早稲田大学

言語処理系実習用コンパイラ tlc 機能仕様書

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言語 tl のコンパイラ tlc の機能仕様書である。tlc は字句解析、構文解析、及び x86 (32 ビット) のコード生成器を備える。一方で、最適化は行わない。エラー処理もほぼ備えない。本処理系は Cygwin、Linux, Mac の各プラットフォームで動作可能であり、かつ各プラットフォームの上で動作する 32 ビットコードを生成する。

2. コンパイラの構築

まず、配布された tlc.zip を展開する。

```
unzip tlc.zip
```

ディレクトリ tl が展開される。ディレクトリ構成は以下の通りとなる。

- tl/
- tl/doc/
  - ドキュメントを格納したディレクトリ
- tl/src/
  - ソースファイルを格納したディレクトリ
- tl/src/test
  - テスト用の tl で記述されたプログラム

tl/src の Makefile の PLATFORM=の行をプラットフォームに合わせて修正する。例えば Cygwin を使っている場合は、“PLASFORM = CYGWIN”の行先頭の“#”を外し、他の PLATFORM =の行の先頭に“#”をつける。他のプラットフォームでも同様の修正を行う。

その後 make コマンドによりコンパイラを構築する

```
make
```

make が終了するとコンパイラ tlc が生成される。**(tlc のソースコードを修正するたびに make により tlc の再構築を行う必要がある)**

動作確認として、テストプログラムをコンパイルする。

```
cd test
../tlc test1.c
```

test1.s が生成されるので確認する。gcc で実行バイナリを生成できる。

```
gcc -m32 -o test1 test1.s
./test1
```

3. コンパイラの実行

ソースプログラムのファイル名を指定してコンパイラのコマンド tlc を実行する。tl の言語仕様は C の下位互換なので suffix は.c とする。例えば、ソースプログラム“sample.c”のコンパイルは以下の様になる：

```
tlc sample.c
```

コンパイル結果の x86(32 ビット)アセンブリファイルを suffix を.s に置き換えたファイルとして出力する。sample.c のコンパイル結果は sample.s となる。実行バイナリを生成するには、gcc を用いる：

```
gcc -m32 sample.s
```

-m32 を指定することで gcc は 32 ビット用バイナリを出力する。
このサンプルでは実行バイナリの名前は a.out になる。例えば出力されるファイル名を sample とする場合には-o sample とオプションを指定する。

4. コンパイラの構成

4.1. 字句解析

ソースプログラムの字句解析を行う。認識するトークンは以下の通りである：

- **記号**
  - **+, -, *, /, <, <=, >=, >, ==, !=, , （カンマ）, （（開き括弧）, ）（閉じ括弧）, ;**
- **整定数**
  - **0-9 の一つ以上の並び**
- **識別子**
  - **a-z もしくは A-Z で始まり a-z, A-Z, 0-9, _の 0 個以上の並び**
- **予約語**
  - **else, for, if, int, main, return, while**

字句解析器の生成には flex を用いる。

4.2. 構文解析

ソースプログラムを言語仕様にて定めた構文規則に沿って解析し、抽象構文木を生成する。構文解析器の生成には bison を用いる。

4.3. コード生成

抽象構文木より x86（32 ビット）のアセンブリを生成する。

5. 中間表現

中間表現は抽象構文木により構成される。抽象構文木のノードは以下の様に分類される：

- 関数
  - 関数の本体として文の双方向リストを抽象構文木ノードの子に連結する。翻訳単位中の関数は双方向リストにて管理される。各関数に対応した id 番号を持つ。
- 文
  - 代入文、if 文、while 文、for 文、return 文のいずれかを表す。各文に必要な式や文のリストを抽象構文木の子ノードに連結する。
- 式
  - 識別子、整定数、括弧で囲われた式、単項式、乗除算式、加減算式、比較式、統合式、代入式のいずれかを表す。格式の子ノードにオペランドの式等を連結する。

6. データ構造

6.1. シンボルテーブル

変数を管理するテーブル。関数のローカル変数用のシンボルテーブルを持つ。ローカル変数用シンボルテーブルはスタックフレーム中のオフセットが記録される。

6.2. 抽象構文木

関数、文、式を表すノードから構成される木構造。各ノードは親のノードを参照できる。

ノードの種類は AST_kind で表す。AST_kind は以下のいずれかをとる：

- AST_KIND_FUNC
  - 関数
- AST_KIND_STM
  - 文
- AST_KIND_EXP
  - 式

ノードは 4 つの子ノードを持つ。また、文リストを持つ。

ノードは各種別ごとの副種別を AST_sub_kind にて表す。AST_sub_kind は以下のいずれかとる：

- AST_KIND_STM の場合
  - AST_STM_LIST
    - 文リスト
  - AST_STM_ASIGN
    - 子 1 は代入式
  - AST_STM_IF
    - 子 1 は条件式、子 2 は then の文リスト、子 3 は else の文リスト
  - AST_STM_WHILE
    - 子 1 は継続判定式、子 2 はループボディの文リスト
  - AST_STM_FOR
    - 子 1 は初期値設定式、子 2 は継続判定式、子 3 は継続式、子 4 はループボディの文リスト
  - AST_STM_DOWHILE
    - 子 1 はループボディの文リスト、子 2 は継続判定式
  - AST_STM_RETURN
    - 子 1 は返値式
- AST_KIND_EXP の場合
  - AST_EXP_ASGN
    - 子 1 は左辺式、子 2 は右辺式
  - AST_EXP_IDENT
    - ノード中に変数名に対応するシンボルテーブルの値を保持する

- AST_EXP_CNST
  - ノード中に値を保持する
- AST_EXP_PRIME
  - 子 1 は括弧で囲われた式
- AST_EXP_CALL
  - 関数呼び出し。子 1 は関数名を表す AST_EXP_IDENT。引き数列はリストに保持される。
- AST_EXP_UNARY_PLUS
  - 子 1 は式
- AST_EXP_UNARY_MINUS
  - 子 1 は式
- AST_EXP_MUL
  - 子 1 は第 1 オペランド式、子 2 は第 2 オペランド式。以降の式は全て 2 項式で同一形式
- AST_EXP_DIV
- AST_EXP_ADD
- AST_EXP_SUB
- AST_EXP_LT
- AST_EXP_GT
- AST_EXP_LTE
- AST_EXP_GTE
- AST_EXP_EQ
- AST_EXP_NE

文は双方向リストにより連結され、関数の本体や while 文のループボディとして連結される。
関数は双方向リストにより連結され翻訳単位を表す。

7. tlc のテスト

tlc/src/test は tlc のテストを行うためのファイルが収められている。本ディレクトリの dotest.sh を利用することで各プラットフォーム用のテストを実施できる。Cygwin の場合は以下のように実行する。

```
./dotest.sh WIN
```

これにより、コンパイルのログ、アセンブリファイル、実行バイナリが tmp ディレクトリに生成される。ログとアセンブリファイルは tlc/src/test/WIN にあるファイルと比較され、差分があればその旨報告される。差分がなければ補画面上に出力はない。Mac や Linux の場合は、./dotest.sh の後をそれぞれ MAC, LIN として実行する。

8. 参考：tlc のソースファイル構造

- **tl_lex.l**
  - 字句解析定義
- **tl_gram.y**

- 構文解析定義

- util.[ch]
  - メモリ確保等ユーティリティー関数群
- symtab.[ch]
  - シンボルテーブル
- ast.[ch]
  - 構文木関連
- parse_action.[ch]
  - 構文解析のアクション関数群
- cg.[ch]
  - コード生成系
- main.c
  - メイン関数