報　告

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——

上　條　史　彦**

この 10 年間というもの，国防関係ではコンピュータを用いたシステムが増加の一途をたどっている．その多くはいわゆるエンベデッド・コンピュータ・システム (ECS) であって，ECS に対する 1974 会計年度の投資額 20 億ドルの 70% がソフトウェア関連に投入されているという．ECS は業務上国防省関係で電子・電機機械系に用いられている用語で，本来情報処理が主目的でない系の一部をなすことが多く，全体の系の設計・取得・運用に必要であり，かつ情報・制御信号・コンピュータ用データなどを出力する形のコンピュータ・システムを指すとされている[1]．

ECS におけるソフトウェアの問題点は，ハードウェアの 2 倍以上のコストを要するにもかかわらず，信頼性が低く，かつ保守が困難であることである．外部から見ると，「ソフトウェア問題の原因」と「解決できぬ理由」が明らかでない．10 年前にはソフトウェアのような新しい技術は若い新時代の技術者に全てをまかせるのが最高の指導方針だといえば，それが受入れられる環境にあった．今日では悪しき指導方針と解決のための技術的努力の欠除の 2 点をもって説明とせざるを得ない．国防省は 1974 年 12 月に Assistant Secretaries of Defense と Director, Defense Research and Engineering 両者の共同プロジェクトとして Secretary of Defense/Service Weapon System Software Management Steering Committee を設立しこの問題に挑戦することとした．

新プロジェクトの目的は

規則の確立，厳重な施行
管理能力の向上
費用の統制
スケジュールの管理
品質の向上

といった 5 項目である．

プロジェクトで取上げられている技術的な作業のなかで，**一番の目玉は ECS のための新しいコンピュータ言語の制定である．1975 年 1 月，国防省はプロジェクトに付属した形で言語のためのワークグループ HOLWG (DOD High Order Language Working Group) を作り，新しい言語のための要請項目を定めることとした．HOLWG は 1977 年 4 月に最終的な要請項目集 "IRONMAN" および新言語の基礎の候補**としてあげられた 23 種の言語との比較結果をとりまとめて公表した[2]．この報告は HOLWG の資料の一部を大づかみにまとめたものである．

1. 背　景

高水準言語といえば COBOL と FORTRAN ということになり，両者の標準化が進められているものの，実現する上での個々のプロセッサの仕様の差は大きく，かつ多様でありすぎる． ECS の分野に利用範囲を限っても，保守・訓練・互換性などの面での不便さは測り知れないものがある．混乱状態を脱しようとして陸軍は "Implementation Languages for Real-Time Systems" 計画，空軍は "High Order Language Standardization for the Air Force" 計画をそれぞれ発足させるといった具合に，対応側も統一を欠いていた．加えて英国防省の共通言語 CORAL-66 の成功も一つの契機となって Institute for Defense Analysis による opinion survey が行われるに至った．協力を得られたのは国防省関係 64 機関，産業界 16 社，大学・研究機関 13 団体であった．

最初にまとめられた要請項目は "STRAWMAN" と呼ばれ例示を主とした予備的な形だったが，これをもとに諸方面の意見を徴して，"WOODENMAN" 更に "TINMAN" ができあがった．最終版は "IRON-

* A New Programming Language——HOL Project of DOD by Fumihiko KAMIJO (Information-Technology Promotion Agency)

** 情報処理振興事業協会

MAN" であって，"TINMAN" にあった冗長な説明を除去し，多少の訂正を加えた形である[3]．

"IRONMAN" は 1977 年 1 月に公表されたが，さらに 4 月までの 3 カ月間に類似言語との比較検討が進められた[4]．新年度からは試作の段階に入る予定だという．

2. 言語設計の一般的な目標

2.1 コスト

新しい言語を求めるに至った基本的な問題はシステム (ECS) の開発・運用面におけるコストを節約することにある．前述のようにハードウェアの費用を節減することに比べると，ソフトウェア側にはいろいろと節約の可能性が秘められていると考えてよかろう．ハードウェアについては国防省はコンピュータの需要全体に占める比率が微々たるものであるので，一般的な性能／価格比の上昇していく流れに頼らざるを得ない．ところがソフトウェアの世界では超大口の発注者であるばかりか，基礎的なコンピュータ・サイエンスや技術に対する最大の援助者の地位にある．したがってソフトウェア技術の現状に影響を与え，何らかの変化を始動させられるというふうに考えたい．管理面での新風を吹込もうとする努力は既に始められている．具体化されている施策としては

交換用ソフトウェア・ライブラリの設置
ソフトウェア生産用ツールの開発
ツール類の国費による配布
保守容易化のため高水準言語の採用
利用できる高水準言語の限定

などがあげられる．

経験から考えると，ソフトウェアは過去に用いられたプロセスの繰返しだということができる．中には新しいアイデアもあるが，大部分は既存である．ところがシステム作りは必ず，全く新規に考えるところから出発する．加えて高水準言語といえども互換性は無きに等しい．しかも新種の言語や新型のプロセッサを作るのがお好きである．結局のところ開発（製造）面において強力な環境作りが行われ難くなってしまう．

言語やプロセッサ上での互換性のみならず，ハードウェアにも互換性は期待できない．事務処理用プログラム程度ならば，ソフトウェア的にプログラムを変換しようとするプロジェクトはあるものの，ECS に応用するのには技術的な困難が多すぎる．

結論は 100% 自己開発にかかる共通言語の開発であって，それによって上述の問題も解決し，かつ管理者側の自由もとりもどせるというものであろう．

2.2 納期

ソフトウェアは本質的に低い見積りに泣く特質を持っている．システムの中で重要な部分を占めるにもかかわらず，物理的には無に等しい．動力も不要，強度試験も不可，必要な材料もない．たかだか磁気テープにビットが立っている程度だ．部分的変更も極めて容易，したがって制作に時間がかかるとは考え難い．変更容易な一面，正確さを保証すること極めて困難である．以上の要因からプロジェクトのスケジューリングは楽観的に過ぎ，ソフトウェアができ上る前にハードウェアが完成してしまうという変な事態が起る．このような場合の損失は，たんにソフトウェア工数の超過分にとどまらず，全システムの稼働遅れという大きなものになってしまう．といっても現在ではソフトウェアの開発スケジュールの見積りは類似プロジェクトとの比較と，経験値に頼るしかない．こうした制作環境に安定化因子となる可能性があるものとして言語があるといえよう．

言語の効用はコーディング時間の短縮にあらわれる．高水準言語はコーディング負荷を減らすのみでなく，ある種の機能を利用するようにすればその効果を増進できる．もっともコーディングの比重はプロジェクト全体の 10〜15% である．デバッグ，検査の工程はこれに反して 50% にも上ることがあるので，プログラムの書易さに較べると，信頼性の方がはるかに重要であることに注目すべきであろう．

仕様の決定と設計の工程も工数のかかる部分なので，工程を簡易化するモジュラ設計なども注目に値する．なかでもプログラム（モジュール）の再利用の可能性を探ることは，それについての全工程を省略できるが故に全ての議論に優先して配慮すべき問題である．過去の蓄積を，ライブラリ方式や，経験活用といった方法あるいは自動コーディング・ツール等を通じて引出す方策が，スケジュール遅れを無くすための最良の知恵といってよかろう．

可搬性の問題は，結局言語の可搬性ということにつきる．再利用は必ずしもシステム・コンセプトや流れ図に限る必要はなく，サブプログラムやルーチンがその対象となるからである．再利用の問題点は過去の蓄積の利用可能性を受入れないところであって，そこに今日のハードウェア技術とソフトウェア技術の違いをみることができる．

2.3 信 頼 性

国防省関係の特色はソフトウェアに信頼性を強く求める点にある．システム的には極度に複雑で，実験は不可能，一二の人命にかかわるといった程度でなく，何百万もの人々に関係するという独特な状況にある．これは工程中の検証・承認・検定などのプロセスや，代替方式・保護手段に反映されている．信頼性がコスト上昇の一因をなしていることは明らかであるが，必ずしも計算できる形にはなっていない．しかし言語を考えるにあたって，信頼性を高めるものであることは当然にして，かつ重要な条件である．

共通言語の制定によって，既に正しさが証明されたコードのブロックを転用できれば，信頼性向上への大きな手懸りとなる．今のところ正しさを証明する技術は未発達といわざるを得ない．コードが意図された通りの働きをし，かつそれ以外のことはしないことを約束するような言語があれば国防省は何をおいてもその実現に力を尽すであろう．残念ながら当面できることはそうした方向への努力に援助の手をさしのべることぐらいのようである．

信頼性を側面から高める言語機能として単純性・一意性があり，かつ“読み易さ”がある．最近提案されているプログラミング手法のなかにはよりよい，かつ信頼度の高いプログラムを作りだせることが判っているものがある．言語の中にそういった手法を取り入れるべきである．なかでも管理手法は重要である．

2.4 保 守 性

ソフトウェア関係の費用の中で最大といえるのは保守のためのものである．規模が大きくなり，複雑化し，かつ他のシステムと絡み合うようになると，システムは長命化せざるを得ない．20年にものぼる長命システムが出てくる．長い年月の間には敵の脅威のあり方，担当者のものの考え方，さらにはハードウェアの一部といった基本的な部分の変更が起る．当然にして保守は継続開発・改良といった形をとることになる．

現在あるデータでは正確なところは判らないが，一説によれば保守費用は開発費の半額に達するといわれている．大型システムがまだ開発段階にある場合が多いので，過去の取組み易かったシステムの経験からの議論は現実を歪めている．システム開発が成熟期に入れば，あるいはこの比率は逆転するかも知れないのである．過去のシステムの保守費と運用費を合計すると，開発・取得費の数倍に達している例が多い．

保守の時期に入るとプログラムについての責任が，受託者から政府へ，またその逆の組合せになって，転転と移動する．移動の度に知識が失われてゆく．プログラムが読み易く，あいまいさが無いほど，移転される知識の量は大きい．保守の段階では主なプログラムの可搬性はそれ程問題にならない．しかし保守及び支援作業用のツールの可搬性は重要である．というのは主システムのハードウェアの変動は比較的少ないが，支援システムのハードウェアの寿命はずっと短いからである．同様にして人もある意味ではツールである．使用する言語の種類が少なければ少ないほど保守作業の効率と信頼性が良くなる．また文書化の良否は決定的な影響をもたらす．高水準言語の利点は，それで書かれたプログラム自身で文書の一部たり得ることである．モジュール化技法，構造化技法はそれを更に一歩進めたものといえる．この分野での新しい技術は大いに利用すべきである．

2.5 教育・訓練

教育の効果は数量化し難い．プログラマにとって既成のコースは必要とされるものの極一部にすぎない．残りは OJT とか経験と呼ばれるものである．科学技術分野を例にとると，訓練はまことに不適切か，全くないような状態にある．プログラマ間の生産性のばらつきの原因は，あるいは訓練の良否にあるのかも知れない．生産性の指標より，プログラムの良否からくる運用費の差異を考えた方が，この場合適切な指標であろう．言語の種類を減らすことは，訓練の負荷の軽減につながる．

優秀なプログラマにいわせれば，言語の変更などは数日の教育と実習でこと足りるということになろう．しかしわれわれの調査によれば，部内・委託先を含めて，そうした優秀なプログラマの数はわずかである．プログラマの平均像は，とかく一つの言語の域を出ず，しかもそれすら十分使いこなせないし，ツール類にも明るくないのである．

訓練に伴う問題としては，構造が一意的で単純な言語が望ましく，フェイルソフトであれば理想的ということができる．

2.6 可 搬 性

高水準言語の利点といわれるものは2つある．その1つは“マシン・インデペンデンス”，すなわち可搬性である．プログラムを汎用的に書けば，コンパイラを通じていくつものハードウェアに使用できるという性質はいろいろな利点を生む．異なるハードウェアを使用しているプロジェクト間で既開発品の融通ができ

る．同じシステムで寿命が長期に及ぶ時，ハードウェアの交換が容易になる．2点ともまことに明らかな利点である．

言語に対する可搬性の要請にはいろいろな例がある．SHAREの初期には，実際問題として対象になる大型計算機はIBM 704しかなかったので，マシン・ランゲージでもある程度インターフェースを守っていればプログラムの交換が成立し得たし，事実かなりの交換が行われた．当時はそれでよかったのである．今日ではハードウェアの種類が多いので，また別個の技術が必要となる．高水準言語による機械からの独立性はある程度は存在するが，必然的に独立性が得られないわけではない．良い言語に効率のよいコンパイラを得てかつ異種言語によるプログラムが適当なリンク機構で結合可能たるべきであるが，現実はそうではない．全ての言語と機械の組合せに良いコンパイラを用意することはできないし，そうそう多種類の言語をプログラマに教え込むことも無理である．ソフトウェア全般についてマシン・インデペンデンスを持たせるには最少種類の汎用的構造を有する高水準言語を設定し，ツール類やコンパイラを多くの機種に準備して充分なサポートを与えるに限る．コンパイラが高水準言語で書かれていればなお良い．

いままでは高水準言語といえどもインプリメンテーションに互換性を欠くことがままあり，可搬性が失われてしまった．言語自身に不完全な，あいまいな定義が含まれていたので，インプリメンテーションにおける解釈の差が生じるのをおさえることができなかったのである．加えてインプリメンタは自分だけの判断で言語を拡張したり，あるいはやり難い部分，さしあたって不要な部分を除いてサブセット化したりしている．インプリメンテーションの差からくる方言を数えればいくらでも例がある．共通言語はこの世界では笑いごとでしかない．しかも相違点が表面にでていないため後になって問題を起すケースもある．可搬性そのものに病的懐疑論がでるゆえんである．可搬性を実現するためには厳格であいまいさのない言語仕様を定めることのみならず，サブセット化，スーパーセット化を禁止し，かつコンパイラを徹底的に検定する必要がある．その上で検定済のコンパイラを多くの機種に広く使える状態にすべきである．

可搬性実現の効果は大である．現在のように大勢のプログラマにやりなおしの仕事をさせなくてよくなる．開発プロジェクトごとにみると本当に新しい部分は少ないので，過去の蓄積を活かすことによって，コスト，納期，信頼性といった課題を大幅に改善できる．もっとも今までのところプログラム・ライブラリを利用する形の方法は決して成功しているわけではない．共通言語はこうした目的を達成するための一つのステップにすぎないかも知れないが，それは真に重要な1ステップということができよう．

2.7 読み易さと書き易さ

高水準言語はマシン語に較べると自然言語に近いので読み易くて書き易いという点が利点とされる．この点にはしばしば行き過ぎがあり，論者の唱えるところを信じると要請項目を英語風に記述すれば，コンピュータなぞ知らなくてもプログラムができると思われがちである．過去の歴史を振返る限りこれは正しくない．自然語的として宣伝された言語は今になってみるとみな一部の専門家しか使わないものになってしまっている．国防省の場合には世帯が大きく，ソフトウェア要員は専門員化するたてまえとなっているので，専門化をさまたげるものはない．小さな企業の場合には専任プログラマをおくとそろばんが合わない場合があるだろうが，国防省のおかれた環境は正反対であることをお断りしておく．

ここでは読み易さ，書き易さは専門家にとっての問題として考える．自然語利用の問題は国防省の中では全く異質であって，クエリイ言語 (query language) とか，HOL プロジェクトの対象外のアプリケーションに限られる．

読み易さは明らかに書き易さに優先する．プログラムは一度“書かれる”と，その後検証，変更などのために何年もの期間にわたって多くの人によって“読まれる”運命にある．兵器システムのソフトならずとも科学技術計算やシミュレーションのプログラムは長命の傾向があるからである．大学のように一人の人がたかだか数カ月の寿命のプログラムを量産する環境では，この条件は自ずから違ったものとなろう．

“書き易さ”問題は無益というわけではなくて，他の手段，たとえばインテリジェント・ターミナル，プリプロセサ，会話式システムなどで対処し得る．したがって言語の評価にあたっては，“読み易さ”を優先すべきである．

2.8 処理効率

効率，その中には処理速度・メモリ占有量すなわち定量的因子が含まれ，アセンブリ言語と高水準言語を比較する定形的手続きとなっているが，いずれにせよ

言語以前の問題ということができる．どちらかというと先入感で処理されてしまい，きちんとした考察をする手間を省きがちである．技術の進歩にともないこの問題の重要性は軽くなっているにかかわらず，感情的な議論に支配されるユーザが多い．

数年前まで問題とされた形の効率問題はもはやないといってよい．たとえばメモリ効率は，昔はメモリの費用がシステム・コストの大部分を占めていたので，他の要件を犠牲にする理由であった．最近ではソフトウェア，なかでもソフトウェアの保守費がハードウェア全体より重要となっている．いまや効率のロスを補う程度のメモリのコスト差は“月”で数えられる期間に発生してしまうし，ましてマシンコードでプログラムを作ろうとすると，高水準言語使用によるプログラミング期間との時間的差が前述の差よりはるかに大といった事態になる．実時間システムはマシン語で作らなければ処理速度に問題がでる，という“常識”も今やあたらない．そうすることによって将来背負い込む問題は，費用に換算するとプロセッサのランクを上げるよりは遥かに高い筈である．

高水準言語は正しいが，必ずしもシステムを取得する時の実態を映しているわけではない．ハードウェアの構成が定まるのは開発工程の最初のうちで，ソフトウェアはそれに合せて作られる，という長い間の習慣がある．これにシステムの寿命が終るまでの間の費用を無視するという状態が加わると，プロジェクト・マネジャーの考えることは効率一点張りになってしまう．システムライフの末期にも似た現象が起きる．定められた構成のハードウェアで処理量を増そうとするからである．

合理的な効率の尺度は定めがたい．10% の差といった単位の話としてみよう．最上級のプログラマの間でもこの程度（あるいはそれ以上）の差がつくのはあたり前なので，高水準言語とアセンブリ言語の効率差は簡単には議論できない．ところが，きちんと利害関係を計算すれば 30% 程度の効率差は許せる場合が多いのである．50% の差ということになると，効率を云々する必要がある場合には，まず高水準言語は採用できない．統計的に考えると，プログラムのロジックに基づく効率の差の方がこれより大きいことは明らかなので，30%～50% という基準すら厳しすぎると思われる．しかし現在ではこれより悪い基準を考えるよりも，技術的に高水準言語の効率増進を考えた方が良い．重要なプロジェクトでは高水準言語によるコーディングを手でコンパイルしたものと，初めからアセンブリ言語で書いたものとでベンチマークを行うが，高水準言語が勝ったケースが今でもいくつもあるという事実を忘れてはならない．現在ではまだ特別な場合といっても良いかも知れないが，ロジックが大規模になればなる程，高水準言語の方が最適化し易いという事実からして，システムが大規模化しつつあることを考え合せると，その優劣は明らかであろう．

現存する高水準言語の問題点はインプリメンテーションの質である．とくに特定のプロジェクトや短期間・節約型の条件の下で作られた言語では顕著である．10倍・100倍といった水準での投資が行える場合に較べて，悪条件下のものが質が劣るのはやむをえまい．良質のインプリメンテーションに恵まれたものは，平均的プログラマより良い質のコードを作り出し，かつ最適化においても優れている．

コードの質の問題は，共通性の条件と並んで，言語に対する管理ないし統制への魅力の源となっている．国防省のプログラムのような厳しい条件の下での実時間システムは，ハードウェアとしては大型のものが，多少の例外を除けば，一般的対象であり，技術水準もそれに合されることとなる．従ってクロスコンパイラは対象機種が小型機の場合には，一つの前提と考えてよい．小型機のプログラムの効率を向上する方法として，まだ一般化されていないとはいえ，是非採用すべき技術である．これに対するものとしてサブセット化があるが，サブセットから得られる情報は十分な効率向上は困難である．

2.9 アクセプタビリティ（受入れられ易さ）

いうまでもなく，言語は使われなければ意味がない．受入れられる条件にはいろいろと有形無形の因子がある．コンパイラが広く受入れられるには，例えば自己拡張性などが重要となろう．

既知の言語との類似性も見落せない．国防省で定めている書式も考慮されよう．今のところ現行の言語との間の変換については考えていない．しかし何かのきっかけで取上げられる可能性はある．

3. 要請の要約

3.1 データと型

（1） データの型は翻訳時に定め，実行時には不変とする．

（2） 整数，固定小数点数，浮動小数点数，論理値，文字，配列およびレコードとする．

(3) 浮動小数点の変換や演算では精度を指定できるものとする.
(4) 固定小数点数は範囲，小数の桁数を指定できること.
(5) 文字の大小の順は指示可能とする.
(6) 配列の下限は指定値に固定，上限は動的に変動させる.
(7) バリアント・レコード（レコード (type) の変型）は実行時に別の扱いをする.

3.2 作用素

(1) データ型に代入・参照操作を行えること.
(2) 全てのデータの型に対し同値操作 (equivalence op.) が行えること.
(3) 数値型および数値化しうるデータに対しリレーショナル演算が可能であること.
(4) 算術演算子は +, −, *, /, ÷, ↑, −(符号反転) とする.
(5) 最小有効数字までの切捨て，丸め操作.
(6) 論理演算子は and, or, not, xor とし，簡約処理を含める.
(7) 複合 (composite) データ型にむじゅんしない形での直接的代入を許す.
(8) 型の変換は暗黙には行わない.
(9) 数値の範囲の変換は行わないこととし，範囲の検査が行えないようにする.
(10) 入出力操作にはファイル，チャネル，通信端末を含める.
(11) 論理配列相互間における演算ができること.

3.3 式と引き数

(1) 副次効果は左から右へ計算するものに従う.
(2) 読みやすく，かつ演算子プレシデンスの階層は少なくする.
(3) 定数，変数が許される場合は式も許す.
(4) 定数式は実行前に計算する.
(5) 手続き，配列，宣言などに伴うパラメータには一定の規則を定める.
(6) 仮引数と引数は型の一致を必要とする.
(7) 仮引数にはクラスを設ける.
(8) 手続きの宣言中にパラメータの属性を許す.
(9) 手続きあたりのパラメータ数は可変とする.

3.4 変数，直定数，定数

(1) 識別子に定数値を与えてよい.
(2) 定数のプログラム中でもデータ中でも値を持てる.
(3) 宣言済の変数の初期化は認めるが，値をデフォルトとして定めてはならぬ.
(4) 固定小数点変数の範囲，ステップサイズは指定による.
(5) 配列とレコードの要素の型はいずれでもよいものとする.
(6) ポインタ変数の信頼度を上げること.

3.5 定義の方法

(1) ユーザにデータの型の追加を許す.
(2) 本来定義されている型と追加された型の区別は無いものとする.
(3) デフォルト宣言は許さない.
(4) 演算は新しいデータ型も含めて行えること.
(5) 型の定義には自動的には演算定義を含まない.
(6) 型の追加法としてエヌメレーション (enumelation)，カーテシアン積，ディスクリミネーテッド・ユニオン (discriminated union)，パワーセット (power set) などが考えられる.
(7) 型の定義でフリーユニオンと，サブセット化は好ましくない.
(8) 型の初期化，終結の手続きは定義できるものとする.

3.6 スコープとライブラリ

(1) 割当てのスコープとアクセスのスコープの区別.
(2) 識別名へのアクセスは定義された場所もしくは CALL される場所で限定してよい.
(3) 識別名のスコープはコンパイル時に定める.
(4) ライブラリは充実するものとする.
(5) ライブラリには異種言語で書かれたものを含める.
(6) ライブラリとコンプール (compool—宣言に関する共有可能な記述などを収容する) は区分しない.
(7) 機種依存性のあるインターフェースについては，定義を標準ライブラリに含めること.

3.7 制御構造

(1) 制御は構造化し，パラレル・プロセシング，エクセプションと割込の処理を含める.
(2) 最小のアクセス・スコープにおいてのみ GO TO を許す.
(3) 十分に区分した形で IF-THEN-ELSE, CASE, Zahn の手法を許す.

(4) 局所的制御変数により繰返し演算を行う.
(5) 再帰型手続きを許す.
(6) パラレル・プロセシング, 同期処理, クリティカル・リージョンなどを含める.
(7) パラメータ化したエクセプション処理をユーザが定義することを認める.
(8) 実時間, 模擬時間を扱う. 優先順位の相対関係を認める. 同期処理をこれについても行う.

3.8 文法と注釈の規則

(1) 自由書式とし, 区切り記号を自由とする.
(2) 原始言語の文法の変更は許さない.
(3) ASCII 64 文字セットを用いる.
(4) 識別子, 直定数には構成則を定める.
(5) 句等のつづり方を行間にまたがらせない.
(6) キーワードは少数とし, 予約語あつかい. 区切りやすくして識別子との混同をなくす.
(7) 注釈は一定の規則に従って書くようにする.
(8) 一対とならないカッコは許さない.
(9) 関数呼出しとデータ選択の区別をしないこと.
(10) 同一文脈中の記号は一意とする.

3.9 デフォルト, 条件つきの翻訳, 言語上の制約

(1) 演算の結果に影響を与えるような未定義デフォルトは許さない.
(2) 言語の機能を実現するにあたって, 最適化に有用ならばデフォルトを活用すべきである.
(3) 対象機種の諸条件を翻訳時変数としてあたえること.
(4) 条件つきの翻訳を可能とする.
(5) 全言語系を効率よく定義できるような, 簡単な基本言語系を設定すること.
(6) 翻訳プログラムの制約は言語の定義に含めること.
(7) 対象機種固有の制約条件は言語には含めない.

3.10 目的プログラムの効率

(1) 汎用性のために実行効率を犠牲にしないこと.
(2) 言語設計にあたって最適化時の安全に配慮すること.
(3) ハードウェアの機能や機械語にはエンカプシュレーテッド (encapusulated) な形でアクセスする.
(4) データについては構造の目的表現 (object representation) を指定できるものとする.
(5) サブルーチンの呼出しはプログラマの指定による. 開いたサブルーチン, 閉じたサブルーチンの双方を可能とする.

3.11 環境条件

(1) オペレーティング・システムは前提条件としない.
(2) モジュールの結合は言語機能のなかで行えるようにする.
(3) リンカ, ローダ, デバグ支援プログラム等を用意すること.
(4) 文書化, 編集処理, 検査その他のプログラマ支援プログラムを整備すること.
(5) アサーション, デバッグ仕様指示, 効率測定などの分野を導入する可能性を残すこと.

3.12 翻訳プログラム

(1) スーパーセット化, 予定されない機能の追加などは禁止する.
(2) サブセット化も禁止する.
(3) 最適化と翻訳時間の選択はユーザにまかせる.
(4) 多種のハードウェアに翻訳プログラムを用意する.
(5) 対象機種には必ずしも翻訳プログラムが無くてもよい.
(6) 構文検査はしても自動修正はしてはならない.
(7) 言語仕様の一部として誤り診断法を指示する.
(8) 翻訳プログラムの内部構造は言語標準の外の問題である.
(9) 翻訳プログラム自身は原始言語で記述されること.

3.13 言語定義上の標準とその管理

(1) 個々の機能は現在の技術水準の上に設定されるべきである.
(2) あいまいさをなくすこと.
(3) 抽象的な計算機をベースとして入門書, 参照マニュアル等を作成すること.
(4) 標準と合致していることを確かめるため, コンフィギュレーション・コントロールを行う.
(5) 委託機関は言語の保守と支援ソフトウェアに責任を持つ.

(6) ライブラリに対しても標準と責任者を定める.

参考文献

1) B. C. De Roze: An Introspective Analysis of DOD Weapon System Software Management, Defense Management Journal, October, pp. 2～7 (1975).
2) D. A, Fisher: Programming Language Commonality in the Department of Defense, Defense Management Journal, October, pp. 29～33 (1975).
3) Department of Defense Requirement for High Order Computer Programming Languages "TINMAN", Defense Advanced Research Projects Agency, June (1976).
4) Department of of Defense Requirements for High Order Computer Programming Languages "IRONMAN", Department of Defense, 14 January (1977).

追記

昨年夏, HOL プロジェクトの契約が行われ, 4社のソフトウェア会社がそれぞれ 20 万ドル程度の規模で開発作業を始めることになった. 本稿を執筆した時には入手できなかったが, 7月に "Revised IRONMAN" が DOD から発表されている. "Revised IRONMAN" は非常に言語仕様を意識した形に構成されている. 新言語の文法, プロセッサなどに興味を持たれる方の一読をおすすめする.

本稿は主として Language Evaluation Coordinating Committee (LECC) の HOLWG への最終上申の情報(昨年4月1日付)によったもので, 技術的な記述は原則として報告中に引用されている文書によった. 上申の内容は大きくいって2項目にわかれる. 一つは ECS 用の新共通言語開発の必要性を解説した部分で, 技術的項目を除去した形で本稿を読んでいただくに等しい. 他は新言語を考えるにあたって既存言語のうちから何を基礎とすべきかという比較論である.

LECC の結論は 23 種の言語のなかから次の3種を選ぶというものであった.

PL/I: FORTRAN, ALGOL 60, COBOL の利点を含む
PASCAL: ALGOL 60 の流れをくみ, 簡易性に富む
ALGOL 68: ALGOL 60 の発展形で汎用性に富む

ところで次の 11 種の言語についても配慮すべき余地が残るそうである.

HAL/S, PEARL, SPL/I, PDL/2, LTR, CS-4, LIS, EUCLID, ECL, MORAL, RTL/2

さらに全く役に立たないときめつけられたものが9種ある.

FORTRAN, COBOL, ALGOL 60, TACPOL, CMS-2, SIMULA 67, JOVIAL J3B, JOVIAL J73, CORAL 66

この結論に至る過程は大部の報告書にまとめられているが, ここでは比較論を受売りする気はない. むしろこうした選択から, 賢明な読者諸氏にプロジェクトの表に出ないニュアンスを読取っていただきたいと思う(比較作業は間に合せでなく本格的なものであった).

技術情報はおおむね "TINMAN" 及び付属文書によっている. より新しい "IRONMAN" を紹介しては, との意見もあったが, 新しくなればなる程, 言語の文法説明的形になっており, しかも用語の意味, 文章の意図に疑問を持つ人が無くならない状態にある. DOD のプロジェクト設立, 技術的項目の選択などに関する内容は "TINMAN" が最もくわしいので, 今回はそれに従うこととした.

言語比較の結論からもわかるように, 文書の表現, 用語などに PASCAL の影響が濃く現れている. しかも定義なしの使用なので, "Revised IRONMAN" ですら不分明な点が多く, HOLWG には多くの質問が寄せられている様子である. 日本からは調べのつかぬ部分も多かったので, 読み難さ, わかりにくさについては, おわびをしておきたい.

今のところまとめられているのは新言語系に対する要請だけであることを注意されたい. 要請は本来プログラム使用者の意図であって, 技術標準とか, あるいはそれ以外の企画を意味するものではない. 米国防省のプロジェクトだけに影響を広範に及ぶ可能性はあるが, 必ずしも新汎用言語の実現を意味するものではないのである. したがって本稿でも言語解説的アプローチは努めて避けた.

保守コストの軽減, 時間的・機種的互換性など, このプロジェクトが最終目標としてかかげている命題は, いわゆるソフトウェア・エンジニアリングのそれと重なる部分が多い. 正統的にシステム開発の始めから終りまでを追いかける行き方に対して, 言語開発をもって対処する DOD の行き方に興味をおぼえるものである.

"Revised IRONMAN" は LECC とは本質的な差

はないが，表記法が要請の記述から言語の概略仕様へと変身している．主な変更を拾ってみる．

(a) TYPE が拡充強化された．

Numeric type
- Integer type
- Fixed point type
- Floating point type

Enumeration type
- Ordered enumeration type
- Boolean type
- Character type

Composite type
- Array type
- Record type
- Composite type は他の type の Cartesian product として定義されている．record には variant type を許す．動的割当て可能な dynamic type の機能も含める．

Set type
- bit string である．

(b) オペレーティング・システムとの独立性が入出力動作にまで適用される．ただし翻訳系に既知なファイルだけを入出力の対象とすることが明記された．

(c) 入出力系

入出力に低・高の2つの水準を指定．低水準ではチャネル入出力の水準でプログラムの制御が行えることを目的とし，高水準 (application level) では理論ファイルを扱う．

(d) データ・コード

データ・コードは任意であることは定められていたが，一歩進んで複数の表現形式を許容する (packed, unpacked など) ことが明示された．

(e) そ の 他

改訂された細目は多いので例示するにとどめる．

↑がオペレータから function に変った．

引数に多重アクセスパスが許されない．

データ型の定義法として表示されていた discriminated union, power set, enumeration などの例示がなくなった．

(昭和52年9月8日受付)
(昭和53年1月25日再受付)