マイクロコンピュータ 40—3
（1986. 6. 27）

新しい関数型言語
-LAPLAS-

原田康徳　北村正直
北海道大学工学部工業数理科学講座

スタック処理を利用してリスト処理を高速に実行できる会話型言語ラプラスを開発した。この言語の語順は逆ポーランドである。すなわち、数値、文字列、リスト等のデータをスタックに積み、その状態でこれらを引数とする関数（ワード）を入力して作用させる。その結果がデータとして戻るときは、スタック上に返される。このようなスタックの利用により、関数の呼出しの際の仕事量の軽減させた。これは局所変数の数、再帰処理の回数の低減にもつながり、プログラムの実行速度、高速なリスト処理を可能とした。

またラプラスは強力な三次元トンボグラフィックスを備えている。これをもちいて、分子構造の立体的表示を与える“CHEMI”、空間点群の対称操作をもつ”SYMM”等の教育用の応用パッケージプログラムが完成されている。

A New Functional Language
-LAPLAS-

Yasunori HARADA, Masanao KITAMURA
Faculty of Engineering, Hokkaido University

A new functional language, which utilizes stack processing to realize fast list processing, has been developed. The word order in the LAPLAS program is inverse-polish. Thus any type of data such as numerals, lists and strings are put on the stack, and then a function, which is called an word in LAPLAS, operates on them when its name is typed. The result of the operation is also put on the stack, ready for the next operation. Such a utilization of the stack lessens the overhead jobs when a word is is called in, thus realized fast performance in running the programs. LAPLAS has a powerful 3-demensinal graphics calledTOMBOW-graphics. We have quite a few application packages such as "molecular structures" and "crystal groups"

１：はじめに

今日、多数のコンピュータ言語が社会のいろいろな分野で使用されている。それぞれの言語はその得意とするところを持ち、そこでは有効に使用されている。大学における理工系の研究にはＦＯＲＴＲＡＮが、企業における業務、事務管理にはＣＯＢＯＬが、また近年にはパーソナル・コンピュータの普及に伴ってＢＡＳＩＣが広範囲に用いられている。最近の人口知能の研究の進捗とその応用の新たな展開はＬＩＳＰやＰｒｏｌｏｇ等のいわゆる人口知能（ＡＩ）言語の普及によるところが大きい。それはこれらの言語が人口知能の研究、その応用に適しているからである。ここに紹介する言語“ＬＡＰＬＡＳ”はこれら言語といはれているものと同じカテゴリーに属するものである。

ＬＡＰＬＡＳは最初からマイクロ・コンピュータ上で開発された言語である。大型計算機上で開発され、そこで使用されていた言語を小型機用に改定したものではない。従って、小型計算機上で使用するには他の小型機向きのＡＩ言語と比べて数多くの便利な面を持っている。例えば、最新ヴァージョンのＬＡＰＬＡＳの本体はスクリーン・エディタを含めて約６０Ｋバイトにしかすぎない。またプログラムの実行に際して、非常に少ないメモリーしか使用しない。このことはまたプログラムの実行の高速化にもつながっている。例えば２００個のデータのリストを逆順に並びかえ（reverse)、さらに他の２００個のデータのリストと合わせる(append)プログラムも人間の目には瞬時と思える時間(< 0.5sec)で処理される。

ＬＡＰＬＡＳはLanguage Processor for Listing and Stacking の意味である。この名の示すようにＬＡＰＬＡＳはリスト処理とスタック処理とを巧みに組合せた言語である。この言語はインタラクティヴに使用する言語であるから、変数、定数、関数等をあらかじめ宣言したり、配列関数の収納のためのその宣言などはかえって邪魔になる場合が多いのでこれらはすべて同等に扱って区別していない。すなわち文法は最小限しか持っていない。このことの功罪についていろいろな意見もあるであろう。しかし、この言語の狙いとする領分においてはこれは都合のよい性質である。

ＬＡＰＬＡＳは最初の版はＰＡＳＣＡＬで組まれた。これはＰＡＳＣＡＬがこの様なシステムを組むのに適した言語であることと、方言の間の差が少なく移植が容易であることによる。その後、ＰＡＳＣＡＬによるＬＡＰＬＡＳも改定を重ね、またＣ－言語によるＬＡＰＬＡＳも新たに作成されている。Ｃ－言語を用いた理由はＰＡＳＣＡＬの場合と同じである。現在、ＬＡＰＬＡＳの最新版はversion 2.5 (C) とversion 1.7 (PASCAL)である。この言語は最初はソード社のM68 上で開発され、その後ソードM68mx 上でＣ－言語版が作成された。ＰＡＳＣＡＬ版はCP/M-86 のＯＳやMS-DOSを持つ多くの機種に移植されている。しかし、アニメーション、シミュレーション機能はハードの処理速度の関係もあり、まだ開発機種に限られている。適当なＣ－言語、もしくはアッセンブラー言語を用いると、複数のグラフイック画面を持つ機種にはこの言語をグラフイック機能と共に移植することができる。

ＬＡＰＬＡＳのプログラムは、スタックを使用するために逆ポーランド記法で書かれる。このことはまたインタラクティヴに用いられるこの言語を非常に使い易いものとしている。関数の引数はすべてスタック上に積まれる、その状

態で関数は呼込まれこれらの引数を受けとって結果をスタック上に戻す。これはプログラムの次ぎのステップにそのままの状態で進めることを意味する。例えば、ｓｉｎ（ωｔ＋δ）を計算するには

＞ω　ｔ　＊　δ　＋　ｓｉｎ

と入力すればよい。スタックの状態を把握しているだけで、変数の受け渡しについてプログラム上で考える必要は無い。これはスタック言語ＦＯＲＴＨと全く同じである。ＬＡＰＬＡＳのスタックには、しかし、数値のみならずデータ、文字列、及びそれらのリスト、さらに実行可能なプログラムもリストとして積むことができる。

ＬＡＰＬＡＳは３次元グラフイック機能を備えている。これを我々はトンボ・グラフイックスと呼んでいる。これは一応、言語としてのＬＡＰＬＡＳとは別のものであるが、トンボ・グラフイックスの特徴をＬＡＰＬＡＳは充分に生かしており、またトンボ・グラフイックスを用いた応用プログラムはこの言語の秀れた特徴を示すよいれだいとなっている。従って、トンボ・グラフイックスはＬＡＰＬＡＳの重要な一部と言うことができる。

２：スタックと簡単な計算

ＬＡＰＬＡＳにおける計算はＦＯＲＴＨと同じである。例えば、３＋４の計算は下のようになされる。

＞３　４　＋　？

　７

“＞”の記号はＬＡＰＬＡＳにおけるプロンプトであり、“？”はスクリーンにスタックの一番上のデータを表示させる“ワード”である。３と４がスタックに積まれた状態で“＋”が呼ばれるとこのワードはスタックの上から一番目と二番目の数値データを取り出し、それらを加え、その結果をスタックに戻す。この計算におけるスタックの状態の変化は下のようになる。

入力	３	４	＋	？
スタック		４		
	３	３	７	
	……	……	……	……

その他“－”、“＊”、“／”等の演算、“ｅｘｐ”、“ｌｏｇ”、“ｓｉｎ”、“ｃｏｓ”、“ａｔｎ”や“ｉｎｔ”も逆ポーランド記法によって入力されるだけで、他の言語における対応する演算、関数と全く同じである。

ＬＡＰＬＡＳのスタック操作のワードは、ＦＯＲＴＨと同じ機能のものを、同じ名前で用いている。すなわち、“ｄｕｐ”、“ｄｒｏｐ”、“ｓｗａｐ”、“ｏｖｅｒ”と“ｒｏｔ”のワードを備えている。ｄｕｐはスタックの一番上のデータの複製してその上に重ねるワード、ｄｒｏｐは一番上のデータを取り除くワード、ｓｗａｐは一番上の二つデータを入替えるワード、ｏｖｅｒは二番目のデータを複製して一番上に重ねるワードであり、ｒｏｔは上から三番目のデータを取り出して一番上に重ねるワードである。ＦＯＲＴＨにはこの他にスタック

操作のワードを持っているが、我々はこれらのワードとリスト処理の組合せで、処理速度を落さずに演算、プログラムの実行ができるので、それらは加えなかった。

この言語のシステム・ワードは、最初はできるだけ少なく押えられた。これらのシステム・ワードを用いて必要なワードが組まれ、それらのうち汎用性のあるものはこの言語のローディングに際して一緒にロードされるようにした。その後、新たな機能がシステム・ワードに加えられたが、ここに述べた我々の方針は変っていない。この結果、この言語の本体の大きさはエディターを含めて６０Ｋバイトしかない。

３：論理演算

論理演算ワードには“ａｎｄ”、“ｏｒ”と“ｎｏｔ”が、比較演算ワードには“＝”と“ｍｉｎｕｓｐ”がある。最初の三つはスタック上の論理値の値を考察すれば、ＬＡＰＬＡＳで簡単に定義することができる。最後のワードはスタックの一番上の数値が負数かどうかを判定する。これを用いて二っの数の大小を判定するワードを定義できる。これらについては後に定義の項で例として触れる。

論理演算ワードの場合も、言語の本体を重くするのを避けるため、基本的なワードのみをシステム・ワードとしている。将来、プログラムの実行速度を上げるために幾つかのワードをシステム・ワードとして加えるようにはなるであろう。しかし、現在はそれほど大きなプログラムもなく、また大きな仕事を年頭において組んだ言語ではないので、しばらくはＬＡＰＬＡＳのサイズに大きな変化はないであろう。

４：制御関数

ＬＡＰＬＡＳは“ｉｆ”、“ｗｈｉｌｅ”と“ｅｌｓｅ”の論理制御ワードと、繰返し操作ワード“ｒｅｐｅａｔ”を持っている。これらは

論理値　（リスト１）　（リスト２）　ｉｆ
（リスト１）　（リスト２）　ｗｈｉｌｅ
論理値　（リスト）　ｅｌｓｅ　……

というように持ちいられる。

ｉｆは「最初の論理値がｔｒｕｅのときはリスト１を、ｎｉｌのときはリスト２を実行しなさい」という意味である。ｗｈｉｌｅは「リスト１を実行した結果がｔｒｕｅである間はリスト２を実行しなさい」と読み、ｅｌｓｅは「論理値がｔｒｕｅのときはリストを実行し、ｎｉｌのときはプログラムの次のステップに進みなさい」ということである。

ｒｅｐｅａｔは同じ操作を繰返すときに使われる。例えば、

＞１　２　＊　２　＊　２

は２の三のであるが、これはｒｅｐｅａｔを用いた

＞１　（２　＊）　３　ｒｅｐｅａｔ

同じである。これを用いて７の階乗は

＞１　（ｃｏｕｎｔｅｒ　＊）　７　ｒｅｐｅａｔ

とかかれる。ここで、ｃｏｕｎｔｅｒはｒｅｐｅａｔに附随する関数で操作の回

数を数えている。ここではこれをプログラムに利用しているのである。

5：ワードの定義

ＬＡＰＬＡＳは関数型言語である。言語の本体は大きくないが、基本的な機能はシステム関数として備わっている。これらを用いて新しい関数を定義をして言語の機能を高めていくことができる。

ＬＡＰＬＡＳのにおいては関数、定数、変数等を区別せず、すべてワードと呼んでいる。新しいワードの定義は

```
>´名前　（定義の内容）　.
```

と書かれる。最後の“．”は定義を完成させる大切なワードである。例えば、ある数の三乗を与えるワードをｃｕｂｅという名前で定義するには

```
>´ｃｕｂｅ　（ｄｕｐ　ｄｕｐ　＊　＊）　.
```

とする。変数を含むワードの定義も同じようになされる。例えば、階乗は

```
>´！　（１　（ｃｏｕｎｔｅｒ　＊）　ｒｏｔ　ｒｅｐｅａｔ）　.
```

と書かれる。これはｒｏｔがスタックの上から三番のデータを一番上に持ってくるワードであることを思出せば、先の７の階乗のプログラムと比較すると、これが階乗の正しい定義であることが直ちに分る。これをローカル変数と再帰処理を用いて定義すると、

```
>´！　（（ｎ）　ｖａｒ　ｎ　０　＝　（１）
                （ｎ　ｎ　１　－　！　＊）　ｉｆ）　.
```

ｉｆが呼ばれたとき、ＬＡＰＬＡＳはスタックの一番上の三つのデータを見て判断し、処理をする。従って、同じ処理が実行されるならばプログラムの記法はどんな形でも良い筈である。例えば、スタック操作を利用し、ローカル変数を用いない階乗の定義を書くこともできる。

```
>！　（ｄｕｐ　０　＝　（ｄｒｏｐ　１）
                （ｄｕｐ　１　－　！　＊）　ｉｆ）　.
```

この例は、スタック操作を利用しローカル変数の数を減らすことにより、変数の値のメモリーへの収納、変数の検索、呼びだしの手順を省くことが可能であることを示している。ＬＡＰＬＡＳがプログラム・リストをもスタックに積めるようにしたスタック処理を用いた理由の一つは、このようなプログラムの高速な実行を狙ったからである。

6：リスト処理

リスト処理の基本的なワードは“ｈｅａｄ”と“ｔａｉｌ”である。これらはそれぞれＬＩＳＰの“ＣＡＲ”と“ＣＤＲ”に相当する。“ＣＯＮＳ”に相当するワードはそのまま“ｃｏｎｓ”と呼ぶ。この逆操作“ｓｎｏｃ”は一つのリストを引数としそれをｈｅａｄとｔａｉｌに別けて返すワードである。これらを用いて次のようなリスト処理ワードを定義できる。

```
>´ｇｅｔｎ　（（ｔａｉｌ）　ｓｗａｐ　１　－
:                   ｒｅｐｅａｔ　ｈｅａｄ）　.
>´ｃｏｐｙｎ　（（ｎ）　ｖａｒ
:                   （ｓｎｏｃ）　ｎ　ｒｅｐｅａｔ　ｄｒｏｐ
                    ｎｉｌ　（ｓｎｏｃ）　ｎ　ｒｅｐｅａｔ）　.
```

ｇｅｔｍはリストのｎ番目の要素を取りだすワード、ｃｏｐｙｎは最初からｎ番目の要素までのリストを移し取る働きをする。

７：ｒｅｖｅｒｓｅとａｐｐｅｎｄ

ＬＡＰＬＡＳの特徴をリスト処理について考察するためリストの逆転と二つのリストの結合のワードの定義を調べよう。

ＬＩＳＰの教科書にはしばしば悪い例として引用されているこの二つの関数の定義はＬＡＰＬＡＳでは次のようになる。

```
>´reverse  ((lis)  var  nil
:              lis  nil  =
:              ()
:              (lis  tail  reverse
:               lis  head  rot  cons
:               append)
:              if)  .
>´append  ((lis1  lis2)  var
:              lis1  nil  =
:              (lis2)
:              (lis1  head  lis2  tail
:               lis2  append  cons)
:              if)  .
```

これらの定義において再帰定義に入れ子の中にまたローカル変数が現れる。従って、使用するメモリーの数と処理のステップ数はリストの長さが増えるに連れて指数関数的に増大する。これを避けるには、再帰処理を用いない方法と、ローカル変数を減らす方法とが考えられる。その様な定義の例は

```
>´reverse  ((lis)  var  nil
:              (lis)
:              (lis  head  swap  cons
:               lis  tail  ´lis  let)
:             while)  .
```

ここでｗｈｉｌｅの最初の引数になっている（ｌｉｓ）はこのときは（ｌｉｓ　ｎｉｌ　＝　ｎｏｔ）を意味する。

```
>´append  (over  dup  length
:              (lis2  n)  var
:             n  0  =
:              (lis2)
:              ((dup  head  swap  tail)
:               n  1  -  repeat
:               head  lis2  (cons)
:               n  repeat)  .
```

これらの定義は前の例に比べると幾分冗長になってはいるが、実行速度に関してはは格段に改善されている。前のｒｅｖｅｒｓｅ定義ならばＲＡＭが500kバイトあっても長さ５０のリストでもメモリーオーバーを引起こす。しかし、

あとの定義ならば長さ４００のリストでも１秒以下で処理される。しかしこれらのワードやその他のワードの定義のアルゴリズムを検討すると改良の余地があることが明らかになり、“ｍａｐｓ”というシステム・ワードが考えだされた。これはあるデータのリストと実行可能なプログラムのリスト（関数）を引数としてデータを頭から取りだして関数を作用させた結果をスタックに積んでいく働きをする。これを用いるとｒｅｖｅｒｓｅを非常に短く一行で定義できる。この定義はローカル変数はあらわにはでてこないが本質的には前の定義と同質である。この定義による実行速度は当然改善されている。

ａｐｐｅｎｄも再帰処理を含むが、ローカル変数に頼らない、スタック操作を利用した短い、しかも効率の良い定義を書くことができる。このようにスタック処理は処理する所がまた収納する所と見なすことができ、データの収納と管理の面で非常に効率のよいプログラミングを可能にする。これは、プログラミングのアルゴリズムの改良と言うより、むしろ言語のアルゴリズムの改良と言える。このようなアルゴリズムの改良はプログラム実行の速度を数倍ではなく数十倍高めることがあるように思われる。

ここに挙げた例でも分るように、ＬＡＰＬＡＳの記法は括弧の数が少なくプログラムを非常に分りやすくしている。一方この言語は関数型であるから、すでに開発された関数（プログラム）を名前を呼び込むだけで利用できる。これらの性質はＬＡＰＬＡＳによるソフトの開発効率を極端に高めている。今日、近い将来におけるソフト・技術者の不足が予測されているが、ＬＡＰＬＡＳはその解決の方向の一つを示しているようにおもわる。

８：トンボ・グラフイックス

ＬＡＰＬＡＳは三次元トンボ・グラフイックスを持っている。これはＬＡＰＬＡＳとは独立した思想の元に開発されたグラフイックスである。しかし、タートル・グラフイックスがＬＯＧＯと一体視されているように、これもＬＡＰＬＡＳと切り離して考えられない。トンボ・グラフイックスはＬＡＰＬＡＳの中でその機能を充分に発揮している。ここでは、タートルに代って、トンボが空間を自由、自在に飛回り図形を描く。トンボは現在の姿勢で自分自身を複製し、現在の位置に残していくことができる。このような作業のために、別にトンボのスタックが用意されている。トンボ・スタック上の一番上のトンボが取りさられるとスタックの上にあらわれたトンボが操作の対象となる。

また対称操作で複数のトンボを複製し、それらを同時に同じ動きをさせて対象図形を一度に描かせることができる。このように制限された意味ではあるが、トンボ・グラフイックスは 3-dimensional simultaneous and multi-turtle graphicsであるといえる。

トンボ・グラフイックスはそれだけで一つのトピックスとなる。従ってここではその応用プログラムを幾つか紹介するだけにとどめる。

“ｃｈｅｍｉｃａｌ”：

酸素、水素、窒素、炭素等の基本的な元素の原子の立体もけを描くワードを用意し、その名前を順に入力するだけで分子の立体模型を描くことができるパッケージである。同じ元素でも結合の仕方が異なるときには異なるワードとして登録することが必要である。またこれらの原子のワードを使ってコンピュータ

が自分で判断しながら結晶を描くこともできる。さらに判定条件をパスした結果の手順をワードを並べたリストとして記録し、このリストをプログラムとしてあとで利用することができる。そのときは判定というプロセスがないため描画の早さは元のときに比較して格段には早くなる。

“ｓｙｍｍｅｔｒｙ（結晶群論）”

結合は考慮いない原子の模型と空間点対称操作のワードからなるパッケージである。一度描かれた結晶図形は視点を替えて高速で再現できる。この対称操作は模様などを描くのに利用できる。このパッケージは大学の理工系の結晶学、群論の学習に役にたつ。

“ｒｏｂｏｔ、その他のシミュレーション”

ロボットを空間の中を歩かせ（宇宙遊歩）マウスでその動きを制御する。三次元の動きには３×４の行列の計算が必要である。そのためこれを充分にこなすハードはまだ少ない。しかし、誓い将来ほとんど全ての１６ビットの機械でシミュレーション、アニメーションができるようになるであろう。

９：まとめ

ＬＡＰＬＡＳは機能において、ＬＩＳＰとＦＯＲＴＨとの両方の性格を受けついでいる。ＬＡＰＬＡＳの狙いは効率の良いリスト処理の実現であった。従って、この言語の得意とする分野はリスト処理である。その特徴は：１）メモリー効率が良い．：２）実行が早い．：３）処理系が小さい．：４）対話性に秀れている．と要約される。

この言語はソード社のSORD M68およびM68 MXで開発され、現在は日本ユニバック社のDS 7、東芝のパソピア 16 等ほとんどの１６ビットマイコンに移植されている。ただし、幾つかの機種ではトンボ・グラフイック機能は未だ移されていない。また現時点ではシミュレーション、アニメーション機能はM68 MXの上でのみ可能である。