インライン処理指向のプログラミング言語†

魚田勝臣††　溝口徹夫†††
小碇暉雄††　富沢研三††

インライン処理はデータを1件別に入力し，チェック，ファイル処理や計算などを行って，結果をその場へ出力する方式で，オフィス・コンピュータや分散処理の端末側で広く実用されている．この方式では，キーボードからのストリング型データやキャラクタ・ディスプレイからの2次元型データを入力し，会話形式で項目単位にデータを処理して，プリンタやディスプレイに出力する．したがって，バッチ処理と比較して最終結果は同一でもプログラムは違ったものになる．インライン処理のプログラムをCOBOLやRPGで作成すると，これらがユニット・レコードの取扱いをベースにしているため複雑なものになる．このことはキャラクタ・ディスプレイのシステムで顕著である．筆者らはインライン処理の特質をシステム，入出力および処理の面から分析し，その結果に基づいて表形式の言語"プログレス"を開発した．プログレスでは8種類の指示書と呼ばれる記入用紙にプログラムの条件を記入するようになっており，COBOL比でコーディング行数が1/2～1/5程度に減少し，プログラム作成の生産性が2～3倍に向上できることが確かめられた．プログレスはMELCOM 80シリーズ（オフィス・コンピュータで主メモリ64 KBから）のユーザで広く活用され，生産性向上に役立っている．本文ではプログレスの言語およびコンパイラについて報告する．

1. まえがき

最近のデータ処理は従来にも増して計算機よりも使用する人間の能率を重視するようになっている．これはハードウェアの性能対価格比が上り，安く使えるようになった反面，システムの開発や運用に必要な費用が増大しているので当然の動きといえよう．

システムの運用面については，計算機室内での一括処理から現場指向になっていることがあげられる．すなわち，現場の端末からデータ・エントリだけを行う段階から，処理の一部をインテリジェンスを持った端末側で行わせる考えになってきている．分散型データ処理はこの傾向を物語っている．一方，オフィス・コンピュータの分野では以前からインライン処理と呼ばれるデータ処理方式が活用されてきた．これは，データをせん孔カードなどの中間媒体を経ず一件別に入力し，チェックやファイル処理などをして結果をその場に出力する方式であり，処理形態が人手によるものに近いため考えやすく，今後も一層増えるものと思われる．

インライン処理や分散処理の端末側処理（以下，これらをインライン型処理または単にインライン処理と呼ぶ）ではキーボードからのストリング型データや，キャラクタ・ディスプレイからの2次元型データ（本文では文字のみを扱う）を入力し項目単位で処理する必要がある．また，オペレータ・ガイダンスやデータのチェックおよび訂正も綿密かつ親切に行わねばならないので，カードに代表されるユニット・レコード型の入出力および処理の方式とは趣きを異にする．

以上に述べたシステム運用面での進展に対応して，システム開発費用を軽減することがユーザやメーカにとって急務である[1]．このために，データベースに代表されるデータ処理方式の研究，プログラミングの方法論の研究（要求仕様記述法や構造化プログラミング）など広範囲におよぶ研究がなされているが，ここに取上げるプログラム言語も重要なテーマの一つである．

インライン型処理用の言語として，COBOL[2]やRPG[3]などを使用することも可能である．しかし，これらはユニット・レコードの取扱いをベースにして言語が構築されており，特にキャラクタ・ディスプレイを用いたインライン・システムではプログラムが複雑になる．これまでもキャラクタ・ディスプレイを対象とするプログラミングを簡単化する試み[4]がなされているが，それらは画面の設計ないし入出力に関するものが多く，処理も一緒にした試みはほとんどなかった．筆者らはインライン型処理のプログラムが簡単にかつ能率よく開発できることを目的とし，COBOLやRPG

† An In-line Processing Oriented Programming Language by KATSUOMI UOTA (Computer Works, Mitsubishi Electric Corp.), TETSUO MIZOGUCHI (Computer Laboratory, Mitsubishi Electric Corp.), TERUO KOIKARI and KENZO TOMISAWA (Computer Works, Mitsubishi Electric Corp.).
†† 三菱電機（株）計算機製作所
††† 三菱電機（株）開発本部

をベースにして独自の言語を開発した[5),6)]．この言語はプログラム作業が少なくて済むようにという願いをこめて，プログレス（プログラム／プログラマ・レス，PROGRESS——進歩）と命名した．

以下，本文では，第2章でインライン処理の特質，第3章でプログレスでの解決策，第4章でインプリメンテーション，第5章で考察（生産性についても触れる）について述べる．

2. インライン処理の特質

この章ではインライン処理の特質をバッチ処理と対比させながら示し，それを指向したプログラム言語および処理システム上の問題点を明らかにする．

2.1 システム上の特質

インライン処理は**図1**に示すように，1件のトランザクション・データごとに処理するもので，データ・エントリ，問合せ，伝票発行などの諸業務に応用されている．この内で伝票発行がそれ以外の業務の処理内容をすべて含んでいて，プログラムとして最も複雑である．たとえば売上伝票の発行というような処理を考えると，原始伝票の1入力項目（伝票発行のための Transaction data）をキーインするごとに関連の処理が行われて伝票が出力される．このような処理を主体としたシステムがインライン・システムである．これに対して，バッチ処理ではデータの入力，チェック，ファイル更新などを，各段階ごとに全レコードを一括して処理する．

インライン・システムはオフィス・コンピュータなどを用いた小規模システムで主として実用されている方式であるが，分散処理の端末側や会話型処理もプログラム上はインライン処理と類似の方式として取扱うことができるので，ここに述べるプログラム方式は広い応用分野を持っている．

2.2 入出力方式

入出力方式はデータの形式上次の3種に分類できる．

（1） ユニット・レコード型　カード・システムで代表されるシステムの入出力はこの型で，1次元固定長のレコードを対象とする．出力もラインプリンタが主体であるからユニット・レコード型である．

（2） ストリング型　紙テープや磁気テープのような媒体，通信回線を経由した入出力がこれである．また，キーボードからの入力およびシリアル・プリンタへの出力もストリング型といえる．

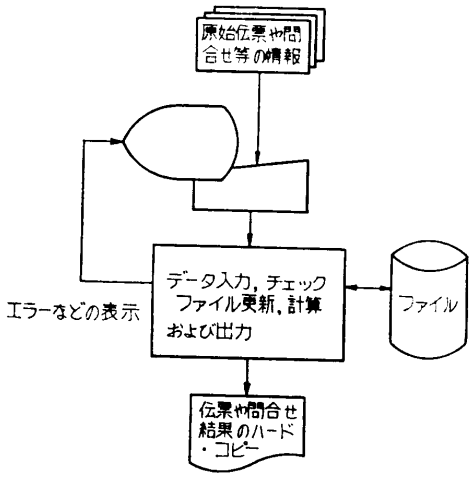

図1 インライン処理
Fig. 1 In-line processing.

（3） 2次元型　キャラクタ・ディスプレイのような表示画面を持つもので，入出力位置が2次元的に移動できるところがストリング型と異なる．

バッチ・システムでは（1）を主体とし，（2）が小形機などで用いられていた．最近は（3）のデータ・エントリも増えている．インライン・システムでは，入出力装置にキーボードとプリンタが用いられていたため，従来は（2）の方式が採用されていた．最近はキャラクタ・ディスプレイが多用されるようになったので（3）に主体が移っている．

インライン・システムの入出力ではオペレータ・ガイダンスとデータのチェックおよび訂正が重要である．すなわち，計算機の全くの素人が簡便に操作できるように，入力を要請するメッセージ (Prompt message) を出力することはもちろん，入力データの訂正も項目ごとやページごとに可能にするなど，きめの細かい配慮が必要となる．

2.3 処理方式

処理方式はシステムおよび入出力方式の特質と関連して次の2種に分類される．

（1） レコード処理型　処理の流れが入力レコードによって制御され，レコードを単位として処理され結果が出力される．合計処理など入力レコードの前後関係によって流れが制御されるものや，ページ冒頭処理のように出力装置の物理的な条件の発生によって制御される処理もあるが，基本的にレコード処理であることに変わりはない．バッチ・システムはこの処理方式でシステム設計されており，COBOL や RPG もレコード処理を基本としている．

なお，紙テープや通信回線経由のストリング型データを取扱う場合も，バッチ・システムでは入出力だけがストリング型で，内部処理はすべてレコードにまとめられるのでレコード処理型である．

（2） 項目処理型 処理の流れが入力項目によって制御され項目を単位として処理され結果が出力される．したがって，この場合の入力方式はストリング型か2次元型になる．インライン・システムではこの型の処理が行われることが多く，各項目の入力によって，データのチェック，ファイル処理や計算などの流れを制御する必要がある．また，項目ごとに結果を出力するケースが多く，オペレータのキーインのタイミングと合致させるために処理の高速性が要求される．

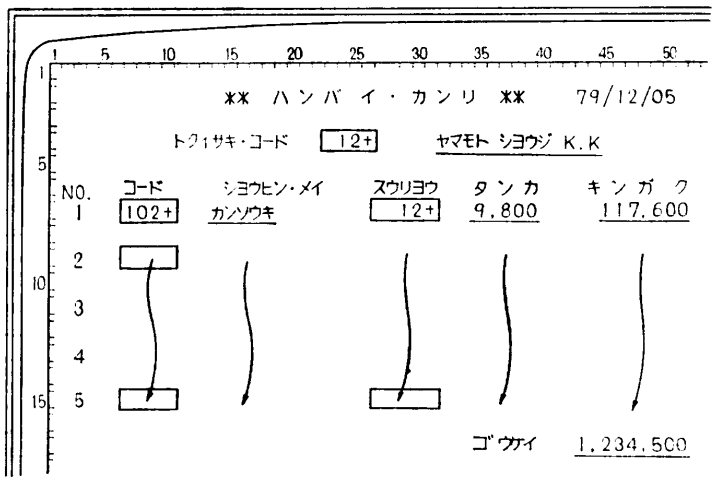

図 2 プログラム例の表示画面

Fig. 2 Screen image displayed by the sample program of Fig. 3 and Fig. 4.

以上記述した3点でインライン処理はほかと違った特質を持っており，ディスプレイを使ったシステムで特に顕著である．インライン・システムでも集計計算などの事後処理はレコード処理型なので，これに適する言語としては両面をカバーする必要がある．

3. プログレスでの解決策

この章では第2章で記述したインライン処理としての特質を持ったプログラムが簡潔に書けるようにするためにプログレスで採用した言語上の方式を記述する．以下，この章では**図2**に示したディスプレイの画面を用い，次に示す処理を行うプログラムを例にあげる．

（a） 図2でけい線と呼ばれる枠で囲まれたところが入力項目である．データは項目単位に入力され必要なファイルの検索・更新および計算などの処理をして結果を表示する．この表示項目は下線で示されているが，下線そのものは表示されない．

（b） 明細行は1画面（1ページ）についてこの例では5行ある．ただし，(商品)コード入力時に“0”がキーインされると，そのページの明細行は終りとなる．また，画面ごとに合計金額を表示する．

（c） 得意先コード入力時に“オワリ”がキーインされると後始末をして処理を終了する．

3.1 言語方式の選択

言語方式として手続き向き言語と非手続き向き言語のいずれを採用するか検討した．次の理由により後者とし，記入文字を最少限にするため表形式（Fill in the blank）にした．

（1） 記述要素を最少限にすること．コーディングが簡単な上，プログラム入力（または媒体変換）も冗長度が少ないから全体の文字数が少なくてすむ．特にオフィス・コンピュータのように専任の担当者を置かないユーザの場合にはプログラム入力を会話型で行うなどの配慮も必要である．なお，英語が全く判らなくてもコーディングできることも選択の理由である．

（2） システム設計とプログラミングが同時に教育できる．また，システム設計書から必要事項を書き写せばコーディングの大半が完成する．

（3） ソフトウェア開発の生産性を上げられること．

プログレスのコーディング・シートは“指示書”と呼ばれ，① プログラム指示書，② 端末指示書，③入力ファイル指示書，④ 出力ファイル指示書，⑤ プリント指示書，⑥ 端末入力指示書，⑦ 作業項目指示書および ⑧ 処理指示書の8種類である．

図2の処理にはこの内の5種を使う．インライン処理として特徴のある2つの指示書の記入例を**図3**および**図4**に示す．

3.2 既定論理

表形式の言語の特長を生かし記入すべき手続きを少なくするために，プログレスではインライン処理の標準的な流れを想定している．これは既定論理と呼ばれ**図5**に示す．この論理はデータ・エントリ，問合せ，

端末入出力指示書

ページ 03 ID RUN1.

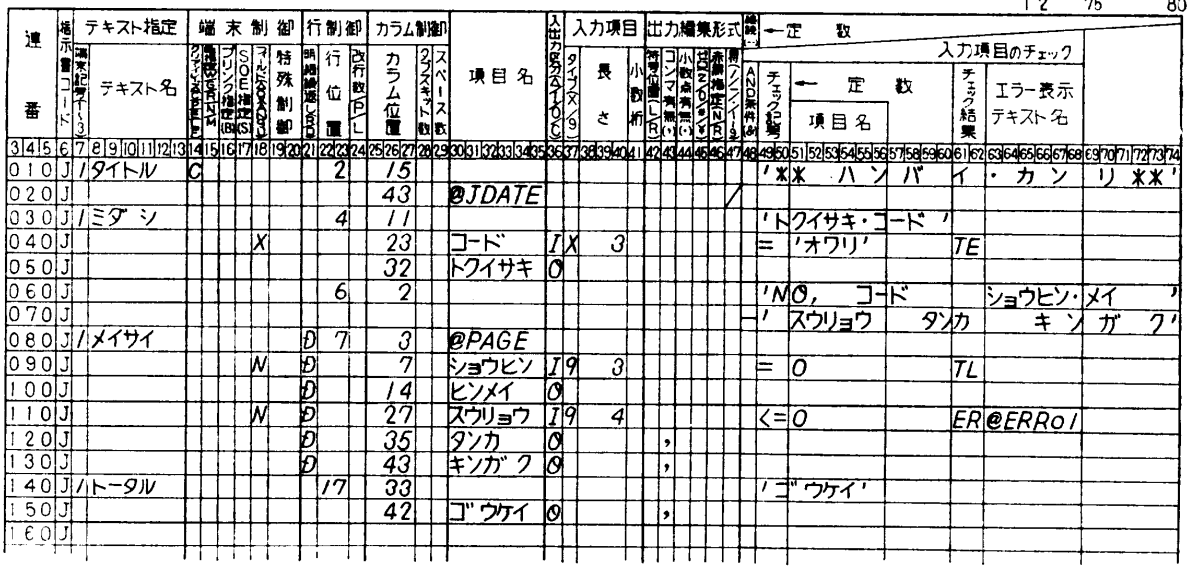

図 3 端末入出力指示書の記入例

Fig. 3 Sample coding of the terminal I-O specification sheet.

処理指示書

ページ ID

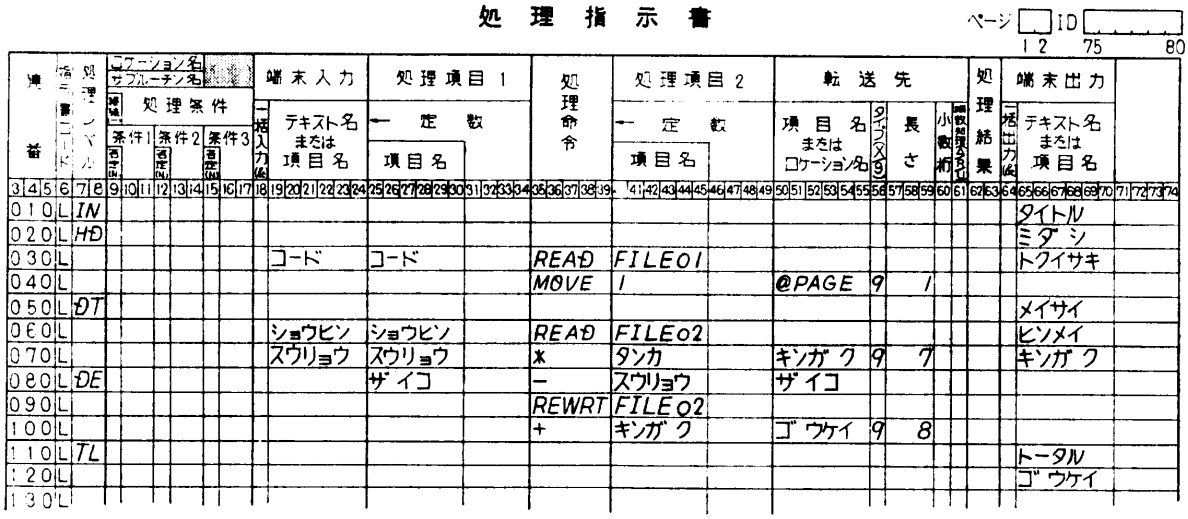

図 4 処理指示書の記入例

Fig. 4 Sample coding of the procedure specification sheet.

伝票発行などほとんどのインライン処理に適用可能で，端末処理を強く意識した流れになっていることや1行訂正の機能も盛り込まれているなど，RPG のようなバッチ型に比べてユニークな内容になっている．

3.3 入出力の指定方法

インラインで最も特徴のある入出力は端末に関するもので端末入出力指示書に記入する(図3)．**図6**はこの記入項目を要約したもの，**図7**はこの内の入力項目のチェック欄を浮きぼりにしたものである．

端末との入出力はデータの性質によってテキストと呼ばれる単位にまとめられる．図3の例では，タイトル，ミダシ，メイサイおよびトータルである．

(1) 会話型入出力の指定 プログレスの目差すインラインの会話型入出力では，

データ入力を促すメッセージの出力→オペレータによるデータの入力およびエラー・チェック→処理→結果の出力

のサイクルが繰り返される．これを関連づけて見やす

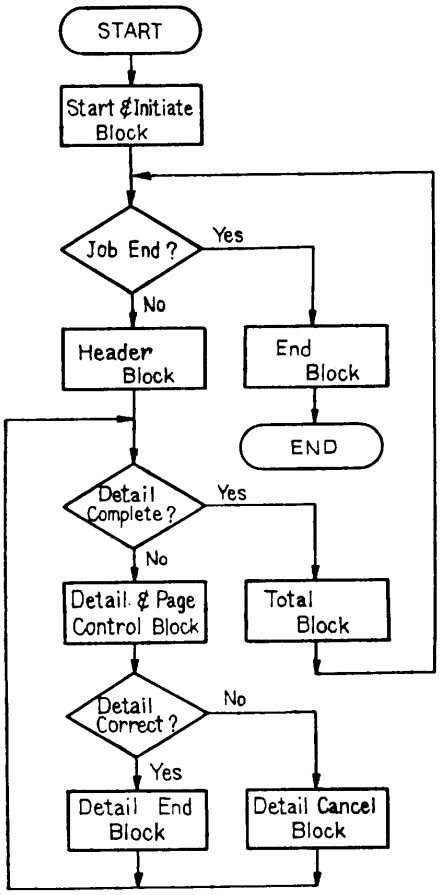

図 5 インライン処理の既定論理

Fig. 5 The default logic of in-line processing program.

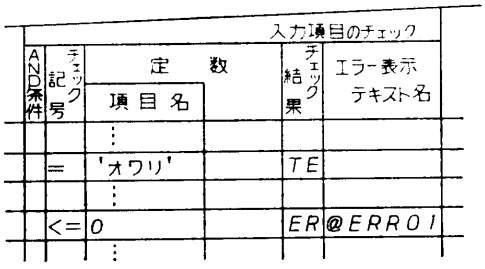

図 7 入力項目のチェック（詳細）

Fig. 7 Checking of input items (detail).

く指定できるようにするために，入出力を同一の指示書に記入することにし，両者を区別する欄：入出力区分（図6）を設けた．ただし，出力すべきデータが記入（定数欄，図3の010行）されていて，文脈から区分が判定できるときはこの欄の記入は不要である．(文脈による記入の省略は随所に盛込まれている．)

端末入出力指示書には，入出力動作の画面上での位置や項目名など入力と出力とで共通に使う記入欄と，

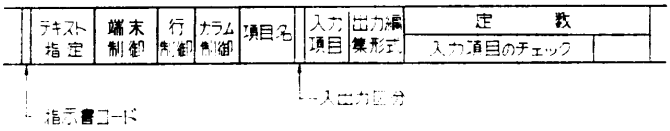

図 6 端末入出力指示書の記入項目（大項目）

Fig. 6 Specification items of the terminal specification sheet (major items).

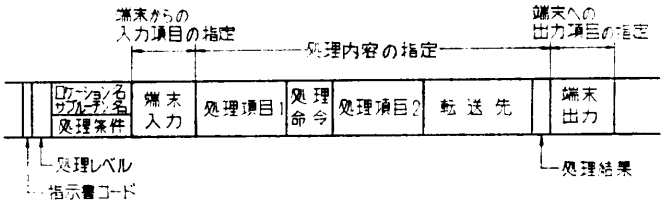

図 8 処理指示書の記入項目（大項目）

Fig. 8 Specification items of procedure specification sheet (major).

入力項目に対する長さの指定や出力項目に対する編集形式の記入欄など，どちらか一方にしか使われない記入欄とが設けられている．この指示書の設計にあたって1行の字数を押えるのに工夫をこらした．

（2） 2次元型入出力の指定 画面の位置を指定するには座標表現するのが一番簡明であるので行位置（Y 座標）とカラム位置（X 座標）とで指定することにした．同一行でカラム位置だけ変わるときは行番号を省略できる．また，明細行などのように同一形式の行が繰返されるときには一行分の記入ですむように“明細繰返し”欄を設けた(図3，21カラムの“D”欄)．

（3） 入力データのチェックと処理内容の選択 入力されたデータをチェックするために“入力項目のチェック”欄が設けられている(図7)．単純条件だけでなく複合条件も指定できるので綿密なデータ・チェックが可能である．チェックの結果でオペレータにメッセージを表示し，必要に応じて再入力（訂正）を促すこともできるし，以後の処理内容を制御するために“チェック結果”欄に指定した標識をたてることもできる．このように本来なら面倒な処理手続きを必要とするデータのチェックや訂正が，画面定義と同一行に簡単に指定できるのでコーディング行数が著しく減少する．

3.4 処理の指定方法

計算やファイル処理の内容は処理指示書に記入する(図4)．**図8**は記入欄を要約したものである．処理内容は既定論理のブロックに対応してまとめて記入され，各処理ブロックの先頭行の“処理レベル”欄に指定される．

(a)	Start Block	ST
(b)	Initiate Block	IN
(c)	Header Block	HD
(d)	Detail Block	DT
(e)	Detail End Block	DE
(f)	Detail Cancel Block	DC
(g)	Page Control Block	P1～P3
(h)	Total Block	TL
(i)	Terminal End Block	TE
(j)	Job End Block	JE

処理ブロックが 10 種類に分かれているのは複雑なようであるが，インライン処理の標準論理で役割が明確であるから，実際上の煩わしさは少ない．(f) や (g) は入力データの訂正を自在にするためのブロックである．なお，通常のデータ・エントリ，問合せや伝票発行などでは 5～6 種の処理ブロックで十分である．

処理指示書では処理内容を図 8 に示したように，左端に書かれた端末入力をし，中央部に書かれた処理をして右端に書かれた端末出力をする，という形で書かれる．バッチ処理の場合は両端は不用であるから，この形式もインライン用言語の特徴である．

入力項目の内容チェックによる処理の起動は図 3 の 090 行と図 4 の 110 行の標識“TL”のようにして行う．すなわち，入力されたデータが“0”ならば“TL”の条件が設定され，それが合計処理に分岐させることになる．また，ファイル処理や計算のための各種の命令 (READ, WRITE, MOVE など) が設けられていて，項目単位の処理が自由にできる．

4. インプリメンテーション

4.1 コンパイラのフェーズ構成

3.2 節で記述したようにプログレスは既定論理をもつ言語であるから，この特徴を生かしてコンパイラを形成するのが得策である．すなわち，既定論理のオブジェクト・コード (これをスケルトンと呼んでいる) をあらかじめ持っておき，それに指示書で与えられた変数や手続きに基づいて肉づけをすればオブジェクト・モジュールが作成できるので，コンパイラは比較的簡単に製作できる．プログレスのコンパイラは**図 9** に示すように論理的に 4 段階からなっている．この図から判るように指示書の内容は一度だけ Scan されて中間結果として解析ファイルが作られ，スケルトンおよびモジュール・ファイルと合成されてオブジェクト・モジュールになる．

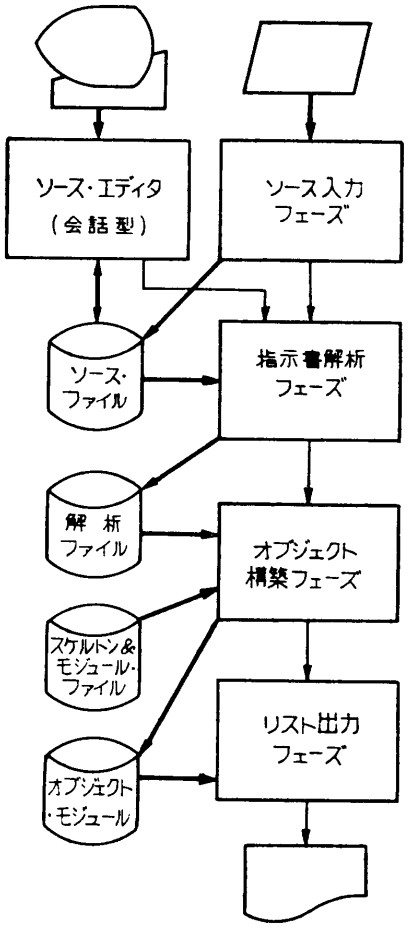

図 9 コンパイラのフェーズ構成
Fig. 9 Structure of PROGRESS compiler.

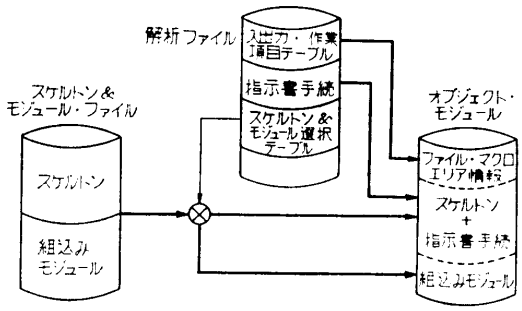

図 10 オブジェクト・モジュールの合成
Fig. 10 Composition of object module.

図 10 は各ファイルとオブジェクト・モジュールの関連を示したものである．太線は情報の流れ，細線は制御を示している．**図 11** は既定論理 (図 5) の 7 つのブロックのそれぞれが，つぎの 3 部分からなることを表わしている．

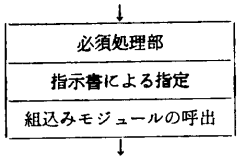

図 11 各 Block での処理の内訳
Fig. 11 The detail of each "Block" processing.

(1) 必須処理部 指示書の内容によらず必ず組込まれる部分.

(2) 指示書による指定 処理指示書に指定された手続き(たとえば図4の内容)が組込まれる部分.

(3) 組込みモジュールの呼出し 明細行の繰返し指定(図3の D 指定)などのように,指定があればそれに対応するモジュールの呼出しシーケンスが組込まれる部分.

このようにプログレスでは指示書の解析結果が組込まれる部分が一意に定められているので,複雑な論理の解析は必要でなくオブジェクト構築フェーズも比較的簡単になった.

4.2 組込みモジュール

プログラムの中で頻繁に使われるテクニックでかつユーザが簡単にプログラムできないものは組込みモジュールとして準備し,指示書の簡単な指定だけで呼出せるようにした.代表的なものを2種報告する.

(1) 明細行の繰返しモジュール 指示書に指定された明細行1行分に関するデータと明細行数(繰返し回数)とをパラメータとして受取ることにより,全明細行分の画面定義,カーソルの制御,処理の繰返しなどを行うモジュールである.

(2) ページ・プリント・モジュール このモジュールは端末指示書で複数の入力端末に対して1台のプリンタが指定されたときに組込まれるモジュールで,各端末からの出力データを一旦ディスクに収容し,端末ごとにページ単位にまとめてプリンタに出力する機能を持っている.この機能により,ページは交錯するがプリンタ台数が少なくてすむので同一伝票を複数のキャラクタ・ディスプレイから発行する場合などに有効である.

4.3 コンパイラの諸元

プログレスのコンパイラはオフィス・コンピュータ MELCOM 80 シリーズ(主記憶:64 KB, 10 MB のディスク,プリンタ,キャラクタ・ディスプレイおよびキーボード各1台が最小構成)の上でインプリメントされた.32 KB の領域で動作させるため,図9の解析フェーズを4,オブジェクト構築フェーズを3に分割し,かつ制御フェーズを設けた.よって,それ以外の3フェーズと合せ計 11 フェーズとなった.コンパイル速度は 150 ステートメント/分で, 同一問題が COBOL の 1/2〜1/5 のステップ数で作成できることを勘案すると十分な速さである.なお,プログレス・コンパイラはアセンブラで書かれており,総ステップ数 63,000(含コメント行)である.また,開発期間は6カ月で同レベルの COBOL に比べて半分程度で作成できる.

表 1 コーディング行数の比較
Table 1 Comparison between the number of steps in coding.

プログラムの種類 \ 使用言語		コーディング行数		
		プログレス	COBOL	比 率
インライン処理	伝票発行	91行	482行	1:5.3
	データ収集	36	219	1:6.1
	問合せ1	36	199	1:5.5
	問合せ2	90	486	1:5.4
	合計	253	1,386	1:5.5
バッチ処理	データ・チェック	209	441	1:2.1
	レコード分割	77	222	1:2.9
	集計表	70	182	1:2.6
	請求明細	58	151	1:2.6
	データ編集	133	291	1:2.2
	合計	547	1,287	1:2.4

5. 考 察

以上に記述したプログレスをユーザに提供した結果 COBOL に比べてコーディング行数で約 1/2(バッチ処理の場合)〜1/5(インライン処理)になった(**表1**に同一問題をコーディングした場合の行数の比較データを示す). また,ユーザのプログレス導入後のプログラム生産量の増加(COBOL で1人1カ月に6本程度のものがプログレスで 20 本になった)を考えあわせるとプログラムの生産性が 2〜3 倍に向上したと推定される(生産性の定義は単純でないが[7], ここではプログラム仕様書作成からデバッグ完了までの所要時間で測られるものと考えておく).

以下,ユーザでの使用経験から二,三の事項について補足する.

(1) インライン処理がプログレスを通じて簡単に理解される.これはプログレスがインラインの標準論理を持っているので言語を通じてシステムそのものを理解できるからである.COBOL では言語を理解したあと別途業務を教えねばならない.

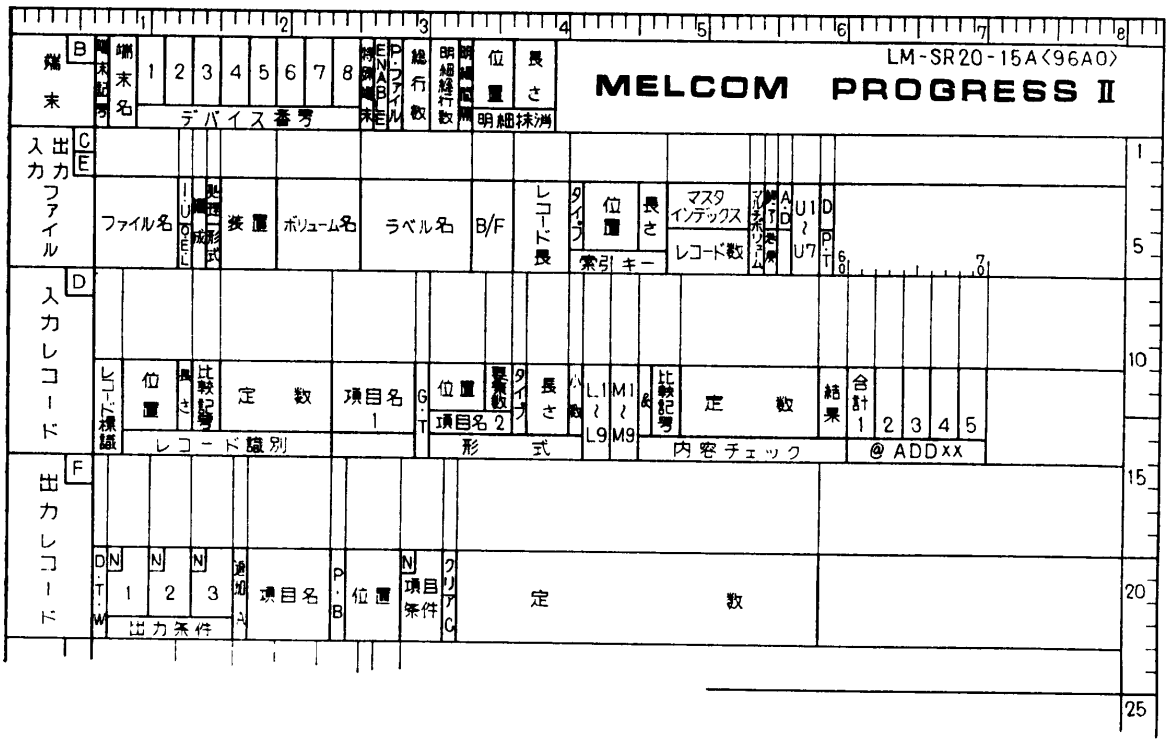

図 12 プログレス用プレート（一部）

Fig. 12 Transparent plate for PROGRESS (a part).

（2） インライン処理を簡便に指定するために採用した，端末入出力の同一指示書への記入，入力データのチェックと訂正機構，入出力位置の座標表現，内蔵機能などが COBOL と比較して著しくコーディング行数を減らせた原因と考えられる．

（3） 表形式の言語であるから記入欄の位置が決まっていてプログラム入力が煩わしい欠点があった．これは専用の会話型エディタ[8]を開発して解決した．このエディタはプログレスの指示書フォーマットをけい線を使ってディスプレイの画面上に表示することにより，使いやすさとキー・ストロークの最小化を追求したものである．シンタックス・チェックの一部の機能も持っている．また，プログラム・リスティングそのものがドキュメントとなり難い面を持っているが，システム設計書，指示書とともに保存することにより解決できる．なお，応用用紙にプログレスのソース・リストをプリントすると指示書の枠や見出しがないため欄の判別ができない．指示書の見出しを印刷した透明のプレートを用意してリスティングを見やすくした（**図 12** に一部を示す）．

（4） プログレスは現在のところ MELCOM 80 シリーズにのみ適用されている．よって他機種の言語との共通性はない．しかし記入様式が単純であり，内部処理も明確であるから，必要なら COBOL へのコンバータは簡単に開発できるであろう．

6. あとがき

以上本文では，インライン処理の特質を，システム面，入出力および処理方式の面で分析し，COBOL や RPG をベースにして，よりプログラム生産性の高い言語を目差して開発した“プログレス”について報告した．モデル・プログラムをプログレスと COBOL とで作成した結果，コーディング行数が約 1/2（バッチ処理の場合）～1/5(インライン処理の場合）になり，プログラム作成の総合的な生産性が 2～3 倍に向上できることを確かめた．オフィス・コンピュータ，MELCOM 80 シリーズのプログレスを備えた機械のユーザの 90% 以上で活用されており，約 70% のユーザでは主力言語となっており，成果を収めることができた．

今後，入出力装置の多様化に応えてゆくとともに，プログラム方法論との結びつき，日本語処理への展開について研究をすすめてゆきたいと考えている．

本研究をすすめるにあたって，MELCOM のユー

ザ，ディーラの方々から種々有益なご助言を賜わった．また，慶応義塾大学工学部助教授，大駒誠一博士および当社開発本部計算機研究部長，首藤勝博士にはいつもご指導を頂く．これらの方々に記して謝意を表する．

参考文献

1) 電子協編：オフィス・コンピュータに関する市場調査，(社)電子工業振興協会，54-C-384，pp. 3-10 (1979-7)．
2) 日本工業標準調査会：電子計算機プログラム用言語 COBOL JIS C6205，日本規格協会．
3) IBM：IBM System/34 RPG II Reference Manual SC 21-7667 (1977-7)．
4) 藤田祐二ほか：キャラクタ・ディスプレイ用ソフトウェア・システムについて，情報処理学会論文誌，Vol. 20, No. 3, pp. 243-248 (1979)．
5) 小碇暉雄ほか：MELCOM 80 シリーズ・オフィス・コンピュータ・システム新モデル，三菱電機技報，Vol. 53, No. 8, pp. 608-612 (1979)．
6) 魚田勝臣ほか：簡易言語プログレスによる生産性向上について，情報処理学会ソフトウェア研究会，12-2 (1979-10-26)．
7) Jones, T. C.: Measuring Programming Quality and Productivity, IBM Systems Journal, Vol. 17, No. 1, pp. 39-63.
8) 小池玲子ほか：非手続き向き言語“プログレス”のソースエディタ，情報処理学会第 21 回全国大会論文集，7B-5 (1980-5)．

(昭和 55 年 1 月 16 日受付)
(昭和 55 年 6 月 19 日採録)