Waseda University

Computer Language Processors' Compiler "tlc" language specification

Keiji Kimura

March, 2016 (Japanese Ver.)

June, 2020 (English Ver.)

1. Introduction

This document describes the language specification of the Computer Language Processors' compiler "tlc". tlc takes a source code written in "tl" (tiny language) whose language specification is a subset of the specification of C language. Its specification is decided to make its compiler's implementation simple as much as possible. This documents referrers the C language specification document [1].

2. Overview of Language

As explained above, tl is a subset of C language. It has the following constrains comparing with C language:

- The type of a variable is only "int".
- Constant type is only integer constant.
- Available operations in expressions are only +,-,*,/,<,<=,>=,>,==,and !=.
- Available statements are only if, while, do-while, for, and return.

In the following syntax definitions, the specification of C language [1] is basically applicable if there is no annotation.

2.1. Variables

For variables, only function local variables can be used.

2.1.1. Scope of Variables

A variable is allocated in a memory when the function where the variable is declared starts its execution. It is kept by the end of the function.

2.1.2. Type of Variables

Only **`int`** can be used as a type of a variable.

2.2. Tokens

2.2.1. Keywords

The keywords of tl are as the following:

```
do, else, for, if, int, main, return, while
```

2.2.2. Identifiers

An identifier start from Beginning from "a-z" or "A-Z", and a series of "a-z", "A-Z", "0-9", or "_" more than 0. It can be used as a variable name and a function name.

2.2.3. Constants

Only integer constants are available.

2.3. Expressions

Syntax:

expression:

assignment-expression

2.3.1. Primary expressions

Syntax:

primary-expression:

identifier

constant

(expression)

2.3.2. Postfix expressions

Syntax:

postfix-expression:

primary-expression

identifier (argument-expression)

identifier ( )

2.3.3. Argument expressions list

Syntax:

argument-expression-list:

assignment-expression

argument-expression-list , assignment-expression

Constraints

Though the syntax definition allows a function call in an argument list, the current implementation cannot deal with such a code. This is because a function call in an argument list requires stack operations in other function's stack operation, and the function in the list must be processed before evaluating the function that has the list as arguments.

2.3.4. Unary expressions

Syntax:

unary-expression:

postfix-expression

unary-operator unary-expression

unary-operator: One of the following.

+ -

2.3.5. Multiplicative operators

Syntax:

multiplicative-expression:

unary-expression

multiplicative-expression * unary-expression

multiplicative-expression / unary-expression

Constrains

The current compiler does not support a divide expression.

2.3.6. Additive operators

Syntax;

additive-expression:

multiplicative-expression

additive-expression + multiplicative-expression

additive-expression - multiplicative-expression

2.3.7. Relational operators

Syntax:

relational-expression:

additive-expression

relational-expression < additive-expression

relational-expression > additive-expression

relational-expression <= additive-expression

relational-expression >= additive-expression

2.3.8. Equality operators

Syntax:

equality-expression:

relational-expression

equality-expression == relational-expression

equality-expression != relational-expression

2.3.9. Assignment operators

Syntax:

assignment-expression:

identifier = equality-expression

2.4. Declarations

2.4.1. Declaration

Syntax:

declaration:

int identifier-list ;

identifier-list:

identifier

identifier-list , identifier

2.4.2. Parameter list

Syntax:

parameter-list:

parameter-declaration

parameter-list , parameter-declaration

parameter-declaration:

int identifier

2.5. Statements

Syntax:

statement:

compound-statement

expression-statement

if-statement

iteration-statement

return-statement

put_int-statement

2.5.1. Compound Statements

Syntax:

compound-statement:

{ block-item-list (opt) }

block-item-list:

block-item

block-item-list block-item

block-item:

declaration

statement

2.5.2. Expressions

Syntax:

expression-statement:

expression (opt) ;

2.5.3. if-statement

Syntax:

if-statement:

if (expression) statement

if (expression) statement else statement

2.5.4. Iteration-statements

Syntax:

iteration-statement:

while (expression) statement

do statement while (expression) ;

for (expression ; expression; expression) statement

2.5.4.1. while-statement

2.5.4.2. do-statement

2.5.4.3. for-statement

2.5.5. return statement

Syntax:

return-statement:

return expression (opt) ;

2.6. Translation unit

Syntax:

translation-unit:

external-declaration

translation-unit external-declaration

external-declaration:

function-definition

2.6.1. Functions

Syntax:

function-definition:

identifier (parameter-list) compound-statement

identifier ( ) compound-statement

2.7. Intrinsic function

“put_int” intrinsic function, which takes an integer parameter and outputs it on the standard output, is provided.

Reference

[1] ISO/IEC 9899:TC2, “Programming Language – C”, (Committee Draft), May 6, 2005