プログラミング言語TL/1

著者：齊藤敦志（さいとうあつし）

概要

この文章はプログラミング言語 TL/1 の言語仕様のまとめです。 歴史的背景、字句的な構成 (トークナイズの規則) 、文法、個々の命令の一覧の順で記述します。

TL/1 の資料は乏しく、見つけることが出来た資料を元にして私の解釈や曖昧箇所（あいまいかしょ）の指摘を加えて言語仕様の体裁（ていさい）に再構成したものがこの文章です。 断片的な情報の寄せ集めを元に実際の動作を検証せずに私の理解に基づいて説明するものですので充分に厳密ではない可能性があります。 また、説明のために元資料にない用語を使う場合もありますので各種資料と突き合わせて検証する場合には用語が指す意味が厳密に一致しない可能性があることはご容赦ください。

この文章中で未定義としている箇所は私が見ることが出来た資料から判断がつかないことを陽に示したものです。 その他にもこの文書を元に言語処理系を作ろうとすれば色々と曖昧な箇所があるはずです。 残念ながら私は TL/1 の文法規則についてこれ以上の詳細についての情報を持ち合わせていませんので問い合わせには応じられません。 どこかにいるであろうより情報を持っている方が積極的に発信することを期待します。

組版にあたってはオープンソースの組版ソフトウェアである Typst を用いました。素晴らしいソフトウェアの開発者たちに感謝します。

目次

§１ 背景

TL/1 は大西博氏(おおにしひろし)によって設計されて MC6800 向けに処理系が作られ、月間アスキーの 1980 年 2 月号で発表されました。 後に MOS6502 や Z80A 向けに移植されました。さらに近年になって MC6809 向けに移植されるなど、レトロコンピュータ界隈(かいわい)ではいまだに存在感があるプログラミング言語です。

TL/1 という名前は Tiny Language 1 を略したものですがあきらかに PL/I をもじったものです。ホビー向けコンピュータでは非構造化プログラミング言語が主流であった頃ですので構造化言語の先駆者である PL/I に対する敬意の意図であると考えられます。

§２ 字句

§2.1 コメント

パーセント記号 (%) から次の改行までは無視され、プログラム的には空白文字と同等の意味しか持ちません。プログラムの説明などを記述するのに使ってください。

§2.2 識別子

英文字で始まり、後続する英数字 0 個以上で構成されます。識別子を構成しない文字 (空白や記号) の直前までを 1 個の語として解釈します。大文字と小文字は区別しません。識別子の長さに上限はありません。

予約語、手続き名、関数名、大域単変数名、大域配列名、小域単変数名、小域配列名が同じ綴(つづ)りを持つ可能性があります。宣言時に既存の名前と重複しないかチェックされません。宣言以外の箇所では処理系は以下の順序で名前の検索を試み、最初に発見された属性の識別子として解釈します。

1. 小域配列名
2. 小域単変数名
3. 大域配列名
4. 大域単変数名
5. 関数名
6. 手続き名
7. 予約語

§2.3 数値

4 種類の表現方法があります。

TL/1 で直接的に扱える数値はバイトサイズです。ゆえにプログラム中で現れる数値表現は常に 0 〜255 の範囲内です。

構文中で論理値を要求している箇所においては 255 を真、それ以外の値は偽と解釈します。論理値を返す関数は真として 255 を、偽として 0 を返します。

§2.3.1 十進定数

0 〜 9 の文字を 1 つ以上並べて 0 〜255 の範囲を数を表現できます。

頭に余計な 0 を付けた場合 (例えば 02 のような) の挙動は未定義です。

§2.3.2 十六進定数

記号 $ に続く 0 〜 9 または a 〜 f によって十六進表現での 0 〜255 を表します。 a 〜 f は大文字 (A 〜 F) を使っても同じ意味です。

記号 $ の後に空白文字を入れてはいけません。

十六進で二桁にするために余計な 0 を頭に付けること (例えば $0A のような) は許されますが、それよりも多くの 0 を頭に付けた場合 (例えば $002) の挙動は未定義です。

§2.3.3 文字リテラル定数

クオートで挟まれた一文字で表現します。その一文字のアスキーコードと同じ数値が書かれたものと見做されます。

たとえば 'A' というように書くと 65 と書いたのと同じ意味です。

§2.3.4 論理定数

識別子 TRUE もしくは FALSE で表します。 TRUE は $FF 、 FALSE は 0 と同じです。

§2.4 記号

その他、演算子や構文には記号を用いますが後述する構文の説明の中で取り上げます。

§2.5 空白文字

語の間に空白文字を挿入することが出来ます。語の区切りとしては 1 個あれば充分ですが、何

個連続しても意味は変わりませんので外観を整えるために活用して下さい。空白文字と解釈する文字は以下の通りです。

- 文字コード $1F 以下の文字 (制御文字)
- 空白
- ピリオド (.)
- セミコロン (;)

言語仕様としての要求というわけではありませんが使い方に習慣的な意味がある場合もあり、それについては後述します。

§ 3 プログラムの構成

プログラムは以下の順序で構成されます。

1. 手続き名宣言
2. 関数名宣言
3. 大域単変数名宣言
4. 大域配列名・配列の大きさ宣言
5. 主プログラムの定義
6. 手続き、または関数の定義群

それぞれの内容は以下のようになります。

§ 3.1 手続き名宣言

プログラム中で使用する手続き名を予約語 PROC に続けて書きます。 手続き名が複数の場合はカンマで区切ります。

```
PROC《手続き 1》,《手続き 2》,
《手続き 3》...
```

手続きが 0 個の場合は宣言を省略します。

§ 3.2 関数名宣言

プログラム中で使用する関数名を予約語 FUNC に続けて書きます。関数名が複数の場合はカンマで区切ります。

```
FUNC《関数 1》,《関数 2》,《関数 3》...
```

関数が 0 個の場合は宣言を省略します。

§ 3.3 大域単変数名宣言

プログラム全体で使用する単変数名を予約語 VAR に続けて書きます。大域単変数名が複数の場合はカンマで区切ります。

```
VAR《単変数 1》,《単変数 2》,《単変数 3》...
```

大域単変数が 0 個の場合は宣言を省略します。

後述の大域配列と合計して 256 バイト以内である必要があります。

更に副プログラムを呼出す場合は 2 バイトの余地が必要 (大域単変数と大域配列を合計した大きさが 254 バイト以下になる必要がある) です。この 2 バイトはリターンアドレスの大きさです。

§ 3.4 大域配列名・配列の大きさ宣言

プログラム全体で使用する配列名とその大きさを予約語 ARRAY に続けて書きます。配列の大きさは配列名の後に角括弧で囲んだ数値で表します。配列名が複数の場合はカンマで区切ります。

```
ARRAY  配列 1 [  配列 1 の大きさ  ],
       配列 2 [  配列 2 の大きさ  ],
       配列 3 [  配列 3 の大きさ  ] ...
```

配列が 0 個の場合は宣言を省略します。

配列の大きさは配列の添字(そえじ)の最大値です。 例えば A[10] と宣言した配列に対しては 0 ～ 10 の添字で安全に参照できることを意味します。C などの言語のように配列の要素数ではないことに注意を払ってください。

§ 3.5 主プログラムの定義

予約語 BEGIN と END で狭んだ 0 個以上の実行文から成ります。

§ 3.6 副プログラムの定義

副プログラムとは手続きか関数です。以下のような順序で構成されます。

1. 副プログラム名
2. 仮引数リスト
3. 小域単変数名宣言
4. 小域配列名・配列の大きさ宣言
5. 予約語 BEGIN
6. 0 個以上の実行文
7. 予約語 END

これが副プログラムの個数分だけ繰返されます。

副プログラムの定義は宣言の順序と一致しなくてもかまいません。

§ 3.6.1 副プログラム名

プログラムの最初の手続き名宣言か関数名宣言で宣言した名前です。

§ 3.6.2 仮引数リスト

丸括弧で囲まれた複数の識別子です。識別子が複数の場合はカンマで区切ります。

```
(《仮引数 1》,《仮引数 2》,《仮引数 3》...)
```

仮引数はいわゆる値渡しのみです。呼出し時の実引数で初期化される点が異なるだけの小域単変数です。(暗黙の小域単変数宣言)

仮引数が 0 個の場合は丸括弧ごと省略することも出来ます。

§ 3.6.3 小域単変数名宣言

当該(とうがい)の副プログラム内だけで参照可能な単変数を宣言します。 形式は大域単変数と同じです。

小域単変数が 0 個の場合は宣言を省略します。

大域単変数・大域配列と同様に、小域単変数と小域配列を合せた大きさが 256 バイト以下である必要があります。

§ 3.6.4 小域配列名宣言

当該の副プログラム内だけで参照可能な配列を宣言します。 形式は大域配列と同じです。

§ 4 実行文

§ 4.1 複文

文括弧で 0 個以上の実行文を囲うことで 1 個の実行文にまとめることが出来ます。

```
BEGIN 実行文リスト END
```

```
{ 実行文リスト }
```

```
[ 実行文リスト ]
```

```
( 実行文リスト )
```

まとめられた文は複文(ふくぶん)と呼ばれます。複文はあたかもひとつの実行文であるかのように振舞いますので、以降の説明で実行文が現れることが出来る箇所のどこにでも現れることが出来ます。 囲まれた実行文が 0 個の複文は空文(くうぶん)と呼びます。

どの括弧記号を使っても同じ意味ですが、必ず対応する閉じ括弧で閉じる必要があります。例えば { で始めて] で閉じるような使い方は出来ません。

複文中のそれぞれの実行文を区切るためにセミコロン (;) を書く習慣があります。 字句の説明で述べたようにセミコロンは空白と同じなのでセミコロンを書くことは仕様上の要求ではありませんが PL/I や PASCAL の文法を踏襲した習慣です。

主プログラム/副プログラムの本文も複文の一種ではありますが上記で示した通り BEGIN と END で挟む必要があり、他の括弧は使えません。

§ 4.2 STOP

実行を停止してモニタに飛びます。

主プログラムの最後には自動的に挿入されるので書かなくてもよいですが、主プログラム/副プログラムの任意の場所に書けます。

§ 4.3 RETURN

手続き、または関数から復帰します。

```
RETURN
```

```
RETURN《式》
```

手続きから復帰する場合には式を持たない書式で、関数から復帰する場合は 1 個の《式》が続く書式で書きます。

手続きの最後には自動的に挿入されるので書かなくてもよいですが、手続き中のどこででも使用することが出来ます。(一部の処理系では FOR ループ内で使ってはいけません。)

関数の定義内では必ず 1 つ以上使う必要があります。(コンパイル時にチェックされないことに注意してください。)

関数内で RETURN 文を通過せずに終端 (関数の定義の終りを表す END) に行き当たった場合の挙動は未定義です。

§ 4.4 FOR

《単変数》の値を変化させながら繰返しする処理を表します。

```
FOR《単変数》:=《式 1》TO《式 2》DO
《実行文》
```

```
FOR《単変数》:=《式 1》DOWNTO《式 2》DO
《実行文》
```

カウント用の《単変数》に式の値を代入し、《単変数》を 1 ずつ増加または減少させながら《実行文》を繰返します。

《単変数》の増分は、 TO を用いたとき +1 、DOWNTO を用いたときは -1 です。(一部の処理系では DOWNTO を使えません。)

§ 4.5 REPEAT

```
REPEAT《文リスト》UNTIL《式》
```

《式》の値が真値になるまで《文リスト》を繰返し実行します。《文リスト》は 0 個以上の実行文を並べたものです。

§ 4.6 WHILE

```
WHILE《式》DO《実行文》
```

《式》の値が偽ならば《実行文》を実行せずに次の処理へ移ります。

《式》の値が真の場合は《実行文》を実行して再び《式》の評価に戻ります。

§ 4.7 IF

```
IF《式》THEN《実行文 1》
```

```
IF《式》THEN《実行文 1》ELSE《実行文 2》
```

《式》の値が真なら《実行文 1》を実行します。

《式》の値が偽であり ELSE 節が省略されていないならば《実行文 2》を実行します。

《式》の値が偽であり ELSE 節が省略されているならば何もせずに次の処理へ移ります。(一部の処理系では ELSE 節は使えません。)

§ 4.8 CASE

```
CASE《式 0》OF
    《式 1》《実行文 1》
    ...
    《式 k-1》《実行文 k-1》
    ELSE《実行文 k》
```

《式 0》の値を《式 1》の値と比較して合致すれば《実行文 1》を実行します。 その後は《実行文 k》の次の処理に移ります。

合致しなければ、同様にして合致するまで次々と式と比較し、合致した式に対応した実行文を実行します。

式の箇所に予約語 ELSE が有った場合は無条件に合致したものとみなして《実行文 k》を実行します。

CASE 文における ELSE 節は CASE の最後の条件であることを示すマーカーでもあるので省略することは出来ません。 ELSE 節に実行すべき実行文がない場合は空文を書いてください。

§ 4.9 WRITE

```
WRITE(《式》:《出力リスト》)
```

《式》の値が表す出力装置に対して《出力リスト》の内容を出力します。数値と出力装置との対応付けについては未定義ですが、一般的に 0 はコンソール画面であるとされています。

出力リストは以下の出力要素からなり、ふたつ以上の場合はカンマで区切ります。

§ 4.9.1 式

式を記述します。 十進数左詰めで出力します。

§ 4.9.2 右詰め

```
#(《式 1》,《式 2》)
```

《式 2》の値を (“式 1”) の桁数で十進右詰めで出力します。

§ 4.9.3 文字列

```
"《文字列》"
```

ダブルクォーテーションで囲まれた《文字列》を出力します。

§ 4.9.4 アスキーコード

```
ASCII(《式》)
```

《式》で与えられたアスキーコードに相当する文字を出力します。

§ 4.9.5 空白

```
SPACE(《式》)
```

《式》で与えられた個数分の空白を出力します。《式》の値が 0 の場合は何も出力しません。

§ 4.9.6 改行

```
CRLF(《式》)
```

```
CRLF
```

《式》で与えられた個数分の改行を出力します。《式》の値が 0 の場合は何も出力しません。

《式》を省略した形式の場合は 1 個の改行を出力します。

§ 4.9.7 十六進数

```
HEX(《式》)
```

《式》で与えられた値を十六進数 2 桁で出力します。

一部の処理系では使えません。

§ 4.10 代入

```
《変数》:=《式》
```

```
《変数 1》,《変数 2》, ...,《変数 k》:= 式
```

《式》の値を《変数》に代入します。

変数がカンマで区切られたリストの場合は《式》の値を左辺全ての変数に代入します。

代入記号はコロンとイコールの 2 語から成っているのでコロンとイコールの間に空白文字が有っても代入記号として認識されますが、一般的には間を空けずに書きます。 処理系 (機種) によっては := ではなく ← を用いるものもあるようです。

§ 4.11 手続き呼出し

手続きを呼出します。

```
手続き名 (《式》,《式》, ...《式》 )
```

```
手続き名
```

引数をもつ手続きでは実引数を与えて呼出します。引数を持たない手続きを呼出す場合には丸括弧ごと省略した記法で呼出せますが、引数を持たない手続きを丸括弧を省略せずに記述した場合はエラーです。

```
FOO( ) % このような呼び方はエラー
```

実引数の渡し方はいわゆる値渡しに限定されているので実引数が変数であっても手続きから戻ったときに値は変化しません。

§ 4.12 CALL

機械語サブルーチンを呼出します。

```
CALL(《AH》,《AL》,《A》,《H》,《L》)
```

```
CALL(《AH》,《AL》,《A》,《H》)
```

```
CALL(《AH》,《AL》,《A》)
```

```
CALL(《AH》,《AL》)
```

各パラメータは以下の意味を持ちます。

《AH》	アドレスの上位 8 ビット
《AL》	アドレスの下位 8 ビット
《A》	アキュムレータに与える値
《H》	80 系 CPU では H レジスタに与える値、6502 系 CPU では X レジスタに与える値
《L》	80 系 CPU では L レジスタに与える値、6502 系 CPU では Y レジスタに与える値

《A》,《H》,《L》は省略した形式がありますが、省略した場合はそれぞれの値は不定となります。

この手続きは一部の処理系では利用できません。

§ 4.13 SENSE

PC では STOP キー、 APPLE では cont-C が押されているか否かを検出し、押されていればモニタモードに戻ります。

§ 5 変数

変数はすべて 1 バイト長です。 以下の 4 種類があります。

§ 5.1 単変数

VAR 宣言された英字で始まる英数字の列です。

大域、小域の区別があります。

§ 5.2 配列変数

```
《配列変数名》[《式》]
```

ARRAY 宣言された配列の《式》番目の要素です。

大域、小域の区別があります。

§ 5.3 MEM 変数

```
MEM(《式 1》,《式 2》)
```

《式 1》の値を上位、《式 2》の値を下位のアドレスとするメモリ内の 1 バイト。

§ 5.4 PORT

```
PORT (《式》)
```

PC 版専用です。

N-BASIC の INP , OUT に相当します。 代入文の左辺にあれば OUT 、右辺にあれば INP と同等の作用をします。

§ 6 式

§ 6.1 定数

数値の項で示した 4 種類のいずれかの形式で定数を表します。

§ 6.2 関数呼出し

```
《関数名》(《式 1》,《式 2》, ... ,《式 k》)
```

```
関数名
```

FUNC 宣言によって宣言された関数、または処理系が用意しているシステム関数 (後述) を呼出します。引数のない関数を呼出す場合は関数名のみで呼出せます。引数がない関数を括弧付きの書式で呼出そうとした場合はエラーです。

```
FOO := BAR() % このような呼び方はエラー
```

§ 6.2.1 MHIGH

1 バイト同士の掛け算の結果は 2 バイトになり得ますが、式の中では下位 1 バイトしか表現されません。 上位 1 バイトは専用の場所に格納されており、 MHIGH 関数で取出すことが出来ます。

§ 6.2.2 MOD

割り算すると商が返りますが、同時に余が計算されて専用の場所に格納されており、 MOD 関数で取出すことが出来ます。

§ 6.2.3 RND

```
RND(《式》)
```

1 以上《式》以下の一様乱数を返します。

§ 6.2.4 GET

```
GET(《式》)
```

《式》が表す入力装置から 1 文字を入力し、そのアスキーコードの値を返します。

数値と入力装置の対応付けは未定義ですが、一般に 0 はキーボードであるようです。

§ 6.2.5 READ

```
READ(《式》)
```

《式》の値に対応する入力装置から十進数を 1 つ入力し、その値を返します。RUBOUT コードは区切り記号とみなされます。

§ 6.2.6 NOT

```
NOT(《式》)
```

1 の補数を返します。 後述の COM と同じです。

§ 6.2.7 NEG

```
NEG(《式》)
```

2 の補数を返します。

§ 6.2.8 COM

```
COM(《式》)
```

1 の補数を返します。

§ 6.2.9 LSR

```
LSR(《式》)
```

1 ビット右シフトします。最上位ビットには 0 が入り、最下位ビットはキャリーに入ります。

§ 6.2.10 ASR

```
ASR(《式》)
```

1 ビット右シフトします。最上位ビットは変化せず、最下位ビットはキャリーに入ります。

§ 6.2.11 ASL

```
ASL(《式》)
```

1 ビット左シフトします。最下位ビットには 0 が入り、最上位ビットはキャリーに入ります。

§ 6.2.12 ROR

```
ROR(《式》)
```

1 ビット右シフトします。 キャリーは最上位ビットに入り、最下位ビットはキャリーに入ります。

§ 6.2.13 ROL

```
ROL(《式》)
```

1 ビット左シフトします。 キャリーは最下位ビットに入り、最上位ビットはキャリーに入ります。

§ 6.2.14 USR

```
USR(《AH》,《AL》,《A》,《AH》,《L》)
```

```
USR(《AH》,《AL》,《A》,《AH》)
```

```
USR(《AH》,《AL》,《A》)
```

```
USR(《AH》,《AL》)
```

CALL 文と機能は同じですが、機械語サブルーチン実行後のアキュムレータの値を返却値として返します。

処理系によっては使えません。

§ 6.2.15 RDHEX

```
RDHEX(《式》)
```

入力装置から十六進数 1 桁を入力します。

処理系によっては使えません。

§ 6.2.16 RRC

```
RRC(《式》)
```

キャリーを経由せずに式の値を右に 1 ビットシフトします。最下位ビットは最上位に入ります。

処理系によっては使えません。

§ 6.2.17 RLC

```
RLC(《式》)
```

キャリーを経由せずに式の値を左に 1 ビットシフトします。最上位ビットは最下位ビットに入ります。

処理系によっては使えません。

§ 6.3 二項演算子

左右に 2 個の項をとって計算する演算子です。

```
《項 1》《演算子》《項 2》
```

演算子の優先順位は表の通りです。優先順位の同じ演算子は左結合します。

1. 乗除算演算子
2. 加減算演算子
3. 関係演算子

4. 論理演算子
5. キャリー付き加減算演算子

優先順位を変更したい場合は式括弧を使用します。 以下 3 種類の括弧が式括弧として使えますが標準的には丸括弧を用いることとします。

```
{《式》}
```

```
[《式》]
```

```
(《式》)
```

演算子が記号ではなく識別子であるようなものについてはピリオドを両側に置く習慣があります。字句の項で述べたようにピリオドは空白の一種であり、ここでピリオドを使うことは仕様上の要求ではありません。外観で関数呼出しとの区別をしやすくする工夫だと考えられます。

```
.AND.
```

§ 6.3.1 乗除算演算子

*	乗算
/	除算の商

§ 6.3.2 加減算演算子

+	加算
-	減算

§ 6.3.3 関係演算子

2 つの値を比較して論理値を返します。 GT と LT は左右の数値を 2 の補数表現の符号付き二進数とみなして比較します。その他の演算子は数を符号なし二進数と解釈します。

>	大きい
<	小さい
#	等しくない
=	等しい
GT	大きい
LT	大きい

§ 6.3.4 論理演算子

AND	論理積
OR	論理和
EOR	排他的論理和

§ 6.3.5 キャリー付き加減算演算子

二項を足した上でキャリーフラグの値を足す、または二項を減算した上でキャリーフラグの値を引く演算子です。

ADC	キャリー付き加算
SBC	ボロー付き減算

一般的には機械語の ADC , SBC にそのまま対応します。つまり TL/1 の仕様でいうキャリーフラグとは CPU が持つフラグレジスタのキャリーフラグです。

システム関数の一部がフラグを変化させることが明記されている他はどの処理がフラグを変化させるかは未定義です。どのような機械語が生成されるか充分に理解していないと予想外なときにフラグが変化するかもしれません。処理系によっては配列要素へアクセスしたときに変化させてしまう場合もあることが知られています。