** (10)
begin
⟨decl_list⟩
⟨statement_block⟩
end

⟨statement_block⟩ ⊨ (⟨statement⟩ ⟨statement_block⟩) | ϵ (11)

⟨statement⟩ ⊨ `identifier` **:=** ⟨expression⟩ **;** | (12)
`identifier` **(** ⟨arg_list⟩ **) ;** |
begin ⟨statement_block⟩ **end** |
if ⟨condition⟩ **begin** ⟨statement_block⟩ **end**
else begin ⟨statement_block⟩ **end** |
while ⟨condition⟩ **begin** ⟨statement_block⟩ **end** |
skip ;

⟨expression⟩ ⊨ ⟨fragment⟩ ⟨binary_arith_op⟩ ⟨fragment⟩ | (13)
(⟨expression⟩ **)** |
`identifier` **(**⟨arg_list⟩ **)** |

⟨fragment⟩

⟨binary_arith_op⟩ ⊨ + | - (14)

⟨fragment⟩ ⊨ `identifier` | - `identifier` | `number` | **true** | **false** | (15)
⟨expression⟩

⟨condition⟩ ⊨ ∼ ⟨condition⟩ | (16)
(⟨condition⟩ **)** |
⟨expression⟩ ⟨comp_op⟩ ⟨expression⟩ |
⟨condition⟩(| | **&**)⟨condition⟩

⟨comp_op⟩ ⊨ = | **!=** | < | <= | > | >= (17)

⟨arg_list⟩ ⊨ ⟨nemp_arg_list⟩ | ϵ (18)

⟨nemp_arg_list⟩ ⊨ `identifier` | `identifier` **,** ⟨nemp_arg_list⟩ (19)

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
∼	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.

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.

```
variable i:integer;

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

main
begin
  variable i:integer;

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

A file demonstrating the use of functions.

```
integer multiply (x:integer, y:integer) is
  variable result:integer;
  variable 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
  variable arg_1:integer;
  variable arg_2:integer;
  variable result:integer;
  constant five:integer := 5;

  arg_1 := -6;
  arg_2 := five;

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