情報処理の核言語としてのModula–2 *

若林信夫

1．はじめに

本稿は，情報処理の研究と教育において新しいプログラム言語Modula–2を使用してきた経験をもとに，情報処理の核言語としてModula–2の占める位置とその重要性を考察することを目的とする。

周知のように本プログラム言語Modula–2は1978年，スイスの連邦工科大学（ETH）のN. Wirth（ヴィルト）により，主として，システムプログラムの記述のために設計され，彼の研究グループにより処理系が作製され，世界中に広まった。この言語は，やはりWirthの設計した手続き型言語のPascalならびにModulaの直接の子孫になっていて，Pascalから基本的な言語の特徴を継承し，Modulaからはモジュールや多重プログラミングの設計哲学の大部分を継承している。したがって，Modula–2はより完成した言語となっていて，初学者から中級プログラマ，専門プログラマまで情報処理システムの核言語として使用できる。

Modula–2は先祖のPascalよりも次の5つの点で改善されている。

1．モジュール構造

「モジュール」とはプログラムの基本的な構成単位である。できるだけ単純で独立な部品化プログラムが作られることが望ましい。簡単な問題解決ならば，一つのモジュールで構成できるが，複雑な大規模な問題を処理するにつれ，全

原稿受領日　1986年10月6日
※　本稿は文部省学内特定研究経費の成果の一部である。記して感謝したい。

体を多数のモジュールに分割し処理した方が全体を見通しよく管理・保守できる。また学生たちや多数の技術者たちの共同開発を可能にし，各モジュールに責任をもたせ完成させることができる。

Modula–2には，通常の（プログラム）モジュールのほかに，定義モジュールとそれと対になる実現モジュールがある。また，システム依存部分をまとめた擬似モジュールがある。各モジュールは再利用可能な完結したプログラム群となっている。[注1)]

2．分離コンパイル

一部のPascalには分離コンパイルの機能をもつものがあるが，標準Pascalにはこの機能がない。数千行，数万行のソースプログラムをコンパイルする場合，一箇所の誤りの修正でも全体を再コンパイルしなければならないことがしばしばある。各モジュールの分離コンパイルが可能ならば，各コンパイル単位が相互に矛盾なく外部データを参照できるので計算機の使用時間や資源の節約になるばかりでなく，プログラム設計の柔軟性が増す。モジュール化と相まって，チーム開発がより容易になる。

3．移植の容易さ，拡張性

移植の容易さ（ポータビリティ）とは，あるシステムで稼動しているModula–2のソースプログラムを，他のシステムにほとんど手を加えずに実行させることができることである。Modula–2言語には，標準規格がないが，Wirth仕様があるので，それほど大きな方言はないが，システム依存の外部ルーチンをインポートする場合，若干の注意が必要である。また，処理系の中には，Wirthの第3版仕様〔18〕にまだ準拠していないシステムがある。

4．明解さ，柔軟さ，オペレーティングシステム（OS）からの独立

Modula–2は，Pascalの豊富なデータ構造と制御構造をさらに改善してい

1）定義モジュール（DEFINITION MODULE）と実現モジュール（IMPLEMENTATION MODULE）の連結はシステム側で整合的かどうか再検査される。これを「バージョン管理」という。

る。データ構造では，可変レコード（variant RECORD）構造や，ポインタ型が改善され，制御構造では，IF文やWHILE文で複文が現れると**BEGIN ... END**のブロック構造を必要とせず，最初の**BEGIN**を不要にしたため，プログラムが書き易く，また読み易くなった。また，手続きの引き数として，**ARRAY OF CHAR**のように不定長の配列を許した。入出力機能は一般にOS依存のことが多いが，Modula–2ではOSから独立のライブラリの形で提供され，Modula–2自身で書かれている（その一部は**付録1**のリストを参照）。

5．多重処理プログラミング，並列処理

Modula–2はModula言語の多重処理プログラミングを継承している。多重処理を生成したり，実行をコルーチンとして渡す能力が，割込みルーチンを実現する能力と結合されている。しかし，マイクロコンピュータのような単一プロセッサでは多重処理は当然制限された意味で使われる。Wirthの第3版ではシステム依存のモジュールであるSYSTEMの中のPROCESS型をとり去り，ADDRESS型で置き換えている。

上のような特徴をもつ高級プログラム言語Modula–2は果たして，これからの情報処理の研究と教育の核言語となりうるかどうかを検討することが本稿の課題である。

以下では，まず最初に，本稿をより自己完結的にするために，Modula–2の字句と構文について明確化する。これは，Modula–2で書かれたソースプログラムを理解したり，プログラミングするときに役に立つ。また，実際にModula–2コンパイラを作製するときの資料にもなる。

次に，Modula–2の代表的な処理系について述べる。これには汎用計算機用からマイクロコンピュータ用に至るまでかなりのものが利用可能になってきた。本稿は，われわれの最も利用してきた8ビット系マイクロコンピュータのCP/M 2．2版上で走るFTL Modula–2処理系を中心に述べる。他の処理系についても言及する。

第4節では，Modula–2における重要な目新しいソフトウェア概念のいく

つかをとり出しこれからの情報処理の課題，具体的には，信頼性・保守性の向上，プログラマへの配慮，効率重視等にどう貢献しうるかを検討する。

第5節は，Modula–2の今後の課題を述べる。

付録において，若干のベンチマークテストによってModula–2の比較優位を論じる。

2．Modula—2の字句解析と構文

本節は，1984年に改訂された第3版のModula–2言語の字句解析と構文について述べる。そのさい，意味を明確にするために特定の処理系FTL Modula–2（後述3．1を参照）を援用することがある。

2．1　Modula—2の字句解析

Modula–2言語の字句構造とは，ソースプログラムで認識されるトークン（記号列）を並べたものである。

1．名前

名前は変数，手続き（関数）等の基本的な記号列である。名前は英字から始まり，英字または数字が続く文字列である。大文字と小文字は完全に識別されているというのがModula–2の特徴である。名前の長さは，Wirthは定めていないが，通常，8文字である。しかし，FTL Modula–2では最初の32文字が有効でさらに特殊文字，ドル符号（'$'）と下線記号（'_'）を含めてよいことになっている。

2．整数リテラル

整数リテラルは，十進数，十六進数，八進数あるいは，数値文字リテラルのいずれかである。十六進数の場合，'h'または'H'を，八進数の場合，'b'または'B'をそれぞれの数値の後におく。数値文字リテラルは八進数（0～7）のあと，'c'または'C'がつく。

例えば，数値 $2^{15}-1$ は次のように表される。

32767　　0FFFFH　　0ffffh　　37777B

3．実数

実数リテラルは，十進数字（0～9）から始まり，さらに十進数が続くか，ひとつの小数点がある。実数リテラルには，単純な小数点表現のほかに，尺度因子E表現がある。例えば，

1986.9　　1.9869E3　　2E3

である。通常のコンパイラでは最後の表現は許されないが，FTL Modula–2では許される。実数xの出力は，**WriteReal**（x, n）; によるが，ここで，xは，全体がn桁で表される尺度因子Eつきの出力となる。

4．文字列

文字列は，引用符（クォート）'か，二重引用符（ダブルクォート）"で始まり，0個以上の文字が来て，最後に初めと同一の引用符または，二重引用符で閉じる。

'A'　"Beginner's Guide"　'"'

文字列の中に，ESC文字を含めたいことがある。FTL Modula–2では，コンパイラオプション（＊ $A^ ＊）を用いて可能である。

5．注釈（コメント）

注釈は，'(＊'から始まり，任意の文字が続き，'＊)'で終る。注釈にはさらに注釈が含まれる。これを入れ子構造（ネスティング）というが入れ子の深さは，基数の最大値（＝65534）まで可能である（これは，処理系では，初め0にセットされたレベルカウンタがあり，'(＊'が現れる度に，1ずつ増やされ，'＊)'がきたら，バランスされる）。

6．ファイルの終り，行の終り

Modula–2 のソースはテキストが1行かそれ以上に構造化され，内部の EOF マークで終了する。他方，行の終り（改行文字）は字句的には空白文字（スペース）と同じである。

7．区切り文字（デリミタ）

区切り文字は，トークンを区切る2文字または1文字の特殊文字である。区切り文字は全て意味があるが，スペースを片側または両側におく必要はない。Modula–2 の区切り文字には以下のものがある。

2文字　：＝　<>　<=　>=　..

1文字　＋　－　＊　／　&　.　,　;　(　[　{　^　=　#　<　>　:　)　]　}　|　~

8．予約語

予約語は，常に大文字をとる。

AND	ARRAY	BEGIN	BY	CASE
CONST	DEFINITION	DIV	DO	ELSE
ELSIF	END	EXIT	EXPORT	FOR
FROM	IF	IMPLEMENTATION		IMPORT
IN	LOOP	MOD	MODULE	NOT
OF	OR	POINTER	PROCEDURE	QUALIFIED
RECORD	REPEAT	RETURN	SET	
THEN	TO	TYPE	UNTIL	
VAR	WHILE	WITH		

上のように全部で40個あるが，ワンパスの処理系（FTL Modula–2 はその例）では，さらに

FORWARD

が追加される。これらは，特定の意味以外には転用できない。

9．標準名

全部で29種あり，予約語と同じく大文字で表す。標準名には型名，特殊関数，特殊手続き，特殊文が含まれているが，オーバーローディング[注2)]以外，他の目的に転用できない。

ABS	BITSET	BOOLEAN	CAP	CARDINAL
CHAR	CHR	DEC	DISPOSE	EXCL
FALSE	FLOAT	HALT	HIGH	INC
INCL	INTEGER	MAX	MIN	NEW
NIL	ODD	ORD	PROC	REAL
SIZE	TRUE	TRUNC	VAL	

2．2 Modula–2の構文

Modula–2言語の構文は，点と枝を用いた構文図式やBNF（Backus Naur Form）の変形によって記述できる。[注3)]構文はパーシング（解剖）の対象となる。拡張BNF表現は，そのまま，LALR(1)パーサ表生成ルーチンの対象となるので，以下，拡張BNFを用い，Modula–2第3版の構文を記述しよう。

拡張BNFでは，各プロダクションが

左辺＝右辺

の形をとるが，＝の代わりに::＝が用いられることもある。プロダクションは，

2）オーバーローディング（overloading）とは名前が多様にロードされることをいう。例えば，標準名 **INTEGER** は，**INTEGER** という型を表すこともあれば変換の関数手続きを表すこともある。オーバーローディングは，標準名の他に，演算子記号でも起る。演算子記号の例としては，'＋'は整数の加算，基数の加算，実数の加算，あるいは集合の和を表す。

3） BNF は Algol 60 の制定（Naur が主査）のときには，Backus Normal Form と呼ばれた。BNF の変形として最近，書式つき構文が提案されている。M. Woodman, "Formatted Syntaxes and Modula–2," *Software-Practice and Experience*, Vol. 16 (7) 1986 を参照のこと。

「あぶり出し」と呼んでもよく，左辺をあぶり出せば，右辺になるという意味である。

右辺にバー（'｜'）が用いられると，｜の左項と右項が選択的にとられる意味である。中かっこ（'〔'と'〕'）は省略可能な部分を表す。大かっこ（'｛'と'｝'）はそれにより囲まれた部分が，0回又は複数回繰り返されることを示す。

1．プログラム＝プログラムモジュール｜定義モジュール
2．プログラムモジュール＝〔**IMPLEMENTATION**〕 **MODULE**　名前';'
　｛インポート｝ ブロック　名前'.'
3．定義モジュール＝**DEFINITION MODULE**　名前';'　｛インポート｝
　｛定義｝ **END**　名前'.'
4．インポート＝〔**FROM**　名前〕 **IMPORT**　名前並び';'
5．ブロック＝　｛宣言｝〔**BEGIN**　複文　**END**〕
6．定義＝　**CONST**　｛定数宣言';'｝　｜
　　TYPE　｛名前〔'＝'型';'｝　｜
　　VAR　｛変数宣言';'｝　｜
　　手続き見出し';'
7．宣言＝　**CONST**　｛定数宣言';'｝　｜
　　TYPE　｛名前'＝'型';'｝　｜
　　VAR　｛変数宣言';'｝　｜
　　手続き宣言';'｜
　　モジュール宣言';'
8．複文＝　文　｛';'文｝
9．定数宣言＝　名前'＝'定数式
10．型＝　単純型｜配列型｜レコード型｜
　　集合型｜ポインタ型｜手続き型
11．変数宣言＝　名前並び1':'型

12. 手続き見出し＝ **PROCEDURE** 名前〔仮引き数〕
13. 手続き宣言＝ 手続き見出し';'ブロック 名前｜
手続き見出し';' **FORWARD**
14. モジュール宣言＝ **MODULE** 名前';' {**IMPORT** 名前並び';'}
{エクスポート} ブロック 名前
15. エクスポート＝ **EXPORT** 名前並び';'
16. 名前並び＝ 名前 {',' 名前}
17. 名前１並び＝ 名前１ {',' 名前１}
18. 名前１ ＝ 名前〔'〔'定数式'〕'〕
19. 定数式＝ 単純定数式〔関係 単純定数式〕
20. 単純型＝ 限定名｜列挙｜部分範囲型
21. 配列型＝ **ARRAY** 単純型 {',' 単純型} **OF**型
22. レコード型＝**RECORD** 複フィールド **END**
23. 集合型＝ **SET OF** 単純型
24. ポインタ型＝ **POINTER TO** 型
25. 手続き型＝ **PROCEDURE**〔仮型並び〕
26. 仮引き数＝'(' 〔仮引き数節 {';' 仮引き数節}〕 ')' 〔:限定名〕
27. 限定名＝ 名前 {'.' 名前}
28. 仮型並び＝'('〔仮型〕')' 〔':' 限定名〕
29. 仮型＝〔**VAR**〕〔**ARRAY OF**〕限定名 {';' 〔**VAR**〕〔**ARRAY OF**
限定名〕}
30. 仮引き数節＝〔**VAR**〕名前 {',' 名前} ':' 仮型
31. 列挙＝'(' 名前 {',' 名前} ')'
32. 部分範囲型＝〔名前〕 '〔'定数式'..'定数式'〕'
33. 式＝ 単純式〔関係 単純式〕
34. 単純定数式＝〔'＋'｜'－'〕定数項 {加算演算子 定数項}
35. 関係＝ '＜'｜'＜＝'｜'＝'｜'＞＝'｜'＞'｜'＜＞'｜'＃'｜**IN**
36. 単純式＝〔'＋'｜'－'〕項 {加算演算子 項}

37. 定数項＝　定数因子 {乗算演算子　定数因子}
38. 加算演算子＝'＋'｜'－'｜**OR**
39. 項＝　因子 {乗算演算子　因子}
40. 定数因子＝　限定名｜数｜文字列｜集合｜
　　'('定数式')'｜**NOT**　定数因子
41. 乗算演算子＝'＊'｜'／'｜**DIV**｜**MOD**｜**AND**｜'＆'
42. 因子＝　数｜文字列｜集合｜指示子〔実引き数〕｜
　　'('式')'｜**NOT**　因子
43. 集合＝〔限定名〕'{'〔要素 {','要素}〕'}'
44. 要素＝　式〔'..'式〕
45. 複フィールド＝　フィールド並び {';'フィールド並び}
46. フィールド並び＝〔名前 {','名前} ':'型｜
　　CASE　〔名前〕':'限定名　**OF**
　　バリアント {'|'バリアント}〔'|'〕
　　〔**ELSE**　複フィールド〕**END**
47. バリアント＝〔場合ラベル':'複フィールド
48. 場合ラベル＝定数式〔'..'定数式〕{','定数式〔'..'定数式〕}
49. 文＝〔代入文｜手続き呼出し文｜**IF**文｜
　　CASE文｜**WHILE**文｜**PEPEAT**文｜**LOOP**文｜
　　FOR文｜**WITH**文｜**EXIT**｜**RETURN**〔式〕
50. 代入文＝指示子':＝'式
51. 手続き呼出し文＝　指示子〔実引き数〕
52. 指示子＝　限定名 {'.'名前｜
　　'〔'式 {','式} ']'｜
　　'^'}
53. 実引き数＝　'('式 {','式} ')'
54. IF文＝　**IF**　式　**THEN**　複文
　　{**ELSIF**　式　**THEN**　複文}

〔**ELSE** 複文〕 **END**

55. CASE文= **CASE** 式 **OF** 場合 {'|' 場合} 〔'|'〕
〔**ELSE** 複文〕 **END**

56. WHILE文= **WHILE** 式 **DO** 複文 **END**

57. REPEAT文= **REPEAT** 複文 **UNTIL** 式

58. LOOP文= **LOOP** 複文 **END**

59. FOR文= **FOR** 代入文 **TO** 式 〔**BY** 定数式〕 **DO**
複文 **END**

60. WITH文= **WITH** 指示子 {',' 指示子} **DO**
複文 **END**

61. 場合=〔場合ラベル':' 複文〕

以上，Modula-2の構文を拡張BNFで示した。言語の構文がこれだけ簡潔な仕様で表現できるのはすばらしいが，プログラミングのさいにはいくつかの説明（意味論）が必要である。以下，PascalやCでのプログラミングと比較しながら若干の注意点を述べる。

1．モジュール内部で使う名前は予約語と標準名を除き，すべてはっきりと宣言されなければならない。入出力手続きもIMPORT宣言する。モジュールからEXPORTできるものは型名と手続きである。

2．基本のデータ型には，Pascalのinteger, real, char（Cでは，int, float, charが対応），Booleanの他に，**CARDINAL**と**BITSET**が加わった。CARDINALは符号なしの非負整数を表し，BITSETは0から15（＝処理系に依存する）までの整数の集合を表す。

3．宣言部を書く順番はPascalと異なり任意でよい。あいまい（オパック）型定義を宣言できる。Pascalのような名札（label）宣言はない。したがって，Modula-2にはGOTO文が存在しないが，**RETURN**文，**HALT**文あるいは**EXIT**文の使用によって事が足りる。

4．レコード宣言において，複数個のバリアントを書ける。Pascalでは1

つに制限されていた。

5．ポインタは**POINTER TO**と明示的に書く。Pascalのように，^，@，＃などの記号は無効である。

6．配列宣言は，Pascalよりも融通性がある。**TYPE** COLOR＝（Green, Red, Black）;に対して，**ARRAY** COLOR **OF INTEGER**;のような書き方ができる。

複数個の指標名を宣言するときは

ARRAY〔1..M〕,〔1..N〕**OF** **REAL**;

のように書き，**ARRAY**〔1..M, 1..N〕**OF** **REAL**;とは書けない。

7．関数の宣言は，すべて**PROCEDURE**で統一され，

PROCEDURE 名前（仮引き数）：型名;

とする（FUNCTIONはない）。型名は，殆どすべての型が許されるが，配列やレコード構造は許されない。後者は仮引き数の中にVAR宣言の名前をおけば解決できる。

8．仮引き数の並びにおいては，

PROCEDURE Writeln (string : **ARRAY OF CHAR**);

のように，オープン配列引き数を使える。引き数は，オープンであるから何文字の文字列でもよい。手続き本体では，文字列の大きさは関数**HIGH**（string）によって調べることができる。

9．MODULEにはどのような仮引き数もこない。手続きにだけ仮引き数が現れる。実引き数の並びは宣言した通りの型と個数に厳密に一致しなければならない。Cと異なり厳密にチェックするので可変の長さは許されない。

10．引き数のない関数呼出しでは，括孤を用いてもよいし省略してもよい。

11．RETURN文は手続き内部では引き数なしで用い，関数手続き内部では引き数つきで用いる。例えば，それぞれ，

PROCEDURE A;
BEGIN RETURN END A;

```
PROCEDURE Fac (Cn : CARDINAL) : CARDINAL ;
BEGIN IF n=0 THEN RETURN 1
    ELSE RETURN n*Fac (n-1) END
END Fac ;
```

のようである。

12. IF-THEN文，WHILE-DO文，LOOP文のような構造文（REPEAT文を除く）では，最後に必ず，**END**という予約語をとる。Pascalのように，BEGIN-ENDのブロック文とすると誤りである。

13. 注釈（*…*）は構文のBNFには現れていない。注釈はPascalでは {...} または，(*…*)，Cでは／*…*／となっているが入れ子は許されていない。Modula-2では（*…（*…*）…*）のように入れ子にできる。

以上，Modula-2のプログラム仕様を見てきたが，より詳しくは，本稿末の参考文献〔18〕を参照されたい。

3．Modula-2処理系

市販されているModula-2処理系は汎用計算機からマイクロコンピュータに至るまで各種のものがある。市販価格も数十万円のものから数千円のものまである。

本節は，マイクロコンピュータのCP/MとMS-DOSオペレーティングシステムの上で稼動している，FTL Modula-2コンパイラに焦点をしぼって処理系の機能をみるが，現在広く使用されている他の処理系についても概略を述べよう。

3．1 FTL Modula-2

FTL Modula-2は，オーストラリアのセレンコフ・コンピューティング社（ブリスベン市）のDave Mooreが製作し，米国Workman and Associatesが49.95ドルで販売している製品である。Modula-2の価格としては最も安価で

あるばかりでなく，8ビット，16ビットのパーソナルコンピュータ上で使い勝手やパーフォマンスがよく実行でき，米国BYTE誌の1985年特選ソフトウェアの一つに選ばれている。

FTLは，サイエンスフィクションのFaster Than Light (Lightning，雷の説もある）の頭文字をとったもので，Turbo（Modula–2）を意識している。

FTL Modula–2には，Z80 CP/M対応のものと，MS-DOS対応のものがあり，さらにMC68000対応（Atari社の1040ST）も開発中である。FTL Modula–2処理系の一般的特徴は次の通りである。

1．1パスコンパイラである。通常のModula–2コンパイラは2ないし4パスである。

2．コンパイラに組込まれた強力な画面（スクリーン）エディタがあり，その中でコンパイルが可能である。エディタは，マイクロプロ社のWord Star風のコマンド体系と，Richard StallmanのEMACS (Editing Macrosの略) エディタのアイデアを採用している。

3．それぞれのオペレーティングシステムのネイティブコードを生成する。

4．Modula–2専用のリンカーをもつ（効率はさほどよくない)。

5．Modula–2に対する第3版仕様をほとんどカバーしている（2版で提案された，LONGINT，LONGREALは持たない）上，いくつかの機能の拡張がある。

6．Modula–2処理系の作製は，最初Mooreが自前のPascalを用いて行なったが，後にModula–2自身で書いた。

7．実行時間が高速になるようコンパイラ作成に工夫を凝らした。

8．プロテクトが掛かっていない（しかし，パブリックドメインにはない)。

9．よく書かれたマニュアル（200ページ）をバンドルし，安価である。

10．エディタとツールキットが数千行のModula–2で書かれていて公開されている。30ドルの追加コストで入手できる。

次に，FTL Modula–2コンパイラの拡張機能と制限事項について述べる。

(1) 集合は最大限1024個の要素をとりうる。したがって，CHARの集合も

可能である。

(2) 名前は32文字まで有効で，下線記号（'_'）やドル符号（'$'）を含むことができる。

(3) 変数宣言のさい，初期値を静的に代入することができる。配列要素への代入もできる。例えば，

```
Int : INTEGER =1986 ;
PaiInt : ARRAY〔1 .. 4〕OF  INTEGER =〔3, 1, 4, 2〕;
Chars : ARRAY 〔1 .. 3〕 OF CHAR = 〔'T', 'H', 'E'〕 ;
```

とする。

(4) 型を指すポインタの宣言は，その有効範囲の型の実体を指すので，

```
TYPE Vtype = RECORD
                VChars : ARRAY 〔1 .. 3〕 OF CHAR ;
                VLength : CARDINAL
             END ;
VPtr = POINTER TO Vtype ;
```

のように書ける。

(5) 実数は，すべて8バイトのメモリを使う。第1バイトは指数部をとり，残り7バイトは仮数部をとる。仮数部は2の補数の16進正規化数である。したがって，指数は16^{127}と16^{-128}の間，すなわち，約10^{152}と10^{-153}，仮数部は有効精度が約15桁の10進数を表わすことができる。

(6) オパック型は2バイトに制限されない。

FTL Modula-2 システム全体の使い勝手は良好である。また，C や Pascal とのベンチマークテストを**付録2**で行なっている。

3．2　各種の Modula—2 処理系

まず，汎用の，あるいはマイクロコンピュータには属さない計算機で使われている Modula-2 コンパイラから概観しよう。

(1) IBM 汎用機

スイスの CERN（ジュネーブ）の R. Blake のグループは32ビットの IBM 汎用機に実現したが，公開されていない。

(2) VAX/UNIX システム

英国ケンブリッジ大学のグループが DEC の VAX/UNIX 用に開発した Modula–2 システムがある。日本では，東京大学大型計算機センターの VAX 8600（UNIX）で共同利用ができる。[4]西ドイツのハンブルグ大学でも VAX 用の Modula–2 コンパイラが作製され，有償で配布されているので各国で利用されている。

(3) M 68000 /UNIX ワークステーション

米国タータン研究所（ピッツバーグ）は M 68000 /UNIX ワークステーション（Appolo DOMAIN や SUN–3）用の Modula–2 を1000ドルで販売している。

次に，マイクロコンピュータ上で定評のある Modula–2 コンパイラを概観する。

(4) 16ビット計算機上で走る Modula–2（PC-DOS, MS-DOS オペレーティングシステム）

4．1）LOGITECH Modula–2/86

米国ロジテック社（レッドウッド市）の製品で，1986年に第2版を89ドルに引下げて（初版は495ドル）販売している。IBM PC 互換機を想定しているが，パッチを行えば NEC PC 98シリーズの MS-DOS でも稼動する（実際，日本の NJK 社は，パッチをあてて，48,600円で販売している）。4パスのためコンパイル・実行時間がかかるが，多機能のモジュール，ライブラリを用意し利用者が最も多い。独立ユーザーズグループ（IUGLM）が形成され，*Modula–2 Chronicles* を隔月に刊行している。

4．2）Modula Corporation (Modula–2，Mac Modula–2)

4）木下佳樹「Modula–2 コンパイラ mc の紹介」，東大大型計算機センター，センターニュース，18巻4号（1986年4月）を参照のこと。

米国 Modula 社（プロボ市）の製品で PC(MS)-DOS 用と Macintosh 用の2種類をともに，150ドルで販売している。Modula–2 の完成時，Wirth の助手であった Richard Ohran（ブリンガムヤング大）の力が与って大きい。旧版は4パスで Lilith と同じ M- コードを生成しインタプリートしていたため効率が悪かったが，1985年の新版では，ネイティブコードを生成し改善した。アップル社マッキントッシュ用の Mac Modula–2 は，機種依存を積極的に採用し，グラフィックシステムの作成によく使われている。

4．3）Interface Technologies 社の M2SDS(-XP)

米国のインターフェーステクノロジー社（ヒューストン市）が M2SDS(-XP) (XP はエクステンションパッケージの略）を80.88ドルで販売している。生成コードが M コードのため，システム開発というよりは，Modula–2 学習用といえる。

4．4）Volition Systems Modula–2

UCSD p– システム（カリフォルニア大学サンディエゴ校）から派生した，先駆的な Modula–2 システムだったが，現在は，研究グループ（代表は Richard Gleaves）の離散により停止している。

4．5）FTL Modula–2

3．1 で述べた。

(5)　8ビット計算機で走る Modula–2（CP/M オペレーティングシステム）

5．1）Hochstrasser Computing Modula–2

スイスの Hochstrasser Computing 社（チューリッヒ市近郊）が CP/M 80 用に製作し，160ドル（420スイスフラン）で販売していたが，現在は，他社の利用を推せんしている。

5．2）Turbo Modula–2

CP / M 80 上で走る米国ボーランドインターナショナル社のベータテスト（出荷前のテスト）品であるがベンチマークテストや紹介記事も現れていて，かなりの波紋がありそうである。

5．3）Modula–2 / 09

OS- 9上で走る日本製（日本ソフトバンク販売）の1パスのModula- 2コンパイラであり，クラスや日本語が使えるなど特徴がある。使い勝手が悪く学習用。

5．4）FTL Moduld-2

3．1参照。

4．Modula-2における新しいソフトウェア概念の意義

Modula-2は，1970年代の後半に主として小型計算機のシステムの処理系を効率的に開発するために設計された汎用のプログラム言語である。Pascalのような教育用，入門者用の言語ではなく，新しいソフトウェア概念を含む実務用，専門家用の言語である。

本節では，Modula-2に特徴的なソフトウェア概念である，オブジェクト指向（オパック型），データ抽象，並列処理等が，情報処理のさいどのように位置づけられるかを考察する。

4．1　オブジェクト指向・データ抽象

プログラム設計の新しい方法論の一つとして最近注目を浴びている概念に「オブジェクト指向設計」がある。SmalltalkやMesa[5)]は「オブジェクト指向言語」といわれ，実際，Modula-2はMesaのアイデアをかなり借用している。「オブジェクト」とは，抽象データ型，つまり，その構造がユーザから見えない形になっているものを押し進めたデータ構造である。それには，せまい意味の「オブジェクト」と広い意味の「オブジェクト」がある。前者としては，例えば，標準名REALは指数部が何ビット，仮数部が何ビットとっているかはシステム側だけが見えていて，ユーザー側からは見えないので抽象データ型といえるが，Modula-2の抽象データ型は，オパック型またはプライベー

5）Smalltalkについては，鈴木則久「Smalltalk」（産業図書，1986），Mesaについては，James Mitchell et al, *Mesa Language Manual*, Ver. 5.0 Xerox CSL-79-3, 1979. を参照。

ト型をいう。オパック型は，定義モジュールで曖昧に型宣言をし，実現モジュールの中で具体的に記述するものをいう。この方法によって実現モジュールの方で型の変更があっても再コンパイルするのは，その実現モジュールだけでよい。抽象データ型の利点は，情報システム（プログラム）設計の当初，具体的にデータ構造を定義することなくプログラミングをし，開発の途上で具体的にデータ構造を与えればよいことになり，プログラムの信頼性や保守性あるいは実行効率を促進すると期待できる。

次に広い意味での「オブジェクト」概念を考えよう。Modula–2は，MesaやSmalltalkのようなオブジェクト指向言語ではなく，汎用の手続き言語である。両者を区別するものは，手続き言語がプログラムをデータ構造と制御コード（手続き，文）に分割するのに対し，オブジェクト指向言語はプログラムをインスタンスとメソッドに分割し，言語はその一部にしてしまう点である。ここで，インスタンス（実体）は，内部状態を動的に記憶するためのデータ領域で，RECORD構造に対応する。他方，メソッド（方法）は，インスタンス変数を呼び出す手続であり，PROCEDUREとPROC変数を用いて記述できる。Modula–2のMODULE概念は，オブジェクト指向言語では「クラス」といわれる。Modula–2によってオブジェクト指向の概念を実現する利点は，ソフトウェア開発においてコードを共有化し，開発と保守を容易にし，ユーザーインターフェイスを柔軟にできることにある。例えば，Logitech Modula–2/86プログラム環境の一連のデバッガー（ポストモーテムダンプや実行時デバッガー）はすべてこのオブジェクト指向アプローチにより開発された。

Modula–2は，このデータ抽象とモジュラー化の採用によって，信頼性・安全性・保守性等で大きな利益を得る。反面，若干効率性の犠牲を伴うのは止むを得ないといえる。

4．2　手続き型

Modula–2では，Pascalのような単純型，配列型，レコード型，ポインタ型に加えて，Mesa言語より手続き型を追加した。手続き型は値として手続き

のアドレスをとる。手続き型の変数を導入すると，役に立つ応用例は多数ある。コンパイラ作成中，ある手続きを不定回実行したいとか，木構造を使って整列や探索をする場合，便利である。

手続き型の使い方として，独立した変数として使う場合と，手続きの引き数として使う場合がある。

手続き型の引き数の型は明示的に宣言されねばならない。例えば，

```
TYPE  Proc 1 = PROCEDURE (INTEGER);
      Proc 2 = PROCEDURE (VAR INTEGER ; REAL);
      procrand = PROCEDURE ( ): REAL;
      procstring = PROCEDURE (ARRAY OF CHAR);
```

のように書く。いま，RANDOM が関数手続きとして宣言されており，

```
VAR   pr : procrand ;
      a : REAL ;
```

とすると，

```
pr : = RANDOM ; (* pr には手続き RANDOM のアドレスが代入される*)
a : = pr( ) ; (* pr の指す手続きを呼ぶ，従って a : = RANDOM ( ) ; と同じ*)
```

手続き型の別の使い方として，手続き変数を仮引き数として用い，それから手続き名を実引き数として渡す場合がある。例えば，

```
PROCEDURE ex (p : PROC) ;
  ...
ex (Example)
ex (AnotherExample) ;
```

のように使う。ここで PROC は引き数のない手続きに対する，手続き型，すなわち，

```
TYPE PROC = PROCEDURE ;
```

であるように予め定義された標準名である。

この場合の使用例としては，実時間処理，並行処理等が挙げられる。

4．3　並行処理，コルーチン

Modula-2がPascalと異なる重要な特徴のひとつは並行処理（コンカレントプロセシング）あるいは多重処理（マルチプロセシング）を行える点である。そのために，コルーチン（並行ルーチン）と同じ意味でプロセス概念が導入された。しかしModula-2は単一のプロセッサの計算機を対象とするのでコルーチンは一度にひとつの単一プロセッサを稼動させるプロセスのことである。すなわち，Modula-2ではC. Strachey の発明した時分割処理（TSS）を容易に実現するためのツールを与えたといえる。TSSでは処理能力の高い1台の計算機が端末を介して計算機本体と比べ処理能力の劣る人間とあたかも一対一であるかのように交信をする。複数のCPUをもった並行処理では実時間の多重処理が可能であり，あたかも複数の処理が同時に行われるという状況を生み出す。

Modula-2は直接にMesaのプロセスを継承し，Adaのような高機能の並行処理機能をもたない。Modula-2処理系のコルーチンへの対応は全般に制限されている。コンカレント処理と割込み処理はコルーチンを通じてプログラムされる。コルーチンはモジュールSYSTEM（または，Processes）からインポートされるシステム依存の手続きを使うので処理系依存の部分が多い。しかし，以前にはアセンブラでしか書けなかったコルーチンが今やModula-2で書けるようになったのは大きな進歩である。

FTL Modula-2のコルーチン対応の概略を述べる。プロセスを処理するためのモジュールはProcessesにあり，完全なプロセス切換え機能をもっている。ユーザーは，

```
FROM Processes IMPORT StartProcess, SIGNAL, Init,
                      Send, Wait, NEWPROCESS, TRANSFER;
```

の形で呼び出す。プロセスの生成のためにStartProcessを使う。StartPro-

cess は2つの引き数をもつ手続きで，与えられたプロセスを起動するために呼ばれる：

```
StartProcess (Proc1, 100);
StartProcess (Proc2, 100);
```

これは，以前に定義されている引き数をもたない手続き Proc1 ， Proc2 をそれぞれヒープ領域から100バイトずつヒープ領域を破壊することなく実行することを示す。SIGNAL は型で

```
VAR  sig : SIGNAL;
```

のように宣言し， Init (sig) ; Wait (sig) ; Send (Sig) ; のように使う。

手続き NEWPROCESS はプロセスの動的な生成に， TRANSFER はプロセス間の制御の切換えを容易かつ柔軟に行う低水準のルーチンである。しかし，ほとんどの Modula–2 処理系は，モジュールの優先度が常に1となっているためコルーチン間でのやりとりはできない。Modula–2 は準並行処理のための簡単なコルーチン機構を与えているだけで，ユーザは自分でプロセススケジューリングやメモリ管理ルーチンを用意しなければならない。

5．結語的覚書

汎用の算法型の高級言語に関する限り，Algol ，FORTRAN ，COBOL は，1960年代の， PL / I ， Pascal は1970年代の核言語であり，1980年代は，中・小型の Modula–2 ，大型の Ada となると予測されて久しい。Pascal は，データ構造とプログラム構造はすっきりしているので情報処理教育や小規模のシステム作成には歓迎され成功をおさめてきた。しかし，設計者の Wirth 自身が明言するように， Pascal は複雑な現実的なシステムプログラムの世界で使用すると無理が生じる。

Modula–2 は，Pascal や Modula–2 の設計・開発・使用の反省から生まれ，1980年代半ばになってようやく使用人口が急増した言語である。その理由は言語自身の問題というよりは使用環境の社会経済的変化にあると思われる。具体的には， Modula–2 の対象とする中・小型計算機システム側の受入れ態勢，

Modula–2 処理系の進歩，価格・性能比の向上，そして Modula–2 書籍の増加である。

中型計算機としては今日，ワークステーションの目覚ましい発達があり，そこでの核言語が模索され，Modula–2 はその一候補となりつつある。また，小型としては，いわゆるマイクロコンピュータの個人用，業務用の量的かつ質的な増加があげられ，学習用からシステム開発まで自由に使用できるようになった。同時に各種の Modula–2 処理系を利用するユーザーズグループが多数結成され，会誌や書籍の増加は指数的ですらある。[6]一時期，Modula–2 に関する文献の少なさが指摘されたがその心配は解消されつつある。しかし，これだけでは Modula–2 が情報処理の真の核言語とはなり得ない。Modula–2 にはまだ国際標準規格がなく，ユーザーは止むなく Wirth の第 3 版 [18] の仕様を基礎とする。そこには彼の好みから初等的な関数，例えばグラフ表示で頻繁に使う四捨五入（ROUND）関数が省かれ，入出力関数も極めて初等的なものしかない。多数の Modula–2 言語の使用者の声を反映する規格が早急に作られねばならない。そのさい，Modula–2 言語が中小型計算機用のシステムプログラム作製を目指した仕様と，情報処理教育ないし小規模の（目安としては 2 千行以内のソース）プログラム作製を目指し Pascal を吸収した仕様の 2 段構えになってもよいだろう。

Modula–2 は全体が小さくまとまっている上に強力な言語である。情報処理の核言語となるためにはなお一層使用されねばならない。

6）Modula–2 に関する成書の部分リストは参考文献の [17] を除く，[1] から [20] である。[6] は最もわかりやすい。Modula–2 の名前の入った雑誌・新聞として，*Journal of Pascal, Ada & Modula–2* (John Wiley & Sons) と *Modula–2 Chronicles* (Namir Shammas) 等が刊行されている。

付録1. Modula—2で書いた入出力モジュール

```
DEFINITION MODULE AdHocIO;
(* Ad Hoc I/O Routines *)
PROCEDURE Write(c: CHAR);
PROCEDURE WriteLn;
PROCEDURE WriteString(s: ARRAY OF CHAR);
PROCEDURE Read(VAR c: CHAR);

PROCEDURE WriteIn(s: ARRAY OF CHAR);
PROCEDURE WriteInt(i: INTEGER; spaces: CARDINAL);
PROCEDURE WriteCard(c, spaces: CARDINAL);

END AdHocIO.

IMPLEMENTATION MODULE AdHocIO;
(* Ad Hoc I/O Routines *)
IMPORT Terminal;
VAR PowerOfTen: ARRAY[0..4] OF CARDINAL=[1,10,100,1000,10000];

PROCEDURE places(c: CARDINAL): CARDINAL;
(* returns the number of places c takes to print *)
VAR i: CARDINAL;
BEGIN
  FOR i:=4 TO 0 BY -1 DO
    IF (c DIV PowerOfTen[i]) > 0 THEN RETURN i+1
    END
  END;
  RETURN 1
END places;

PROCEDURE WriteNum(c, spaces: CARDINAL; neg: BOOLEAN);
VAR i:INTEGER; p: CARDINAL;
BEGIN
  p:=places(c);
  FOR i:=1 TO INTEGER(spaces)-INTEGER(p) DO Terminal.Write(' ');
  END;
  IF neg THEN Terminal.Write('-')
  END;
```

```
  FOR i:=INTEGER(p)-1 TO 0 BY -1 DO
    Terminal.Write(CHR(c DIV PowerOfTen[i])+ORD('0'));
    c:=c MOD PowerOfTen[i]
  END
END WriteNum;

PROCEDURE WriteIn(s: ARRAY OF CHAR);
BEGIN  Terminal.WriteString(s); Terminal.WriteLn
END WriteIn;

PROCEDURE WriteInt(i: INTEGER; spaces: CARDINAL);
BEGIN
  IF i < 0 THEN
    WriteNum(CARDINAL(-i), spaces-1, TRUE)
  ELSE
    WriteNum(CARDINAL(i), spaces, FALSE)
  END
END WriteInt;

PROCEDURE WriteCard(c, spaces: CARDINAL);
BEGIN WriteNum(c, spaces, FALSE)
END WriteCard;

PROCEDURE Write(c: CHAR);
BEGIN Terminal.Write(c)
END Write;

PROCEDURE WriteLn;
BEGIN Terminal.WriteLn
END WriteLn;

PROCEDURE WriteString(s: ARRAY OF CHAR);
BEGIN  Terminal.WriteString(s)
END WriteString;

PROCEDURE Read(VAR c: CHAR);
BEGIN  Terminal.Read(c)
END Read;

END AdHocIO.
```

付録2． FTL Modula—2の比較優位性

同一の計算機上でいくつかのコンパイラを計算速度，使用資源，価格等について比較することを「ベンチマークテスト」という。多数の類似のコンパイラの中からユーザはどの製品を入手・使用すればよいか迷うとき，このベンチマークテストは有用な情報となる。しかし偏った情報をもたらすことも多いのでその使用には注意が必要である。

ベンチマークテストの例題としては，「エラトステネスの篩による素数生成プログラム」や「騎士の漫遊パズル」，「行列の乗算プログラム」，「三角関数プログラム」等が使用されてきた。また，**リスト1**に示すような「機能別ベンチマークテスト」は，Modula–2コンパイラー間の比較に役立つ。

以下では，「エラトステネスの篩による素数生成プログラム」（**リスト2**）について，FTL Modula–2コンパイラをBDS C (Ver. 1.50a)とTURBO Pascal (Ver. 2.0)の各コンパイラと比較してみる。

実験は，NEC PC–8801 mk Ⅱ (4MHz)の56 K CP/M (CP/M 2.2版，CBIOS 3.0版）上で，ストップウォッチを用いて行なった。

周知のように，CP / Mでは，高級言語Cといえば，BDS C，Pascalといえば，TURBO Pascalが指名されるほどパーフォーマンスのよいコンパイラである。

下表からわかるように，FTL Modula–2は実行時間に関して最良の成績をあげている。しかし，コンパイル時間や生成コード数では，TURBO Pascalの方が優れている。BDS Cはさらに，使い勝手でも劣る。

Modula–2は，上のような小さなプログラムよりは大きなプログラムで効果が発揮されシステム設計の効率の向上を期待できる。FTL Modula–2は使い勝手もよいので，さらに，リンク時間が節減でき，コードの最適化が可能なら，強力なソフトウェアツールとなるであろう。

表　素数生成のベンチマークテスト

コンパイラ名	コンパイル時間	リンク時間	実行時間	生成コード数	価格
BDS-C	35秒	25秒	1分22秒	FD0	150 (ドル)
TURBO Pascal	3秒	0秒	51秒	12C	69.95
FTL Modula-2	22秒	80秒	34秒	E6F	49.95

リスト1．機能別ベンチマークテスト

```
MODULE Bench;
(*ST-
  a: empty REPEAT loop
  b: empty WHILE loop
  c: empty FOR loop
  d: CARDINAL arithmetic
  e: REAL Arithmetic
  f: standard function
  g: array of single dimension
  h: same as g but with index tests
  i: matrix access
  j: same as i but with index tests
  k: call of empty, parameterless procedure
  l: call of empty procedure with 4 parameters
  m: copying arrays (Block moves)
  n: pointer chaining
  o: reading of file
*)
FROM Storage IMPORT ALLOCATE;
FROM Terminal IMPORT Read, BusyRead, Write, WriteLn;
FROM Smallio  IMPORT WriteCard;
FROM Files IMPORT FILE, Lookup; (*, ReadWord, Reset, Response; *)
FROM Maths IMPORT SIN, EXP, LN, SQRT;
TYPE NodePtr = POINTER TO Node;
     Node = RECORD x,y: CARDINAL;
                   next: NodePtr;
            END;
VAR A,B,C: ARRAY(0..255) OF CARDINAL;
    M:     ARRAY(0..99),(0..99) OF CARDINAL;
    m: CARDINAL;
    head: NodePtr;
PROCEDURE Test(ch: CHAR);
 VAR i,j,k: CARDINAL;
     r0,r1,r2: REAL;
     p: NodePtr;
  PROCEDURE P;
  BEGIN
  END P;
  PROCEDURE Q(x,y,z,w: CARDINAL);
  BEGIN
  END Q;
  BEGIN (* Test *)
   CASE ch OF
    "a":  k:=20000; REPEAT k:=k-1 UNTIL k=0;
    "b":  i:=20000; WHILE  i>0 DO i:=i-1 END;
    "c":  FOR i:=1 TO 20000 DO END;
    "d":  j:=0; k:=10000; REPEAT k:=k-1; j:=j+1; i:=(k*3) DIV (j*5)
                          UNTIL  k=0;
```

```
      "e":  k:=5000; r1:=7.28; r2:=34.8;
            REPEAT k:=k-1; r0:=(r1*r2)/(r1+r2) UNTIL k=0;
      "f":  k:=500;
            REPEAT r0:=SIN(0.7);r1:=EXP(2.0);r0:=LN(10.0);r1:=SQRT(18.0);
             k:=k-1
            UNTIL k=0;
      "g":  k:=20000; i:=0; B[0]:=73;
            REPEAT A[i]:=B[i]; B[i]:=A[i]; k:=k-1 UNTIL k=0;
      "h":(*$T+ *) k:=20000; i:=0; B[0]:=73;
            REPEAT A[i]:=B[i];  B[i]:=A[i]; k:=k-1 UNTIL k=0 (*$T- *);
      "i":  FOR i:=0 TO 99 DO FOR j:=0 TO 99 DO M[i,j]:=M[j,i] END END;
      "j":  FOR i:=0 TO 99 DO FOR j:=0 TO 99 DO M[i,j]:=M[j,i] END END;
      "k":  k:=20000; REPEAT P; k:=k-1 UNTIL k=0;
      "l":  k:=20000; REPEAT Q[i,j,k,m]; k:=k-1 UNTIL k=0;
      "m":  k:=500; REPEAT k:=k-1; A:=B; B:=C; C:=A UNTIL k=0;
      "n":  k:=500; REPEAT p:=head; REPEAT p:=p^.next UNTIL p=NIL; k:=k-1
               UNTIL k=0;
      "o":  k:=5000; (* REPEAT k:=k-1; ReadWord(f,i)
                        UNTIL k=0; Reset(f)  *)
    END  (* CASE *)
  END Test;
  VAR ch, ch1: CHAR;  n: CARDINAL; f: FILE;
      reply: INTEGER;   q: NodePtr;
  BEGIN
    Lookup(f, "BENCH.MOD", reply);    (* From Files.SYM *)
    head:=NIL; n:=100;
    REPEAT q:=head; NEW(head); head^.next:=q; n:=n-1
    UNTIL n=0;
    Write(">"); Read(ch);
    WHILE ("a"<=ch)&(ch<="p") DO
      Write(ch); WriteLn; n:=0;
      REPEAT n:=n+1; Test(ch);
       IF (n MOD 50)=0 THEN WriteLn END;
       Write("."); BusyRead(ch1)
      UNTIL ch1<>0C;
      WriteCard(n, 6); WriteLn; Write(">"); Read(ch)
    END;
    Write(14C)
  END Bench.
```

リスト2．素数生成プログラム

```
MODULE PrimeNumber;
 (* Benchmark Example: Eratosthenes Sieve Method *)
 FROM Terminal IMPORT WriteString, WriteLn;
 FROM Smallio IMPORT WriteCard;
CONST  nSize = 8190;
VAR Flags: ARRAY[0..nSize] OF BOOLEAN;
    i, prime, k, iter, count: CARDINAL;

BEGIN
  WriteString(' 10 iterations.'); WriteLn;
  FOR iter:=1 TO 10 DO
    count:=0;
    FOR i:=0 TO nSize DO Flags[i]:=TRUE END;
    FOR i:=0 TO nSize DO
      IF Flags[i] THEN
        prime:=i+i+3; k:=i+prime;
        WHILE k<= nSize DO
          Flags[k]:=FALSE; INC(k, prime)
        END;
        INC(count)
      END
    END
  END;
  WriteCard(count, 5); WriteString(" primes.")
END PrimeNumber.
```

参考文献

[1] J. Beidler and P. Jackowitz, *Modula-2*, Prindle, Weber & Schmidt Pub., 1985.
[2] Paul M. Chirlian, *Introduction to Modula-2*, Weber System, 1984 (武市哲也監訳, アスキー出版局, 1986).
[3] Kaare Christian, *A. Guide to Modula-2*, Springer-Verlag, 1986.
[4] G. Ford and R. Wiener, *Modula-2, A Software Development Approach*, John Wiley & Sons, 1985.
[5] R. Gleaves, *Modula-2 for Pascal Programmers*, Springer-Verlag, 1984 (阿江忠訳, CQ出版社, 1986).
[6] Edward J. Joyce, *Modula-2 : A Seafare's Manual and Shipyard Guide*, Addison-Wesley, 1985.
[7] I. Kaplan and M. Miller, *Modula-2 Programming*, Hayden, 1985.
[8] John W. L. Ogilvie, *Modula-2 Programming*, McGraw-Hill, 1985 (中村和郎監訳, マグロウヒル, 1986).
[9] Lewis Pinson, *Introduction to Computer Science with Modula-2*, John Wiley & Sons, 1985.
[10] Arthur Sale, *Modula-2 Discipline and Design*, Addison-Wesley, 1986.
[11] Russell L. Schnapp, *Macintosh Graphics in Modula-2*, Prentice-Hall, 1986.
[12] H. Schildt, *Modula-2 Made Easy*, Osborne / McGraw-Hill, 1986.
[13] D. Thalmann, *Modula-2: An Introduction*, Springer-Verlag, 1985.
[14] B. Walker, *Modula-2: Programming with Data Structures*, Wadswoth, 1986.
[15] R. Wiener and R. Sincovec, *Software Engineering with Modula-2 and Ada*, John Wiley & Sons, 1984.
[16] R. Wiener and R. Sincovec, *Data Structure in Modula-2*, John Wiley & Sons, 1986.
[17] Niklaus Wirth, *Modula-2*, ETH Berichte no. 36, 1980.
[18] N. Wirth, *Programming in Modula-2*, 3rd ed. Springer-Verlag, 1985.
[19] N. Wirth, *Algorithms and Data Structures*, Prentice-Hall, 1986.
[20] 中村和郎,「Modula–2 文法入門」CQ出版社, 1986.