文字列解釈実行型言語 AIL†☆

河村 知行††

文字列の解釈実行を基本とする手続型言語 AIL を紹介する．AIL は，プログラムテキストを中間コードに変換することなく，直接解釈実行する．テキストのまま実行するため，実行速度は低下しているが，その見返りとして，種々の強力な機能を，小さな言語仕様の中で可能にしている．たとえば，文字列・コメント・関数本体は，同一の構文要素であり，また，関数呼出しと文字列の評価も統一されている．

1. はじめに

計算機の発達に伴い，これまで多くの言語が開発されてきた．それらの多くは，大型計算機上での大型計算をプログラミングするための言語であり，その言語自身も大型のものであった．

しかし，本論文で述べる言語 AIL は，それら大型言語と異なり，小さな言語仕様，小さな処理系により実現される言語である．このような言語がこれまでなかったわけではなく，初期の LISP[1] や D 32[2] などがある．

LISP と AIL と対比すると，LISP はポインタセルを基本としたインタプリタ型言語であり，プログラムもデータもすべてS式により表現されている．また，その処理系も，pure LISP に限れば，たいへん小さなものとなっている．

一方，AIL は，文字列を基本とするインタプリタ型言語であり，プログラム自身や，多くの構文要素が，文字列により表現されている．ただ，LISP がS式のみを扱う言語であるのに対し，AIL は，一般の手続き型言語の扱う数や文字や文字列を扱うように設計されている．

AIL の設計の目標は，手続き型言語において従来使われているスタック，ヒープ，変数，代入，間接参照などの要素をそのまま引き継ぎ，その中でできるだけ単純でかつ強力な言語を作ることであった．その結果が，文字列と文字列の解釈実行（インタプリト）を基本とした言語 AIL である．

LISP では，プログラムテキストをポインタの集りに置き換えて解釈実行しているのに対し，AIL では，プログラムテキストを文字列のまま扱い，そのままの形で解釈実行する点が大きく異なっている．文字列によるプログラム表現により，従来の言語では困難だったいくつかの記述が可能となった．

1) いくつかの構文要素を文字列で表すことにより，構文要素が減少した．
2) [♯S] (入力した文字列の解釈実行を意味する：後述) により，デバッグ時の変数の参照や代入が，言語の仕様の中に組み込めた．
3) 補助記憶からの文字の読み込みと，その実行により，オーバレイ制御が，そのための特別な言語仕様なしに実現できた．
4) 文字列の引数機構による受渡し・文字列の配列により，手続きの引数化・手続き配列が，そのための特別な言語仕様なしに実現できた．
5) 文字列をそのままプログラムとみなすため，言語処理系と，既存のエディタとの結合を容易に行えた．

制御構造に関しては，カッコ式（後述）に統一されている．カッコ式は，LISP の COND 式の拡張と手続き型言語におけるループ文とを融合したものになっている．

この言語の応用には，次のようなものが考えられる．

1) 実用的使用よりも，実験的使用に向いていると思われる．とくに，ヒープ・スタック・変数・代入・間接参照を扱えるので，ノイマン型計算機言語の入門として使用できる．
2) 実行時に手続きをプログラムで生成して，それに制御を移すことができるため，これまで LISP でしか実現が困難だった問題が，直感的に理解しやすい文字列の形で実現できる．
3) LISP のようにポインタを扱うのでなく，文字

† String Interpretation Directed Language AIL by KAWAMURA TOMOYUKI (Department of Information Electronics, Tokuyama Technical College).
†† 徳山工業高等専門学校情報電子工学科
☆ An Interpretation Language

列をそのまま解釈実行することから，その処理過程が理解しやすく，初学者が解釈実行型言語のそれらしい機能（文字列の評価やプログラムの実行時生成など）を学ぶのに適当である．

4) 処理系が小さく作成が容易なことから，初学者の言語処理系作成問題として適当である．

2. 言語仕様

AIL は，セルを基本単位としている．セルは，16 ビットないし 32 ビットの記憶単位で，整数を値とする．また，セルは，プログラム・スタック・ヒープ領域を形成する．**図1**に標準的メモリマップを示す．

AIL で使用する文字および文字コードは，JIS7 または，JIS8 とする．常に1文字1セルを占有するものとする．

AIL は，タイプレスな式言語 (expression language) であり，値に型はない．値をもたない手続きはなく，すべて関数である．また，プログラムの実行は，**左から右**に行われることを大原則としている．さらに，AIL は，ソースプログラムを計算機の主記憶上にそのままの形で置いて，それを解釈実行（インタプリト）することを指向した言語である．

AIL の構文流れ図を**図2**に示す．AIL の構文は，図2からもわかるように小さなものである．以下に各項目を説明する．

2.1 定　数

定数は，数定数・文字定数・文字列定数のいずれかである．数定数は数字の並びである．その値は数字の並びを 10 進数とみなした値である．文字定数は文字 % とその直後の1文字より成る．その値は，文字 % の直後の文字を文字コードとしてみなした値である．値は必要な 0 を左に補って，1セルで表現される．文字列定数は，任意の文字列を文字 { }* で囲ったものである．その値は，先頭の文字 { が格納されているセルの計算機（プログラム）内での番地である．この値（番地）をとくに，文字列ポインタという．両端の文字 { } で囲まれた部分は，文字 { } と ( ) についてバランスがとれていなければならない．また，各文字は文字定数と同じく1セルを使って表現される．

* { } が使用できない場合は，"(と) を代わりに用いる．

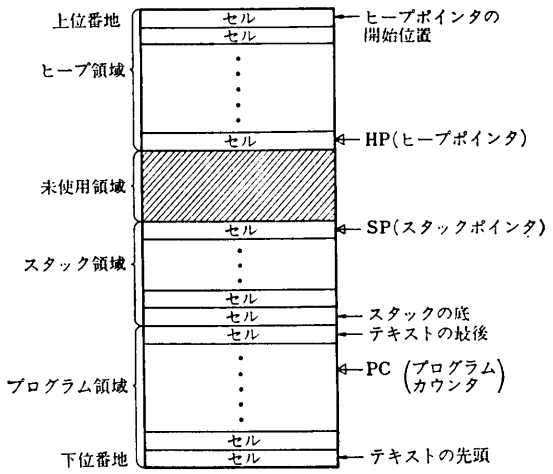

図 1 AIL のメモリマップ
Fig. 1 Memory map of AIL.

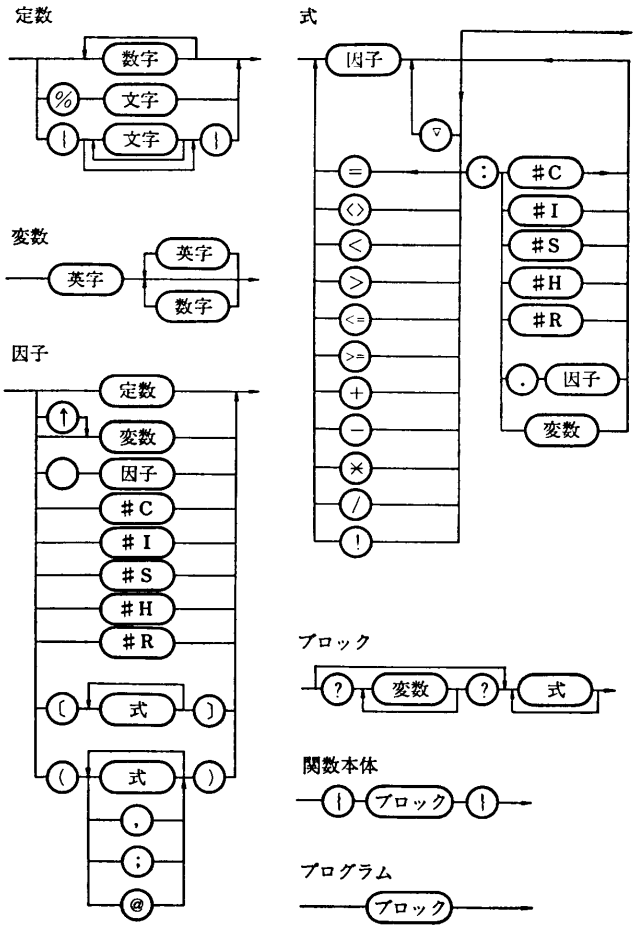

図 2 AIL 構文流れ図
Fig. 2 AIL syntax chart.

2.2 変　数

変数は，英字で始まる英数字の列である．広域変数と局所変数の2種がある．広域変数はプログラムの先頭で宣言し，プログラム中のどこからでもアクセスが可能である．局所変数は関数の先頭で宣言し，その関数の中だけでアクセスが可能である．広域変数名と局所変数名が同じ場合は，局所変数がアクセスされる．変数の scope は，内部手続きを用いずトップレベルの手続きだけで記述された PASCAL プログラムと同じであり，LISP や APL[3] で見られる局所変数のネスティングは行わない．局所変数は引数機構により，初期値を与えることが可能である．

2.3 システム変数

#I・#C・#S・#H・#R がシステム変数である．それぞれ，整数・文字・文字列・ヒープポインタ・関数からの返り値を表す．

#I・#C・#S は，その値が参照された時入力を行う．入力した値がシステム変数の値である．

#I では，入力数字列を10進数として読み込み，値とする．最後の数字の次の1文字が，無条件に読み捨てられる．なぜなら，AIL には，PASCAL の READLN に対応する機能がないため，端末から入力した数字列の直後にある行末文字（制御文字）を，自動的に読み飛ばす等のためである．

#C では，入力された文字の文字コードが値となる．

#S では，行末文字の直前まで読み込んで，文字{ }で囲んでヒープ領域に格納し，文字{が格納されたセルの番地（文字列ポインタ）をその値とする．行末文字は読み捨てられる．

#I・#C・#S に代入が行われたときには，出力を行う．

#I では，代入する値を整数とみなして，システムで定められた桁数内に右詰めで出力する．

#C では，代入する値を文字コードとみなして，1文字出力する．

#S では，代入する値は必ず文字列ポインタでなければならない．このポインタが指す文字列の両端の文字{ }を除いた文字列の内容の部分のみを出力する．

#H は，ヒープポインタで，これの参照または代入により，ヒープ領域を自由に操作できる．配列等は，ユーザがヒープ領域に，実行時に取ることになる．

#R は，関数からのリターンを行う．#R に代入が行われたときは，代入する値を関数値として，現在実行されている関数からリターンする．#R がたんに参照された時は，0 を関数値として，関数からリターンする．

2.4 カッコ式

一般の計算機言語では，文字( )は，算術演算における計算の順序を指定する役割をもつ．AIL では，この機能を大幅に拡張して，他言語の IF 文や，LOOP 文に対応する機能をもたせている．これをカッコ式と呼ぶ．カッコ式の最も単純な形は，(式$_1$ 式$_2$…式$_n$) である．これは各式を左から右に順に評価していき，式$_n$ の値をこのカッコ式の値とする（式がないとき，値は0）．一般のカッコ式は，式と式の間に文字 , ; @ が含まれる．各式を評価していく過程で，文字 , ; @ のいずれかに出会ったなら，以下 1)〜4) のように動作して制御の流れを変える．ただし，以下の説明で現在の値とは，時系列上で直前に評価した式の値である．また，走査とは，文字 ( ) のバランスのとれたものの走査であり，文字 ( ) に関して，走査開始時点より深いレベルの文字は走査の対象とはならない．走査は評価ではないので，現在の値は変化しない．

1) ',' のとき

現在の値により，次に評価する位置を変える．現在の値が 0 以外ならば，この文字 ',' の次の文字から評価を続ける．現在の値が 0 ならば，プログラムを左から右に走査して，最初の ';' の次の文字から評価を続ける．一般の計算機言語の IF 文に対応する．

2) ';' のとき

プログラムを左から右に走査して，最初の ')' から評価を続ける．一般の計算機言語の ELSE に対応する．

3) '@' のとき

プログラムを**右から左**に走査して，最初の '(' の右の文字から再度評価を続ける．一般の計算機言語の LOOP 文に対応する．

4) ')' のとき

現在の値をこのカッコ式の値として，このカッコ式の評価を終える．

以下に，PASCAL プログラムと対応する AIL プログラムを示す．

例1 PASCAL : if A then B else C
AIL : (A, B ; C)

例2 PASCAL : if A and B then C
else if D and E then F
else if G then H
AIL : (A, B, C ; D, E, F ; G, H ;)

例3 PASCAL: while A do begin B; C end
AIL : (A, B C @;)

例4 PASCAL: repeat A; B until C
AIL : (A B C,; @)

例5 PASCAL:
```
10: A1; A2
    if B then goto 20;
    C1; C2;
    goto 10;
20:
```
AIL : (A1 A2 B,; C1 C2 @)

2.5 関 数

関数の本体と定義は多くの場合（常にではない），次のような形をしている．

{?$v_1 \cdots v_n$? $e_1 \cdots e_l$} : F

文字：は文字列ポインタの代入を示す．$v_i(1 \leqq i \leqq n)$ が局所変数，$e_k(1 \leqq k \leqq l)$ が評価される式である．関数呼び出しは，実引数の並びを初めに置いて，[式$_1$…式$_m$ F] の形をしている．局所変数 $v_i(i \leqq m)$ は，式$_i$ の値により初期化される．残りの変数 $v_j(m < j \leqq n)$ は，作業用変数である．呼び出された関数は，式 e_k を左から右に順に評価し，式 e_1 の値を関数の値として呼び出した部分に制御を戻す（すなわちリターンする）（e_k がないとき，値は ∅）．?$v_1 \cdots v_n$?（局所変数）および 式$_1$…式$_m$（実引数）はなくてもよい．とくに局所変数がない場合は，関数呼び出しを，文字列の単なる評価 (eval) と考えることができる．

例6 {?X? X+X} : F
[5 F] : #I
整数 10 が出力される．

例7 {{ABC} : #S {XYZ} : #S} : G
[G]
文字列 ABCXYZ が出力される．

関数呼び出しの規則は，[ ] で囲まれた式の並び（必ず一つ以上なければならない）の右端の式の値が，文字列ポインタでなければならないということである．関数呼び出しの実現は，[ ] で囲まれた式の並びの各値をスタックに積んで，最後に積んだ値をポップアップし，それが文字列ポインタであること（値（＝番地）が指し示すセルの値が '{' であること）を確認して，そこに一時的に制御を移す（サブルーチンコール）ことにより行われる．

例8 [X Y (X=Y, F; G)]
X, Y を実引数とし，X=Y のときには関数 F を，そうでなければ関数 G を呼び出す．

また，関数の本体を関数呼び出しの中に置くことが可能である．この場合，関数の定義は不要である．

例9 [5 {?X? X+X}] : #I
10 が出力される．（例6の変形）

例10 [{123 : #I}]
123 が出力される．たんに 123 : #I としたのと同じ．

例11 [#S]
実行時に入力された文字列が評価（関数呼び出し）される．たとえば，A : #I と端末から入力すれば，その時点での A の値を表示できる．これは，デバッグ機能として使用できる．

さらに進んで，実行時にプログラムでヒープ上に関数の本体（文字列）を生成して，それを呼び出すことも可能である．また，文字列ポインタを配列の要素に格納して，関数配列を作ることも可能である．関数（文字列ポインタ）を実引数として，他の関数に渡すことも容易に行える．

また，AIL の構文規則のなかには，関数定義の構文規則は存在しない．なぜなら，それは文字列ポインタの変数への代入そのものだからである．

このように，AIL における関数本体は，文字列と形式上の区別がなく，その文字列が文字列ポインタにより1セルの値として扱われるので，関数の取り扱いが非常に柔軟性に富んだものになっている．

2.6 因 子

因子は，定数，変数，番地，間接参照，システム変数，関数呼び出し，カッコ式のいずれかである．定数は，2.1 節で示された定数であり，その値を因子の値とする．変数は，2.2 節で示された変数であり，変数に対応してスタック上に確保されたセルの値を因子の値とする．番地は↑〈変数〉の形をしており，変数に対応してスタック上に確保されたセルの番地を因子の値とする．間接参照は .〈因子〉の形をしており，〈因子〉の値を計算機内の番地とみなして，その番地が示すセルの値を因子の値とする．システム変数は，2.3 節で示された入力を行い，その値を因子の値とする．関数呼び出しは，呼び出した関数の値を因子の値とする．カッコ式は，カッコ式の値を因子の値とする．

2.7 式

式は因子だけより成るか，または，複数個の因子を演算子により結んだものである．因子だけよりなる場合は，因子の値が式の値である．演算子がある場合は，演算の値が式の値である．演算子が複数個ある場

合は，左から右に演算が行われて，最後に行った演算の値が式の値である．すなわち，次の2式は等価である．

A+B*C+D　　((A+B)*C)+D

演算子には，＝(等号)，〈 〉(≠の意)，＞，＞＝(≧の意)，＜，＜＝(≦の意)，＋，－，＊(乗算子)，/(除算子)，！(剰余を値とする演算子)，' (符号反転子)，：(代入演算子) がある．比較演算の値はその真偽により，1(真) または ∅(偽) の値をとる．加減乗除演算子は一般的な計算機の定義* に従う．符号反転子は唯一の一項演算子であり，演算子の左辺の値をオペランドとする．代入演算子は左辺の値を右辺の変数に代入する．ただし ：.〈因子〉 の形は間接代入であり，〈因子〉 の値を番地として，それが示すセルに代入が行われる．また，システム変数への代入は，2.3 節で示された出力を起こす．代入演算の値は，代入された値である．

PASCAL と AIL を対比して，以下に例を示す．

例12　PASCAL：write (－(100＋10))
　　AIL　　：100＋10'：＃I

例13　PASCAL：read (I) ; I＞＝123
　　AIL　　：＃I＞＝123

例14　PASCAL：VAR Q：array [∅..99] of integer
　　AIL　　：＃H-100：＃H：Q

例15　PASCAL：Q[X]：＝Q[X]＋1
　　AIL　　：.(Q+X)+1：.(Q+X)

2.8 ブロック

ブロックは，式の並び，または，式の並びの前に変数宣言をつけたものである．変数宣言とは，文字？で変数の並びを囲ったものである．

2.9 関数本体

関数本体は，文字｛ ｝でブロックを囲ったものである．関数本体は，プログラム中の文字列定数，または，実行時にヒープ領域上に作り出した文字列である．

2.10 プログラム

プログラムは，ブロックである．変数宣言は，広域変数の宣言とみなされる．

3. 言語の特徴

3.1 文字列定数の汎用性

文字列定数・コメント・関数本体は，構文的には同一の要素である．

例16　{CHOHOKEI NO MENSEKI}
　　{?X Y?　X＊Y}：MENSEKI
　　{2 HEN O NYURYOKU SEYO=}：＃S
　　[＃I ＃I MENSEKI]：＃I

例16の文字｛ ｝で囲まれた部分は，上からそれぞれコメント，関数本体，文字列定数として扱われているが，形としてはすべて文字列定数である．コメントは，文字列ポインタを値としてもつものであるが，その値が何にも使われず，式の評価が終わってしまうのである．

3.2 関数呼び出しと文字列評価

関数本体と文字列が同一のものであると同時に，関数呼び出しと文字列の評価も同一のものである．引数のない関数呼び出しが，文字列の評価というわけである．また，関数呼び出しの［ ］は，LISP の eval，文字列の｛ ｝は，LISP の quote と似た意味をもっている．すなわち，次の四つの式は等価である．

A　　[{A}]　　[[{{A}}]]　　[{[{A}]}]

文字列定数，カッコ式，関数呼び出しを，{A}，(A)，[A] とするとき，これらは即 (immediate)，直接 (direct)，間接 (indirect) の関係にあるといえる．

3.3 制御構造の統一

AIL の制御構造は，カッコ式と関数呼び出しの二つしかない．カッコ式は，2.4 節でも述べたように，いくつかの制御構造を合わせたものになっている．

3.4 オーバレイ

補助記憶からの入力（拡張 AIL の仕様）により，オーバレイ構造が容易に実現できる．すなわち，あらかじめ作成しておいたオーバレイサブプログラムを，実行時にヒープ領域に読み込み，それを関数呼び出しすればよいのである．

3.5 エディタ結合

AIL は，ソースプログラムを主記憶上にそのままの形で置いて，解釈実行（インタプリト）するシステムである．そのため，AIL プログラム用のエディタとしては，主記憶上に全ソーステキストを置く形のテキストエディタで十分である．この形のテキストエディタは，入手するのも作成するのも容易である．使用できるテキストエディタがあれば，AIL 処理系をそのなかに埋め込むことにより，専用エディタを作ることなく対話型システムにすることが可能である．この点が，BASIC[4] や APL 等の中間言語表現をもつインタプリタ型言語と AIL の大きな違いである．

---

* 表現可能な数の上下限，負数の演算についてはあえて定義しない．

4. 稼動状況

現在 OKI 4300C (16ビット語) と，μCOM 16 PASCAL システム[5] (16ビット語) の上で，AIL は稼動している．付録Aは，μCOM 16 PASCAL システムでのインタプリタである．約230行で記述されていることからも，言語仕様の小ささがわかる．一方 OKI 4300 C 上の AIL は，拡張 AIL であり，そのインタプリタは機械語で書かれている．そして，OKI 4300 C に接続された4台の端末装置から同時に，AIL システム (エディタを含む) を使用することが可能となっている．

現在 AIL で記述され稼動している本格的プログラムは，次の三つである．

μCOM 16 マイクロアセンブラ

μCOM 1600 アセンブラ

任意多倍長計算システム

OKI 4300C-AIL はこれまでに三つの版が作られている．第1版では，OKI 4300 Cにメモリ保護機能がないため，セル (16 ビット語) のアクセスの度にメモリ使用権の検査をしていた．第2版では，メモリ使用権の検査を必要なところだけに絞った．第3版では，プログラムの各文字が置かれているセルの未使用部分 (16 ビット中7ビットを文字に使用．9ビット未使用) を利用し，実行時にある種の情報のコード化を行った．これら各版で，5女王問題(8女王問題の変形)を解いたところ，次のような CPU 時間を必要とした．

第1版：26秒　　第2版：15秒　　第3版：6秒

AIL は，文字列を直接解釈実行するため，たしかに遅いシステムである．しかし，アセンブラや，その他初歩的なツールを記述する言語としては，実用にも耐えられるというのが，使用しての実感である．

5. むすび

AIL は文字列を基本とした言語である．そのためにも，文字列関数 (比較・検索・挿入・置換・連接) を自然な形で導入すべきであると考えている (現在，拡張 AIL は，文字列比較の演算子をもっている)．今後は，上記の文字列関数を，できるだけ自然な形で表現できる機構の開発が必要であろう．

参考文献

1) 中西正和：LISP 入門，p. 194，近代科学社，東京 (1977)．
2) 和田英一，高橋義道：システムプログラム言語 D32，D34 の試作，第12回プログラミングシンポジウム報告集，pp. 183-196 (1971)．
3) 竹下 享：プログラミング言語 APL，p. 293，共立出版，東京 (1981)．
4) Kemeny, G. John and Kurtz, E. Thomas：森口繁一監訳：ベーシック入門，p. 248，共立出版，東京 (1971)．
5) 河村知行：パーソナルコンピュータの総合設計，第23回プログラミングシンポジウム報告集，pp. 1-10 (1982)．

付　録

1) μCOM 16 PASCAL は，標準 PASCAL に準拠するものであるが，FILE 型，REAL 型，PACK 機能，手続き外ジャンプ，手続き引数化機構がない．

2) I＃CHAR，RESET (5, …)，READ＃5 (CH) は，μCOM 16 PASCAL 固有の記法である．

I＃CHAR の＃は，左辺の型を右辺で示される型に，強制的に変更するのに用いられる．

reset (5, ファイル名, 状態変数) は，論理機器番号5のもとに，ファイル名のファイルを OPEN する文である．状態変数には ERROR CODE (正常時 0) が返される．

read＃5 (CH) は，論理機器番号5から1文字読み込む文である．

3) ＃R は，実現されていない．実現するには，EXPR ではそのまま，FACTOR では，push (φ) を行った後，FUNC の最後の行に GOTO すればよい．しかしこれは，PASCAL の文法では許されない．FUNC の中に，EXPR と FACTOR を内部手続きとして書き直せば，手続き外 GOTO で実現できる．しかし，μCOM 16 PASCAL では，許されない．

4) ＃D は，拡張 AIL の仕様である．＃D への文字列の代入によりファイルが OPEN され，＃D の参照により，そのファイルから1文字入力される．一時期に OPEN できるファイルは一つであり，入力は文字入力だけである．

```
(*******************************************************)
(*          AIL INTERPRETER   IN   PASCAL              *)
(*                    BY  T. KAWAMURA       1982.5.13  *)
(*******************************************************)

PROGRAM AIL;
CONST ENDSTK = 1000;
TYPE STKTYPE=ARRAY[1..ENDSTK] OF INTEGER;
VAR T, T2, TP, PP, BOS, BP, SP, HP, PC, TMP, CNT : INTEGER;
    CH, TCH : CHAR;
    STK : ^ STKTYPE;

PROCEDURE ERROR(I:INTEGER); BEGIN WRITELN('ERROR',I,PC); END;
PROCEDURE PUSH(I:INTEGER); BEGIN SP:=SP+1; STK^[SP]:=I; END;
PROCEDURE POP;  BEGIN SP:=SP-1; END;
FUNCTION  POPVAL:INTEGER;  BEGIN POPVAL:=STK^[SP]; SP:=SP-1; END;
PROCEDURE APPHP(I:CHAR);  BEGIN HP:=HP-1; STK^[HP]:=I#INTEGER; END;
FUNCTION  NEXTCH:CHAR;  BEGIN PC:=PC+1; CH:=STK^[PC]#CHAR; NEXTCH:=CH; END;
FUNCTION  GETCH :CHAR;  BEGIN WHILE NEXTCH=' ' DO; GETCH:=CH; END;
PROCEDURE GETCHP;       BEGIN WHILE NEXTCH=' ' DO; END;
PROCEDURE LOAD;  BEGIN STK^[SP]:=STK^[STK^[SP]]; END;
PROCEDURE STORE; BEGIN STK^[POPVAL]:=STK^[SP];  END;
PROCEDURE EXPR; FORWARD;

FUNCTION  ALFDGT(I:INTEGER):BOOLEAN;
LABEL 1;
BEGIN ALFDGT:=TRUE;
 IF ('A'<=I#CHAR) AND (I#CHAR<='Z') THEN GOTO 1;
 IF ('0'<=I#CHAR) AND (I#CHAR<='9') THEN GOTO 1;
 ALFDGT:=FALSE;
1:END;

PROCEDURE SEARCH(DLMTR:CHAR);
LABEL 1;
BEGIN TMP:=0;
 REPEAT IF GETCH=DLMTR THEN
          IF TMP=0 THEN BEGIN GETCHP; GOTO 1; END;
        IF      CH='%' THEN PC:=PC+1
        ELSE IF CH='(' THEN TMP:=TMP+1
        ELSE IF CH=')' THEN BEGIN TMP:=TMP-1; IF TMP<0 THEN ERROR(1); END
        ELSE IF CH='_' THEN ERROR(2);  (*####*)
 UNTIL FALSE;
1:END;

PROCEDURE BACK;
LABEL 1;
BEGIN TMP:=0;
 REPEAT PC:=PC-1; CH:=STK^[PC]#CHAR;
        IF CH=')' THEN
         BEGIN IF STK^[PC-1]#CHAR<>'%' THEN TMP:=TMP+1; END
        ELSE IF CH='(' THEN
         IF STK^[PC-1]#CHAR<>'%' THEN
          IF TMP=0 THEN BEGIN GETCHP; GOTO 1; END
          ELSE TMP:=TMP-1;
 UNTIL FALSE;
1:END;

FUNCTION  FINDVR(ROOT:INTEGER):INTEGER;
LABEL 1, 2;
BEGIN FINDVR:=0;
 IF STK^[ROOT]=0 THEN GOTO 1;
 TP:=STK^[ROOT];
 IF STK^[TP]#CHAR='?' THEN TP:=TP+1 ELSE ERROR(3);
 PP:=PC; CNT:=1;
 REPEAT WHILE STK^[TP]#CHAR=' ' DO TP:=TP+1;
        IF STK^[TP]#CHAR='?' THEN GOTO 1;
        REPEAT IF ALFDGT(STK^[TP]) THEN
                 IF STK^[TP]=STK^[PP] THEN BEGIN TP:=TP+1; PP:=PP+1; END
                 ELSE BEGIN WHILE ALFDGT(STK^[TP]) DO TP:=TP+1;
                            PP:=PC; CNT:=CNT+1; GOTO 2;
                      END
               ELSE IF ALFDGT(STK^[PP]) THEN
                      BEGIN PP:=PC; CNT:=CNT+1; GOTO 2; END
                    ELSE BEGIN PC:=PP-1; GETCHP; FINDVR:=ROOT+CNT; GOTO 1 END
        UNTIL FALSE;
2:UNTIL FALSE;
1:END;

PROCEDURE VARADDR;
LABEL 1;
BEGIN
 T:=FINDVR(BP);  IF T<>0 THEN BEGIN PUSH(T); GOTO 1; END;
 T:=FINDVR(BOS); IF T<>0 THEN BEGIN PUSH(T); GOTO 1; END;
 ERROR(4);
1:END;

PROCEDURE WRITESTRING;
LABEL 1;
BEGIN
 FOR T:=STK^[SP]+2 TO ENDSTK DO
  IF STK^[T]#CHAR=')' THEN GOTO 1 ELSE WRITE(STK^[T]#CHAR);
1:END;

PROCEDURE READSTRING;
BEGIN TMP:=SP;
 WHILE NOT EOLN(INPUT) DO BEGIN READ(TCH); PUSH(TCH#INTEGER); END; READLN;
 APPHP(')');
 WHILE SP<>TMP DO APPHP(POPVAL#CHAR);
 APPHP('('); APPHP('"');
 PUSH(HP);
END;

PROCEDURE DIGIT;
BEGIN T:=0;
 REPEAT T:=T*10+(CH#INTEGER-'0'#INTEGER);
 UNTIL (NEXTCH<'0') OR ('9'<CH);
 PUSH(T); PC:=PC-1; GETCHP;
END;
```

付録 A　AIL インタプリタ

```
PROCEDURE PAREXPR;
LABEL 1;
BEGIN PUSH(0);
 REPEAT IF CH=')' THEN BEGIN GETCHP; GOTO 1; END;
        IF CH=';' THEN BEGIN SEARCH(')'); GOTO 1; END;
        IF CH=',' THEN IF STK^[SP]=0 THEN SEARCH(';')
                                     ELSE GETCHP
        ELSE IF CH='@' THEN BACK
        ELSE BEGIN POP; EXPR; END;
 UNTIL FALSE;
1:END;

PROCEDURE CHKFUN;
LABEL 1;
BEGIN CNT:=0; TP:=0;
 IF CH='?' THEN
  BEGIN TP:=PC; GETCHP;
   REPEAT IF CH='?' THEN BEGIN GETCHP; GOTO 1; END;
          CNT:=CNT+1;
          WHILE ALFDGT(NEXTCH#INTEGER) DO; PC:=PC-1; GETCHP;
   UNTIL FALSE;
  END;
1:END;

PROCEDURE FUNC;
BEGIN PUSH(BP); PUSH(0); PUSH(0); PUSH(0);
 REPEAT EXPR;
        T:=POPVAL; T2:=POPVAL+1;
        PUSH(T); PUSH(T2);
 UNTIL CH=']';
 STK^[SP-STK^[SP]-2]:=PC;  (*RETURN ADDRESS*)
 BP:=SP-STK^[SP]-1;        (*NEW BASE POINTER*)
 PC:=STK^[SP-1]+1; (* NEW PC *)
 GETCHP; CHKFUN;   (* SET TP , CNT *)
 STK^[BP]:=TP;  (* FORMAL PARAMETER POINTER SET *)
 CNT:=CNT-(STK^[SP]-1);
 IF CNT<0 THEN ERROR(5);
 SP:=SP-2+CNT;  (* NEW SP *)
 IF SP+30>=HP THEN ERROR(6);
 PUSH(0); WHILE CH<>')' DO BEGIN POP; EXPR; END;
 T:=BP-2; PC:=STK^[T+1]; BP:=STK^[T]; STK^[T]:=STK^[SP]; SP:=T;   GETCHP;
END;

PROCEDURE FACTOR;
BEGIN
 CASE CH OF
  '0','1','2','3','4','5','6','7','8','9' : DIGIT;
  '%' : BEGIN PUSH(NEXTCH#INTEGER); GETCHP; END;
  '"' : BEGIN PUSH(PC); PC:=PC+1; SEARCH('"'); END;
  '^' : BEGIN GETCHP; VARADDR; END;
  'A','B','C','D','E','F','G','H','I','J','K','L','M',
  'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'
      : BEGIN VARADDR; LOAD; END;
  '.' : BEGIN GETCHP; FACTOR; LOAD; END;
  '#' : BEGIN
         IF     NEXTCH='C' THEN BEGIN READ(TCH); PUSH(TCH#INTEGER); END
         ELSE IF CH  ='I' THEN BEGIN READ(T); PUSH(T); READ(TCH); END
         ELSE IF CH  ='S' THEN READSTRING
         ELSE IF CH  ='H' THEN PUSH(HP)
         ELSE IF CH  ='R' THEN ERROR(11)  (* NOT IMPLEMENTED *)
         ELSE ERROR(7);
         GETCHP;
        END;
  '[' : BEGIN GETCHP; FUNC; END;
  '(' : BEGIN GETCHP; PAREXPR; END;
  ' ','!','$','&',''','*','+',',','-','/',':',';','<','=','>','?',
  '@','$',']';'_'
      : ERROR(8);
 END;
END;

PROCEDURE EXPR;
LABEL 1;
BEGIN FACTOR;
 REPEAT
  CASE CH OF
   ''' : BEGIN GETCHP; STK^[SP]:= - STK^[SP]; END;
   '!' : BEGIN GETCHP; FACTOR; T:=POPVAL; STK^[SP]:=STK^[SP] MOD T; END;
   '+' : BEGIN GETCHP; FACTOR; T:=POPVAL; STK^[SP]:=STK^[SP] + T; END;
   '-' : BEGIN GETCHP; FACTOR; T:=POPVAL; STK^[SP]:=STK^[SP] - T; END;
   '*' : BEGIN GETCHP; FACTOR; T:=POPVAL; STK^[SP]:=STK^[SP] * T; END;
   '/' : BEGIN GETCHP; FACTOR; T:=POPVAL; STK^[SP]:=STK^[SP] / T; END;
   ':' : IF GETCH='#' THEN
          BEGIN
           IF     NEXTCH='C' THEN WRITE(STK^[SP]#CHAR)
           ELSE IF CH    ='I' THEN WRITE(STK^[SP])
           ELSE IF CH    ='S' THEN WRITESTRING
           ELSE IF CH    ='H' THEN HP:=STK^[SP]
           ELSE IF CH    ='R' THEN ERROR(11)  (* NOT IMPLEMENTED *)
           ELSE ERROR(9);
           GETCHP;
          END
         ELSE IF CH=',' THEN BEGIN GETCHP; FACTOR; STORE; END
         ELSE BEGIN VARADDR; STORE; END;
   '>' : IF GETCH='=' THEN
          BEGIN GETCHP; FACTOR; T:=POPVAL; STK^[SP]#BOOLEAN:=STK^[SP]>=T END
         ELSE BEGIN     FACTOR; T:=POPVAL; STK^[SP]#BOOLEAN:=STK^[SP]> T END;
   '<' : IF GETCH='=' THEN
          BEGIN GETCHP; FACTOR; T:=POPVAL; STK^[SP]#BOOLEAN:=STK^[SP]<=T END
         ELSE IF CH='>' THEN
          BEGIN GETCHP; FACTOR; T:=POPVAL; STK^[SP]#BOOLEAN:=STK^[SP]<>T END
         ELSE BEGIN     FACTOR; T:=POPVAL; STK^[SP]#BOOLEAN:=STK^[SP]< T END;
   '=' : BEGIN  GETCHP; FACTOR; T:=POPVAL; STK^[SP]#BOOLEAN:=STK^[SP]= T END;
   ' ','"','#','$','%','@','(',')',',','.',
   '0','1','2','3','4','5','6','7','8','9',';','?','@',
   'A','B','C','D','E','F','G','H','I','J','K','L','M',
   'N','O','P','Q','R','S','T','U','V','W','X','Y','Z',
   '[','¥',']','^','_' : GOTO 1;  (*EXHANGE RETURN*)
  END;
 UNTIL FALSE;
1:END;

BEGIN  (* BODY *)
 NEW(STK);
 RESET(5,'PRG1    ',T); BOS:=1;
 WHILE NOT EOF(5) DO
  BEGIN READ#5(CH); STK^[BOS]#CHAR:=CH; BOS:=BOS+1; END;
 STK^[BOS]:='_'#INTEGER; BOS:=BOS+1; BP:=BOS; PC:=0; HP:=ENDSTK;
 GETCHP; CHKFUN; STK^[BOS]:=TP; SP:=BOS+CNT;
 PUSH(0); WHILE CH<>'_' DO BEGIN POP; EXPR; END;
END.
```

付録 A AIL インタプリタ（前ページからのつづき）

```
?A FACT LOAD AREA?                                  } 広域変数宣言
%0:#C                                               } 文字，文字列，整数の出力
"(ABC):#S
30:A  A+4':#I                                       } 整数入力とカッコ式
(#I)=#I,"(GE);"(LT)):#S
[   "(111:#I)  ]                                    } 文字列定数と関数呼び出し
[["("(222:#I))]]                                      111 222 333が出力される
["([")(333:#I)])]
"(?X?' ( X<=0, 1 ; X*[X-1 FACT] ) ) :FACT           } 階乗関数の定義と実行
[5 FACT]:#I
"(?U CH I?                                          }
"(FILE = ):#S #S:#D                                 } オーバレイプログラムをヒープ領域に
#H-1000:#H:.U                                       } ロードするためのプログラム
0:I                                                 }
(#D:CH=3,:CH:.(.U+I) I+1:I @)                       }
):LOAD
[^AREA LOAD]                                        } オーバレイプログラムのロード
[AREA]  "(EXECUTION OF OVERLAY PROGRAM)             } オーバレイプログラムの実行
[#S]    "(EXECUTION OF INPUT TEXT)                  } 入力文字列の実行
```

付録 B AIL プログラムの例

(昭和 57 年 6 月 2 日受付)
(昭和 57 年 11 月 8 日採録)