早稲田大学

言語処理系実習用言語 tl 言語仕様書

木村啓二

2016 年 3 月

1. はじめに

本文章は「言語処理系」実習用簡易言語「tl (tiny language)」の言語仕様書である。tl は C を言語仕様のベースとしつつ、これから大幅に機能を削減し、コンパイラ実装の容易さを確保した教育用の言語である。ただし、C をベースとしているため、標準的な C の文法をサポートする拡張の手段は残されている。

本言語仕様書は特に文法定義に関して参考文献[1]を参照している。

2. 言語

前述の通り、tl は C 言語のサブセットの言語仕様を持つ。具体的には以下のような制限を持つ。

- 変数は int 型の変数のみ宣言できる
- 定数は整定数のみ利用できる
- 式は+, -, *, /, <, <=, >=, >, ==, !=のみ利用できる
- 文は if, while, do-while, for, return のみ利用できる

以下の文法定義において、特に断りのない限りは文献[1]に記されている C 言語と同様の仕様を持つ。

2.1. 変数

変数としては関数内部で利用する局所変数を利用できる。

2.1.1. 変数の寿命

局所変数は、当該の変数が宣言された関数を実行する度にその変数のためのメモリが確保される。寿命は関数を実行してからその関数の実行を終了するまでである。関数内部のどの複合文で変数を宣言しようとも、変数の寿命は上記の通りで変わらない。そのため、変数名の有効範囲も関数内部となる。

2.1.2. 変数の型

変数の型としては整数型である **int** のみ利用可能である。

2.2. 字句

2.2.1. 予約語

tl の予約語は以下の通りである：

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

2.2.2. 識別子

識別子は英字で始まり英字、“_” もしくは数字の並びにより構成される。識別子は変数名や関数名として利用できる。

2.2.3. 定数

整定数のみ利用できる

2.3. 式（Expressions）

文法：

expression:

assignment-expression

2.3.1. 一次式（Primary expressions）

文法：

primary-expression:

identifier

constant

(expression)

2.3.2. 後置式（Postfix expressions）

文法：

postfix-expression:

primary-expression

identifier (argument-expression)

identifier ( )

2.3.3. 実引数式列（Argument expressions list）

文法：

argument-expression-list:

assignment-expression

argument-expression-list , assignment-expression

制限

文法上は実引数に関数呼び出しを持った式を記述できるが、現状ではこの記述を正しく処理するコードを生成できない。これは、スタックの操作中に別の関数のスタック操作を行うことになるためである。（場合によっては動作するかもしれない）

2.3.4. 単項演算子（Unary expressions）

文法：

unary-expression:

postfix-expression

unary-operator unary-expression

unary-operator: 以下のいずれか

+ -

2.3.5. 乗算演算子（Multiplicative operators）

文法：

multiplicative-expression:

unary-expression

multiplicative-expression * unary-expression

multiplicative-expression / unary-expression

制限

現在のコンパイラでは除算は実装されない。

2.3.6. 加算演算子（Additive operators）

文法：

additive-expression:

multiplicative-expression

additive-expression + multiplicative-expression

additive-expression - multiplicative-expression

2.3.7. 比較演算子（Relational operators）

文法：

relational-expression:

additive-expression

relational-expression < additive-expression

relational-expression > additive-expression

relational-expression <= additive-expression

relational-expression >= additive-expression

2.3.8. 等号演算子（Equality operators）

文法：

equality-expression:

relational-expression

equality-expression == relational-expression

equality-expression != relational-expression

2.3.9. 代入演算子（Assignment operators）

文法：

assignment-expression:

identifier = equality-expression

2.4. 宣言

2.4.1. 宣言（Declaration）

文法：

declaration:

int identifier-list ;

identifier-list:

identifier

identifier-list , identifier

2.4.2. 仮引数リスト（Parameter list）

文法：

parameter-list:

parameter-declaration

parameter-list , parameter-declaration

parameter-declaration:

int identifier

2.5. 文

文法：

statement:

compound-statement

expression-statement

if-statement

iteration-statement

return-statement

put_int-statement

2.5.1. 複合文

文法：

compound-statement:

{ block-item-list (opt) }

block-item-list:

block-item

block-item-list block-item

block-item:

declaration

statement

2.5.2. 式

文法：

expression-statement:

expression (opt) ;

2.5.3. if 文

文法:

if-statement:

if (expression) statement

if (expression) statement else statement

2.5.4. 繰り返し文

文法：

iteration-statement:

while (expression) statement

do statement while (expression) ;

for (expression ; expression; expression) statement

2.5.4.1. while 文

2.5.4.2. do 文

2.5.4.3. for 文

2.5.5. return 文

文法：

return-statement:

return expression (opt) ;

2.6. 外部定義

文法：

translation-unit:

external-declaration

translation-unit external-declaration

external-declaration:

function-definition

2.6.1. 関数

文法：

function-definition:

identifier (parameter-list) compound-statement

identifier ( ) compound-statement

2.7. 組み込み関数

int 型の引数を一つ取りその値を標準出力に表示する“put_int”が利用できる。

参考文献

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