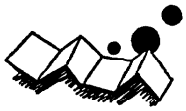
解　説



Ada—米国国防総省新言語†

筧　捷彦††

さまざまな新技術を包含した高級言語のひとつとして，Adaの初版が世に出てから1年が経った．近々その改訂版が公表されるという．Adaについての大まかな紹介をし，改訂版を含め，今後の追跡への手がかりを与えることを目的とする．

1. 経　緯

米国国防総省 (Department of Defence, 以下 DoD と略す) が，組込み型計算機システム*のための新言語問題を取り上げたのは，1974 年のことであった．翌 1975 年に，共通高級言語 (Common High Order Language) 計画が正式に発足し，分科会 (High Order Language Working Group, HOLWG) が活動を開始した．

活動は，DoD 内部における高水準言語への要求のとりまとめと，既存の言語のうち，これらを満たすものがあるかどうかの調査とに始まった．

一方で，DoD は通達を出して，今後開発される国防システムは，DoD が管理する高水準言語に依らなければならないことにした．そして当分の間の認定言語として，つぎのものを与えた．

COBOL FORTRAN TACPOL CSM-2 SPL/1
JOVIAL J3, J73

HOLWG の調査活動から，高級言語の利用増大が経済的利益に結びつくこと，さらに最新の技術を取り入れた統一言語を用意する方が一層の利益を持たらすであろうとの結論が得られた．こうして，新統一言語制定作業が開始された．

その第一段階は，要求仕様書作りにあてられた．DoD 内部に限らず，学界・産業界からも広く意見を求め，何度も改訂作業が行われた．それらは，

STRAWMAN, WOODMAN, TINMAN,
IRONMAN[1]

そして，STEELMAN[2]としてまとめられた．

この要求仕様書に従って，既存言語がそれに該当しうるか否かが 10 指に余るものについて詳細に検討された．その結論は，既存言語で要求仕様を満たすものは存在しない，ということであった．

1977 年，IRONMAN による予備設計が 4 機関に委託された．各機関からの設計書は，その機関名を秘匿するため，GREEN, RED, YELLOW, BLUE の表紙が付けられ，それぞれこの色がその呼称となった．

1978 年 3 月，このうちの GREEN が第 1 候補，RED が第 2 候補として選ばれた．一方で IRONMAN の不備も指摘され，その改訂版である STEELMAN が公表されたのは，1978 年 6 月のことであった．

1 年間の作業の後，1979 年 5 月に GREEN が採択され，Ada の名が付けられ公表された[3,4]．Ada という名は，C. Babbage の計算機のプログラマでもあった Ada Augusta にちなんだものである．

公表されたものに対して，再び広くから，言語事項の問題点報告**や，実際的なプログラムを書いてみることによる試用評価***の報告が集められた．これらの意見は整理・分析され，それに基づいて GREEN 提案を行った CII Honeywell Bull 社が改訂作業を行っている．改訂版は 1980 年 7 月 15 日公表の予定であったが若干公表時期がおくれる見込みである．

2. 要求仕様書の大略

STEELMAN[2]は，言語・翻訳系に対する要求（必須のもの・望まれるもの・可能なら盛り込みたいもの），利用者に関連する要求（許可してならないもの・許容しなければならないもの）を明確に書き分けている．ここでの紹介は，それらの区別を多少ともあいまいにしてしまうことをあらかじめお断りしておく．

設計方針として，一般性を持つこと，信頼性が高いこと，保守性が良いこと（書きやすさより読みやすさを重視すべきことが強調されている），効率が良いこ

† Ada—a new language of DoD by Katsuhiko KAKEHI (Department of Mathematics, Faculty of Science Rikkyo University).
†† 立教大学理学部数学科

* embeded computer system.
** LIR, Language Issue Report.
*** T&E, Test and Evaluation.

と，不要に複雑化しないこと，具現*しやすいこと，機機独立**であること（機械に依存することも表現できることを要求するがそういう部分は明確に他から分離されていること），そして言語定義が完全であることが挙げられている．

構文に関する条件としては，プログラムを表現するのに使用できる文字集合を明確に規定していること（英大文字・数字と限定された特殊文字の計55文字），構文の拡張性を禁じていること（いわゆる拡張可能言語***は読みやすさという点から捨てられている），行という単位が実質的に区切りとなるべきこと（字句要素****が行をまたがってはならず，また注釈すらも行をまたがってはならない）等が目を引く．

言語全体は，強い意味で型付け*****されていることが要求されている．つまり，構文要素はそれぞれ（翻訳時に決まる）型が定まっていなければならない．その型は，いくつかの基本的なものから始まって，Pascal風の各種の構造を持ったものを利用者が定義できるものであることが要求されている．

式の評価において，副作用の順序まで規定してはならないこと，いわゆる宣言について有効範囲が明確であること等を含め，従来の言語にも備っていた部分についての条件は，Pascalのそれを彷彿とさせるものである．

これらを越えるものとして挙げられている条件にはつぎのものがある．

いわゆる密封機能 (encapsulation) を持たねばならないこと，手続きや関数については，その引数の型などが違えば，同じ名前でいくつものものを代表させることができること（これをオーバローディング (overloading) と呼んでいる）．並行処理を記述する能力を持つこと，例外的な状況に達したとき安全にプログラムの動作を終了するための機構（例外処理機構 (exception handling)）を含んでいること．データ等をその機械の上でどう表現するかといった指示が書けること，分割翻訳 (separate compilation) ができること，さらに，手続き等を，任意の型のデータに対して一般的に用意しておける機構（汎用体(generic unit)）を含んでいること．

以上のほかに，入出力機器を直接に駆動できる入出力命令・利用者がふつうに用いる入出力等も用意しておくべきことも明記されている．

さらに，言語を囲む環境にも言及し，明確な仕様書・規格の制定・翻訳系の完全性・各種プログラム開発用道具の存在等までが要求されている．

3. 言語仕様の大略

言語仕様は，STEELMAN[2]の要求にほとんど忠実に従ったものである．表記上の若干の違いをのぞけば，Pascalが持つ能力範囲内のことがらは，ほとんどPascalと同じだと言っても，そう間違っていない．もちろん，Pascalの標準化において先ず問題となった配列型の扱いが，動的に改められている等の変化はある．しかし，ここでは，Pascalを越える部分の仕様について紹介することにとどめておく．

3.1 オーバローディングと汎用体

強く型付けされている言語の特徴を一層強力にしようとすれば，つぎのようなことも望まれよう．

例えば，FORTRAN で実数型として十把一からげに計算しているデータも，よく考えると，あるものは「速度」，またあるものは「距離」，またあるものは「時間」であったりする．型を明確に区別する，という立場からすれば，これらはそれぞれ別の型とすべきであろう．

```
V, V1 : VELOCITY;
X, X1 : DISTANCE;
T, T1 : TIME;
```

こうしておけば，うっかりして V:=X; などと代入文を書けば，翻訳系が型の違いを検出してくれるので，誤りを検出できる．

しかしながら，型が違うものの，計算のそれは，いわゆる浮動小数点型 FLOAT と同様であってほしい．例えば，定数は，どの型についても 0.0 と書きたい．つまり，定数は多義にならざるを得ない．これを定数のオーバローディングという．Ada では，

```
TYPE VELOCITY IS NEW FLOAT;
```

というふうに，もとの型から新しい型を導出することを許し，それに伴って定数等のオーバローディングが自動的に生じる．演算記号のオーバローディングも自動的に生じ，本来 FLOAT 型同志の演算記号として定義されていた + も，そのまま V+V1 とか T+T1 として使えるようになる．

一方，（速度）*（時間）が（距離）を与えるようにも定義することができる．

---

* implementation.
** machine independent.
*** extendible language.
**** lexical unit.
***** strongly typed.

```
FUNCTION "*"(X: VELOCITY;
  Y: TIME) RETURN DISTANCE IS
BEGIN
  RETURN DISTANCE (FLOAT (X)
  *FLOAT (Y));
END;
```

つまり，結果は速度・時間をそのまま浮動小数点に型変換した上で乗じた値を，再び距離に型変換したものである，として定義してやればよい．

Ada では，演算記号を関数呼出しの省略記号と見る．かくして，一般に手続き・関数に対しても，オーバローディングが許されているのである．

このオーバローディングを許容しているため，翻訳作業は従来のものとは違った困難を伴うことになる[5]．

強い型付けのもとで，汎用の手続きを準備するのは容易でない．例えば，2変数の値を入れ換える手続き SWAP を準備してみよう．

```
PROCEDURE SWAP (X, Y : IN OUT T);
  W: T;
BEGIN
  W:=X; X:=Y; Y:=X;
END;
```

Tは，代入の許される型ならどれでもよいことにしたい．ところが利用者はいくらでも新しい構造を持った型を定義できるから，それを予測することはできない．そこで，この手続き全体に

```
GENERIC (TYPE T)
```

という句を前置することで，汎用化することを認めるのである．利用者は，必要に応じて

```
PROCEDURE SWAP_INT
        IS NEW SWAP (INTEGER);
```

などと具体化して利用する．

注意すべきことは，Adaの手続き・関数への引数として，手続きや関数を渡せないことである．これらを渡したい場合にも，汎用体機能を通じて（汎用化に際しては，型のほかにこれらもパラメタとできる）実現することになっている．

3.2 モジュールと分割翻訳

いわゆるモジュール*化，あるいは密封化といったことがらは，現代的プログラミングのひとつの基本的手法となってきた．Adaは，これを実用的な言語に持ち込んだ最初の例といってよかろう．

Adaにおけるモジュールには，密封化だけを目的とするもの（package）と，並行動作の単位ともなるもの（task）と2種類がある．

モジュールは，外部からの利用を許すものの仕様を書くための仕様部**と，その内容構造を書くための本体部***とからなる．

仕様部には，外部からの利用を認める変数・配列あるいは手続き等を書く．手続きについては，その引数の名や型（関数については，さらにその結果の型）のみを書く．この点からは，いわゆる抽象データ型****という考え方までは取り入れていないといえる．まだ抽象データ型というものは実用化段階にない，との判断からであろう．

本体部には，手続きの本体，あるいはそれらで共通に利用する変数・局所的な手続き等を書く．これらのものは，外部から直接には利用できない．いわば，そのモジュールに占有的（own）なものである．

モジュール内で新たに導入した型も，仕様部に定義を与えておけば，外部から利用できる．このとき，密封化を強力におし進めるために，その型の定義の精細を外部から隠すこともできるようになっている．

本体部の最後には，占有的な変数等の初期設定のための文を並べることもできる．

モジュールは，また，手続きとともに翻訳の際の単位ともなる．翻訳は，これらを単位として分割して行うことができるが，型の整合性等の検査は，一括して翻訳した場合と同様に厳密に行われる．これを分割翻訳という．

分割翻訳のためには，各単位の翻訳の順序が部分的に規制されることがある．これらの制御は，翻訳系を含めた処理系が管理しなければならない．

3.3 並行処理機能

並行処理の単位は，taskと呼ばれるモジュールである．（もちろん，いわゆる主プログラムも，ひとつのtaskとして取り扱われる．）

並行処理機能の主体は，並行処理単位間での通信・同期を何によって行うかにある．Adaでは，これをランデブー（rendezvous）と呼ぶ機構によっている．

各単位には，いくつかの入口*****を置くことができる．他の単位と通信（同期）するには，その単位の入口を指定して（必要ならば引数を伴って）呼出しを行

---

* module.
** specification.
*** body.
**** abstract data type.
***** entry.

う．一方，各単位の中では，自分の入口に対して，呼出しの受付けを行うことができる．

入口への呼出しと，受付けとがともに揃ったとき，ランデブーが行われる．すなわち，受付けが実際に履行され（引数を用いた処理もその時行われ）る．それが終わると，始めて，呼び出しを行ったものと受け付けたものとがそれぞれ独立に動作を再開する．

呼出しを行っても受け付けられないものは，その入口のところに待ち行列を作る．一方，受付けに達したものの，呼出しが行われていなければ，その単位は実際の呼出しが行われるまで待ちにはいる．

いくつかの入口に対して，呼出しがかかっているものがあれば，そのひとつをランダムに選んで受け付けるための文も用意されている．この時，それぞれの入口に対して受付けのための条件を付加することもできるし，一定時間待ってもどの入口への呼出しも起きない場合にどうするかの指定も書ける．

Adaでは，言語仕様上，共通変数を用いての通信も可能である．ただ，ランデブーの機能は十分に強力であるので，多くのプログラムにおいて，同期・通信のために共通変数を介する必要はないはずである，との立場に立っている．

3.4 細部記述機能

可搬性（portability）の追求とは相矛盾するが，組込み型計算機のシステム・プログラム開発用ともなれば，当該の計算機の機構を十分に活用する必要も生じる．このため，こうした細部にわたる記述も書けるようになっている．しかし，これを野放しにしたのでは困るので，これらは決められた箇所にしか書けないようになっている．

これらの機能のひとつは，各データの記憶上での表現を指示するものである．たとえば，各型のデータについて，割り付けるべき長さを指定したり，列挙型*データの内部番号の指示を与えたり，記録型**のデータの各欄***の配置を指示したりする．

またひとつは，番地の指定を行うものである．例えば，変数の割り付けられるべき番地を指示する．（多くのマイクロ・コンピュータでは，入出力が記憶上の番地に割り付けられている．したがって変数をそれに重ねることで，入出力が記述できる．）あるいは手続き等の番地を指定したり，並行処理での入口を番地に結び付けたりできる．（マイクロ・コンピュータでの割込み処理は，その処理プログラムをひとつの並行処理単位としておき，入口を割込みベクトルと重ねておくことで実現される．つまり，割込みを，その入口への呼出しと解するのである．）

また，対象計算機の仕様について，記憶の単位長・総量等の値を引き出すための機能も準備されている．

その他にも，翻訳系への指示がいろいろとできるようになっている．これらは，pragma と呼ばれる．たとえば，手続き・関数の本体を呼出しの場所ごとに埋め込んで翻訳することの指示も，この pragma のひとつである．

* enumeration type.
** record type.
*** component.

4. 展 望

Adaは，その第1段階を終えようとしている．つまり言語仕様の公表と，それに対するさまざまな反響をふまえて，最終仕様が確定される段階にある．

米国内では，さまざまな講習会もひらかれているしまた試用評価のためのテスト用コンパイラも ARPA ネットワークを通して利用可能であるという．日本でも，Ada についていくつかの紹介[6~9]・検討[10~11]さらには翻訳系[12]の試作が行われている．

第2段階は，実用的な翻訳系の作成ということになろう．すでに DoD は，翻訳系に対してそれが Ada のそれであると言ってもよいかどうかを認定するに足りるテスト・プログラム集作りを Softech 社に対して発注している．予定では，明 1981 年には，公表されるはずである．

翻訳系についても，カーネギー・メロン大学を始めとしていくつかの大学で研究が進められている．DoD の陸軍・空軍もそれぞれ翻訳系をすでに発注しており，1981 年末から 1983 年末にかけて，完成する予定となっている．

一方，標準化に関しても意を注いでおり，すでに ISO/TC 97/SC 5 の PLIC (Programming Language for Industrial Computer) のひとつとして，米国から Ada が候補として提案されている．

P. Wegner[13),14)]のように，1995年には，Ada が，FORTRANを陵駕するであろうと予想する者もいる．この予想が当るか否かはわからないが，1980年代において，良きにつけ悪しきにつけ Ada がプログラミング言語についてのひとつの中心話題となることはまちがいない．

DoD は，同時に Ada を含むプログラム開発環境についても，言語の場合と同様の手順で準備を進めてい

る．その要求仕様書として，1980年2月に STONEMAN[15]が公表されている．これに基づいて，陸軍は従来のオペレーティング・システムの上で使えるものを，空軍はさらに統合的なものをそれぞれ企画しており，今年のうちにも，発注に至る予定である．

参考文献

1) IRONMAN Language Requirements, SIGPLAN Notices (1978-12).
2) STEELMAN: Requirement for High Order Computer Programming Languages, DoD (1978-6).
3) Preliminary Report on the Ada Programming Language, SIGPLAN Notices (1979-6).
4) Rationale for the design of the Ada Programming Language, SIGPLAN Notices (1979-6).
5) 徳田雄洋：Ada実現に関する問題点について，情報処理 (1979-3).
6) 上條史彦：プログラミング言語への期待——米国国防省の HOL プロジェクトについて——，情報処理 (1978-3).
7) 安村通晃：米国国防省 (DoD) 規格による言語，情報処理 (1979-3).
8) 大野侑郎：Ada——米国国防総省の新言語——，**bit** (1979-7).
9) 新しいプログラミング言語 Ada, TSINKY, **bit** (1980-2).
10) 工業用計算機システムのソフトウェアに関する調査報告書——Ada言語に関する調査——，日本電子工業振興協会 (1980-3).
11) 市場創出型プロジェクト——システム記述高水準言語——，共同システム開発(株) (1980-3).
12) 近山 隆：プログラミング言語 Ada 処理系の試作，第 21 回プログラミング・シンポジウム予稿集 (1980-1).
13) Wegner, P.: Programming with Ada—an Introduction by Means of Graduated Examples, Prentice-Hall (1980-1).
14) Wegner, P.: The Ada Language and Environment, SIGSOFT (1980-4).
15) STONEMAN Requirements for Ada Programming Support Environments, DoD (1980-2).

(昭和55年7月21日受付)