1. ALGOLの入出力実行命令について

牛島和夫　有田五次郎（九州大学）

はじめに

ALGOL 60は，周知のように入出力命令を各implementorに委ねている（少くとも1964年5月まで）．したがつて入出力命令をどのような形でALGOLプログラムの中にその文法と出来るだけマサツのないように組込むかということはALGOL 60 で規定した文法をコンパイラで実現することよりも，ある意味でやつかいな作業である．即ち，文法の実現は，既に仕様の定まつたものをいかに実現するか－解析－に重点がおかれるのに反して，入出力命令の場合は，(1)プログラマーに使いやすく，(2)使用可能な入出力機器をかなりの程度まで有効に活用でき，(3)ALGOL文法にできるかぎり違反しない，ような仕様を作り上げることが，いわゆる狭い意味のALGOL コンパイラーづくりとちがつたむずかしさをもつていると考える．

一方実用的なコンパイラーを作成する際には，計算機の容量，速度によりおのずから文法に制限が加わるのはやむを得ないし，当然な処置であると考える．ところが入出力装置は制限された文法で取扱うには豊かすぎる場合の方が多く，文法にのらないという理由でrejectするのは本末転倒であろう．そこでいわゆる標準手続きなる概念を導入して，制限された文法では律し得ない部分を補なつて入出力装置の利用を図つてきていると考えられる．

この報告では，我々が九州大学計算センターに与えられた機器構成と制限されたALGOL文法の中で，入出力実行命令という形で，入出力装置を計算依頼者にどのように開放し又開放しようとしているかを報告したい．

I. 入出力実行命令の最小構成

I.§1　機器構成とALGOL本体

九州大学計算センターに与えられた主な条件は，次の2点である．

(1)　第1表に示すようなハードウエアの機器構成．

(1−1)　容量8Kは一見大きいようであるが1語1命令で，データ用の記憶場所を考えれば，オブジエクトプログラム用としては大きいとはいえない（ALGOL文法の豊かさに比較して）．

(1−2)　入出力機器は比較的豊富である（ALGOL 60の入出力の貧弱さに比較して）．

(2) ALGOL程度のソフトウエアがなければ，計算機を使用できない多数の計算需要者．この2点を勘案して，九州大学 ALGOLが成立した．主な流れ図は第1図に示すとおりである．我々はこの範囲内で(2)の条件を満たす計算者が最小限計算機とcommunicate できる入出力実行命令の仕様を作り上げた．

I.§2 INPUT

データはカードにパンチすることにして，入力命令は下記のように決定した．

READ(<list>)

ここで

<list>::=<simple variable>|<subscripted variable>|
<simple variable>,<list>|<subscripted variable>,<list>

例

READ($A, B, C[I, J], D, E[K+1]$)

ただしA, B, Dはinteger，real又はBoolean型のsimple variable，C, Eは<type> arrayで宣言されているものとする．

データの書き方はstandard formatを採用した．下記の約束を設ける．

(1) Revised ALGOL 60 2.5 Numbers 2.5.1 syntaxに許される形全部．

(2) separatorをコンマとする．

(3) 1枚のカード上で／（スラント）パンチ以後のパンチは無視する．

(4) 理論値はTRUE又はFALSEとパンチする．

計算実行中（第1図 phase 5）にFORMAT指定によりデータカードを読むこと（on line，off lineいずれでも）は，FORMATを作成し，データをカードからarrangeするための管理プログラムのスペース，その作業用番地，又かんじんなFORMATなどが記憶場所を相当ふさぐ．illegal cardが存在した場合に計算の続行が無意味になる．又プログラマーの立場からいえば，FORMATプログラムの作成は，計算機になれないものにとつては，かなりやつかいな代物である．このような理由で上の約束が選ばれた．

以上の条件を勘案して我々はコンパイルの流れの中に，データカードを前処理するphase 3を導入した（第1図）．phase 3 では，カードをカードリーダから読みとりその時間内に10進2進変換を行い型の情報と共に磁気テープに書くことにした．適当なblockingを必要とするので67語を1 blockとし第2図のような構成をもつバッファにデータが充たされるとテープに1 recordとして書き込む．データはinteger，real又はBooleanのいずれかであるから，それらを01，10，11 で表わすものとすれば，1語に21個のデ

ータに対する型の情報を盛り込むことができる．したがつて，1 block の長さは，磁気テープ装置の速度にも関係するが，それより計算実行時に使用しうる主記憶装置の大きさから1+21×3+3=67語が選ばれた．phase 2，phase 3で磁気テープ#2に書かれるrecordは第3図のようになり，phase 4では，磁気テープ#2よりオブジエクトプログラム1，2，…，nとデータ1を主記憶装置に格納することになる．

phase 1では

READ(A,B,C)は

```
jump      read routine
store          A
jump      read routine
store          B
jump      read routine
store          C
```

のようにコンパイルされているので，phase 5のREAD管理プログラムでは，indexのcountを1ずつあげて，型の一致を確かめつつバッファの内容をAレジスタにload することが行われる．indexが63に達すれば，データを磁気テープ#2から更に補充しておく．

以上phase 3のプログラムは10進2進変換，カード読み込み用作業番地，磁気テープ書き込み用の67語の作業番地を含めて500語を要した．又phase 5 の管理プログラムはバッファ67語，プログラム60語を費している．管理プログラムは第4図に示すとおりである．図中，型の不一致をERRORとしているのは，コンパイラーが混合演算を許さないためである．

この方式によれば，すでに述べたほかに

(1) 計算実行時に127語という比較的小さい記憶容量を占有するだけですむ．

(2) 計算実行以前（phase 3）で表現のあやまりのあるカードをrejectし，無用の計算に入ることを防ぐことができる．

(3) 入力媒体がカードから紙テープにかわつてもphase 3 の処理プログラムをとりかえるだけで十分である．

などの諸長所が考えられる．短所としては，

(1) カードの切れ目が失われるためにプログラムにとつて一度illegal dataが存在すると以下の計算が全部だめになるおそれがある．

(2) separatorとしてコンマを用いなければならないので，1枚のカードに収容しうる情報は高々40である．

(3) データ自身がreal，integer又はBooleanの型の情報をもつていなければならない．

(4) 文字が読めない.

(1)については，READ(<list>)の終の)とカードの切れ目を対応させ型の情報を 3 bit にしてカード上最後のデータであるか否かの情報を指定することも考えられるが，実行は考えていない.

(2) separatorにコンマではなくACM I／O案(文献[1])のstandard formatで提案しているように相続くk個以上のスペースをseparatorと考える方法もあるがこの場合は更に一枚のカードにもりこめる情報の数は少くなる．しかし科学技術計算の場合，カード一枚当りの情報は少くしデータの変更を容易ならしめる方向もあるのではないだろうか.

このALGOLコンパイラーは，動的割付を許していないので，IFIP案(文献[2])のinarrayに相当するものはあまり意味がないと考えるが，ステップ数を非常に多く有するfor statementを用いてarrayの要素を読みこむことを考えると記憶容量節約の点では意味があるかもしれない.

I. §3 OUTPUT

OUTPUTについては，実行命令の形と，ACM案による書き方をそえるにとどめる.

PRINT(n, A)	output1(6, "$B-.nD_{10}-ZDB$", A)
PRINT(n, I)	output1(6, "$B-(n-1)ZDB$", I)
PRINT("××··×")	output0(6, "<title format>")
SPACE(I)	対応するものがない.
CRLF (I)	output0(6, "／") 但しCRLF(1)の場合

format処理プログラムが主記憶装置を大きく占有しては困ることは，INPUTの場合と同じ事情なので，有効ケタ数を可変にすることを許すほかは，きまつたformatのOUT−PUTのみとした. ここでプリンタ上に曲線をプロットすることは，邪道かもしれないが，結果を直視する上からかなり有効な場合がある．ACM案でSPACE(I)(Iは変数)に相当することを行わせるには，LAYOUT及びLISTについてかなり面倒な手続きを要すると考えられる．SPACEのような手軽な命令がほしいように考えられる.

Ⅱ. 入出力実行命令の拡張

Ⅱ. §1　拡張の検討

Ⅰに述べたものは，ACM案が出される以前に計画されすでに実現されてしまつたものであり，その上入出力の最小限の要求を満たすにすぎなかつた．ところが我々の利用し得る機器構成は，本体こそ８Ｋであるが，中間結果を保存するためのカードせん孔機はまだ利用していないし，記憶容量の不足を補うべき磁気テープ＃１，磁気ドラムは計算に直接利用されていなかつた．すでに歩き出したREAD PRINTについてもACM案を参照して，プログラマーに対しては，変更という形ではなく，拡張という形で，提供するために検討を加えた．

(1) 磁気ドラムは12800語なので頻繁に使用するシステムプログラム用に確保する．

(2) カードパンチはPRINTプログラムとほとんど同じ処理でありプログラム容量のみが問題である．

(3) FORMAT指定のREADがほしい．

(4) 磁気テープ装置を外部記憶装置として，ACM案put及びgetの可能性を探る．

Ⅱ. §2　PRINT及びPUNCH

PRINTに関しては，fix型のformatの導入とREADのAlpha formatに対して，S formatを導入する．PRINTに対応する命令は，SPACEを除いて全てPUNCHにも可能にした．OKITAC 5090Hに特徴的なプレビアスSCCを利用して，現在の出力命令がPRINTであるか，PUNCHであるかを検知して，処理プログラムが，該当する出力命令と合致するものならば，そのまゝ，さもなければ，現在のOUTPUTプログラムをドラムに退避させ，ドラムから，合致する処理プログラムを本体の同じ場所に読み出して実行する．従つて出力媒体の増加は，本体における管理プログラムスペースの増加にはならなかつた．

Ⅱ. §3　FORMAT指定のREAD

ALGOLのソースプログラムの中にFORMATをいれることは，Ⅰで述べたように，記憶容量の許すところではない．しかし短所の項で述べたように１枚のカードを80欄十分に使いたい方面，特に１個体に対する色々の情報，例えば，調査の回答，診断の結果などを１枚のカードに納めることが出来るならば，計算以外の目的にも都合がよいといつた計算需要者の要求があり，計算センターでは，需要開拓の意味からもFORMAT指定READを考えることにした．

FORMATのsyntaxはACM案に従うことにして，それをどのphaseで行うかが問題となるが，解答は，上の要求そのものが示唆している．即ち我々はphase 3 に対して，FORMATプログラムをcontrol cardの形で与える方法をとつた．勿論前章に述べたstandard formatも非常に使い易いので，混用できることにしてstandard format

とformat指定をcontrol cardで交代可能にした．したがつてデータカードと総称されるものは，例えば第5図のような構成をもち，FORMATプログラムとDATAプログラム（文献[3]）が交互にあらわれる．なおphase 3の初期状態は，standard formatになつており，これまでのプログラムに対しては何等の変更もないようになつている．

control cardは

第1欄　　＊

第2～7欄　FORMAT

と約束することにする．FORMATプログラムは1枚のカード上でつくせるとはかぎらないので，2枚目以後のcontinuationは第1欄の＊で知ることにする．即ちデータカードのうちFORMATプログラムに属するカードは第1欄に＊をパンチすると約束する．

処理プログラムの概要

FORMATプログラムカードであることを検知したらcontrol cardのつづくかぎりよみとつて，（の次からFORMATプログラムを処理し構成されたformatがカード1枚分に達すれば，データプログラムカードを，完成したformatに従つて処理する．今度は，FORMATプログラム自身がreal，integer又はBooleanの情報をもつているので，その情報と10進2進変換されたデータをバッファに書いてゆき，充たされたらI.§2の方法に従つて磁気テープ＃2に1 recordとして書いてゆく．

以上の処理からもわかる通り，計算実行時に於ける入力命令には全く変更を加える必要がない．

control cardがSTANDARD FORMATであれば，以下次のcontrol cardがくるまでは，standard formatでデータプログラムカードを処理する．

ここで同じFORMATプログラムをもつデータ群が別々の組になつている時は，同じFORMATプログラムを再び別のデータ群に対してもその直前に与えなければならず，ソースプログラムにFORMATを書いた場合のようにFORMAT プログラム名を指定するだけですますというわけには行かなくなるが，これはたいした障害にはならないと思われる．このように，ALGOL本体のプログラムに比して，machine orientedなFORMATプログラムをALGOLプログラムから分離してしまい，初歩的なプログラマーもFORMATプログラムは，エキスパートに分担してもらい全体のプログラムを完成することも可能になると考えている．

II.§4　磁気テープ装置の利用

ACM案は次のように述べている．

「中間結果を外部機器（external devices）に確保するために，さらに2つの手続き，put，getを導入する．例えば

put(100, LIST)

という実行命令は，LISTと名付けられたlist procedureで規定される値の組を100というラベルをはつて，外部機器に一時格納する．もう一つの

get(100, LIST)

という実行命令は，上で格納した値を再び主記憶装置に呼びもどす．外部機器としては，デイスク，磁気ドラム，磁気テープなどがあるが，機器の選択，データが記録されるformatなどについては，プログラマーの関与するところではない．」

put，getの詳細な仕様については更に別の個所で

(1) put(n, LIST)

(nはinteger parameterでvalueで呼ばれLISTはlist procedureの名)は，nというラベル(identification number)をつけてLISTによつて選ばれる値の組を一時格納する．既に同じnというラベルをつけて格納されている組があれば更新され古いデータは失われる．

(2) get(n, LIST)

は先にputによりラベルnをつけて格納されていた値の組を主記憶装置に呼びもどす．その際，LISTによつて順序づけられる要素とnで格納されていた値とは，その型について同じ順序で並んでいなければならない．型の異る場合は換算函数が介入する．

(3) get(n, LIST)

において，LISTできめられる要素の数が，同じnのラベルで格納されている値の組の要素の数より少い場合は，数の一致したところまでとり出される．多い場合には，未定義(undefined)となる．

(4) get(n, LIST)

の命令によつて，外部機器内の値は，変更を受けない．

ACM案が述べている所は以上で尽されるが，既に完成しているコンパイラーの中にlist procedureのわり込む余地はないので，LISTの部分は，一種のopen subroutine（READの<list>ですでに行つている）の形をとることにして，以下のような実行命令を書くことにする．

PUT(<label name>, <list 1>)

GET(<label name>, <list 1>)

value<label name>;

integer<label name>;

I. §2で定義した<list>を拡張して<list 1>は次のように定義する．

<list 1>::=<array identifier>|<list>|
<array identifier>,<list 1>

我々に与えられた磁気テープ装置2台のうち1台は入力テープとして既に使用しているので，ACM案のいわゆる外部機器としては，磁気テープ＃1が利用できる．

さてOKITAC 5090HのINSTRUCTION MANUALによれば，磁気テープ関係の命令は，

(1) 可変長の記録が可能である．

(2) header label指定の読みとりができる．

(3) header label指定の書きこみができる．

となつている．ここでheader labelは1 record上の最初の12 bitがその役を持ち，これをそのまゝ<label name>に利用すれば，磁気テープ装置1台当り4096通りの<label name>を許すことができることになる．

さてデータを1 recordとしてテープに記録したり読み出したりするためには，少くとも1つの一連の主記憶装置のメモリーを必要とする．又各値に対して型の情報を与えなければならない事情はREADの場合と全く同じである． しかしこのバツファはREADのバツファと共用するわけにはゆかないため別に設けなければならないので，READと同様67語を割当てることにして構成も第2図のようにした．PUT及びGETは全て計算実行中（phase 5）であるがphase 5に入る直前に，磁気テープは

(FM) head (FM)

の順にならんでいるものとする．

PUT管理プログラムの概要（第6図）

PUTがOPENされると<label name>をバツファの先頭番地の頭から12 bitに格納する．<list>の要素を型の情報と共にバツファに送り込み，indexのcountが63に達すれば，要素数63をバツファの先頭番地に記入し，<label name>をheader labelとしてtapeを探索する．指定されたlabelがtape上にあれば，そのrecord の上に現在のバツファの内容を書き込む．headはそのrecordの次にある．header labelがなければend of fileになるのでFMの上から新たに記録し，そのうしろに改めてFMを記録する．headはこのFMの直前にもどしておく．<list1>がまだつきなければ再び<list1>の要素をバツファに送りこみ，充ちれば，同上のheader labelで同上の処理を行う．充たずにPUTがCLOSEされれば， 現在バツファに送り込まれた要素数をバツファの先頭に記入して，同上の方法でtapeに記録し，管理ルーチンを脱出する．

GET管理プログラムの概要（第7図）

GETはheader labelを探索してバツフアに読み取り<list 1>の要素に与えてゆく，その際に型の一致を検査し，バツフアがつきてその語数が63であれば，さらに同じheader labelでバツフアに読みとり同じように行う．バツフアがつきてその語数が63未満ならば，未定義になる．

以上でわかるように，我々のREADは，磁気テープ#2に対するheader label 0の広い意味のGETであるといえよう．

PUT及びGET管理プログラムは，バツフアを含めて264語を要した．全てのプログラムがPUT，GETを用いるわけではないのでこれらを使用するjobについてだけ264語が使用される．又<list>の定義の拡張により，inarrayに関するREAD 命令も可能になつた．これでREAD管理プログラムは60語から52語に減少した．

（例） tapeが第8図(1)の状態にあるとき

PUT(h_1, LIST1)

（LIST1の要素数＝63×4＋5）

の実行後 tapeは(2)のようになる．さらに

PUT(h_3, LIST2)（要素数＝15）

PUT(h_1, LIST3)（要素数＝63×5＋20）

の後 tapeは(3)のようになる．さらに

PUT(h_1, LIST4)（要素数＝63＋25）

を実行すると tapeは(4)の状態になる．

この例からheader labelを介して，いもづる式recordになつていることが知れよう．

II. §5　金物からみたPUT，GET

前節でプログラムの構想は出来上つたが，金物が受け付けなければ机上の空論に終つてしまう．金物からみて次のような問題がある．（これはOKITAC 5090Hの磁気テープ装置のチヤンネルコントロール及びTo型ハンドラーのみにいえることかもしれないが，筆者らの磁気テープ使用経験はこの型についてしかないのでおゆるしを乞う）

§4に述べたINSTRUCTION MANUALの仕様のうち，

(1) header label指定の読み出しは正しく実行する．

(2) header label指定の書き込みについては，header labelを探索して，headが第9図aの位置まできて再びbの位置にもどりそこからはじめて書き込みを行う．このときheadの逆進がわずかに12ビット分なので既に書いてあるrecordと多少ずれることもありうる．又記録される語数が同じだとしてもcの位置がわずかずれることも

あり，次のrecordの先頭dにnoiseを与えることもありうる．

(3) (2)で書きこみERRORを生じたときにそのrecordは抹消されなければならない．

以上主としてPUTの際に生ずる問題でGETで読みとりERRORが生じたときはやむを得ないし，正しく書かれたrecordならば読み取りでERRORを生ずることはないといつてよい．

以上の解決策として，

(1) header label指定の書き込みは行わずそれと同じ結果をもたらす別の方法を用いる．即ちheader labelの部分のみ読みとり正しく1 record分IRGまでheadをもどしてから書きこむ．

(2) (3)のnoiseを生ぜしめない程度にIRGを十分広くとる．即ちend of fileになつて新しく記録する際にはそのrecordの後をm回eraseすることにする．

erase命令はtape上の約3インチの記録を抹消するのでmをいくらにすればよいかが問題になる．そこで次のような予備実験を行つた．

（実　験）

最初に1 recordをi語として，相異るheader label $h_1, h_2, \cdots, h_8$をもつrecordをFM，h_1 record, erase m回，h_2 record，erase m回，$\cdots$，h_8 record,erase m回，FM

の順に書き，常にheadを先頭のFMにもどして，

header,label指定読みとり

backward IRG

header label無指定書き込み

を$h_1, h_2, \cdots, h_8$について行いこれをn回くり返す．

結果は第2表のようになるが，読みとりでERRORが生ずるようであれば，書き直しによつてnoiseが生じていたことになるし，何回まで可能であるかを知ることができる．なお最初の書き込みでERRORが生じた場合は，tapeのその部分にキズがあると考えて，eraseを行わせて書き込むので，そのheader labelに先んずるrecordにはさらにIRGがふえていることになる．したがつて終了までの所要時間についてはtapeの状態にもよるので，一がいに結論できない．

II. §6　計算時間からみたPUT, GET

§4に述べたGETの際の解決策(1)ではtapeの始動，停止の時間が無視できないものになつてくるが，今はtapeへの記録密度の上から概算してみると

67語　　約93mm

本来のIRG　　約19mm

1回のerase　　約75mm

したがつてeraseをm（≧1）回行えばmacroの記録密度は減少する．よつてmacroにみた伝送時間もほぼ記録密度に逆比例すると考えられるから

$$t \propto \frac{93+19+75\times m}{93+19} = 1+\frac{75}{112}\,m$$

が大雑把にいえる．

次に読み書きheadの位置について，プログラマーが外部記憶装置として，磁気テープ装置を用いることを関知していれば，REWIND BACKSPACE のような，実行命令を用いて，あらかじめ予想されるPUT(n, list) 又は，GET(n, list)に応じてheadの位置をプログラムの進行順序に最も都合のよい位置に置くことができよう．しかしACM案の思想によれば，機器についてはプログラマーの知る所でなく，管理プログラムとしては，今のところ適当な1ケ所にheadを待機させておくより方法がないと考えている．

II. §7　将来の問題

現在の磁気テープ装置は，phase 1 でのスクラッチテープと共用しているために無理であるが，更に磁気テープ装置を増設することができれば，chain jobに似たことも考えてみたい．即ちjob 1 でputしたdataをjob 2 でいきなりgetするというようにして磁気テープもカードのように中間結果保存に使えるようになれば更に便利であろう．この場合ALGOLプログラムとしては，job1とjob2を2つの並行なblock として含むような更に大きいbegin endの対を考えれば，この面では文法違法はおこらない．

以上表記の題目のうち，READを含めて，ALGOLにおける磁気テープ装置利用を中心に述べてきた．このうちPUT，GETについては，11月15日現在プログラムが出来上つて日が浅いために，使用実績が十分でなく，erase 回数も安全のため $m=2$ を選んでいる．さらにデータを累積した上で報告したいと考えている．

最後にII §4の実験プログラムを作成していただいたOBM九州大学駐在保守員太田靖彦氏，IIに関するプログラムの実現に協力していただいたOBMプログラム課鈴木茂雄，村越重正の両氏に深く感謝する次第である．

［文　献］

1. D. E. Knuth (Chairman)
"A Proposal For Input-Output Conventions In ALGOL 60"
Comm. of ACM Vol 7, No. 5, May 1964

2. ISO/TC97/SC5(IFIP－5)53
"Report On Input-Output Procedures For ALGOL 60"
Apr. 1964

3. A. J. Perlis　"A Format Language"
Comm. of ACM Vol 7, No. 2, Feb. 1964

4. 沖電気工業株式会社　"OKITAC5090H INSTRUCTION MANUAL"
Nov. 1963

第1表　九州大学計算センター機器構成（OKITAC－5090H）

中央演算処理装置	
語の構成	純2進　42ビット，1語1命令
容量	磁気コア　8192語
サイクルタイム	同期式　10μs
演算速度	固定　浮動
	加減算　30μs　85μs(60μs)
	乗算　250μs　250μs
	除算　390μs　330μs
補助記憶装置	
磁気ドラム	1台　12800語
磁気テープ	1チャンネル2台
	伝送周波数　10Kc
	記録語長　可変
入出力機器	
電動タイプライタ	1台　450桁／分
光電式テープリーダ	1台　200桁／秒
ラインプリンタ	1台　500行／分
カードリーダ	1台　80欄　500枚／分
カードリードパンチ	1台　80欄　150枚／分

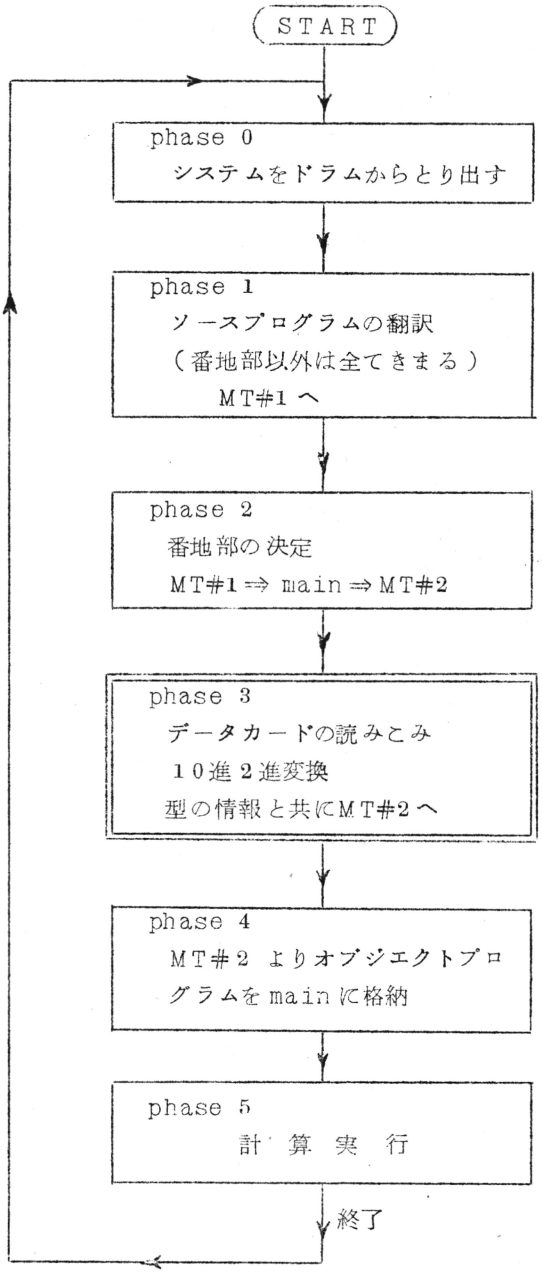
第2表　所要時間の表

READ 回数	ERASE 回数	記録語数 $i=64$				
n	m	1	2	3	4	5
3	0	25	e	25	25	e
	1	35	37	35	35	36
	2	45	45	46	47	45
4	0	32	e	32	32	32
	1	45	45	45	45	45
	2	57	57	57	57	57
5	0	e	e	e	e	e
	1	53	53	53	53	53
	2	68	68	68	68	68
10	0	e				
	1	96				
	2					

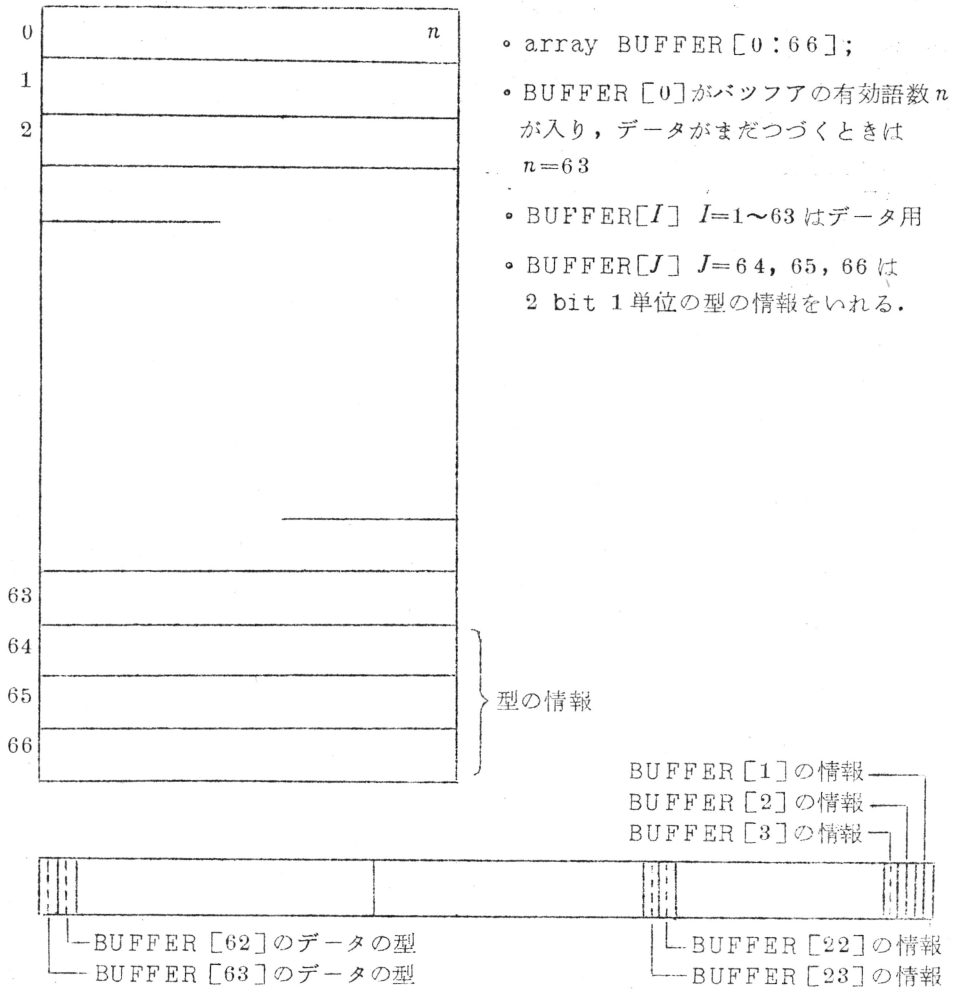
$i=128$

n	m	1	2	3	4	5
3	1	52	53	53	53	53
	2	63	63			
4	1	65	65			
	2	79	79			
10	1					
	2					
20	1					
	2					

第１図　九州大学ALGOLの流れ図

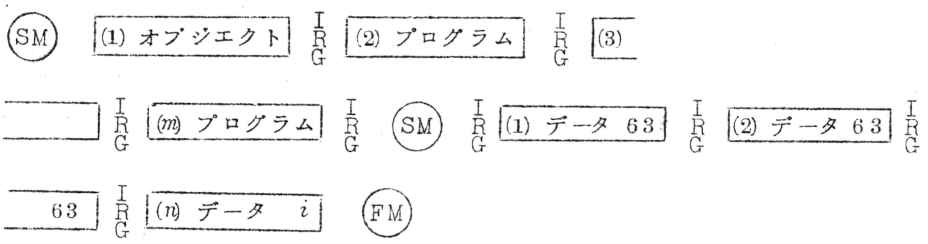


第2図　リードデータバッファの構成



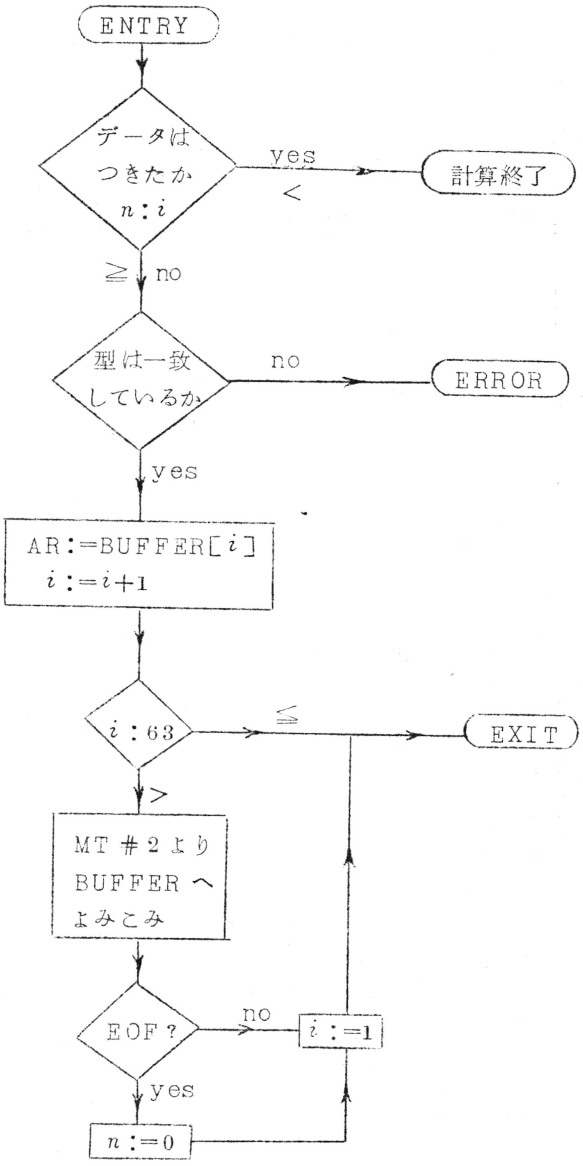
- array BUFFER [0:66];
- BUFFER [0]がバッファの有効語数 n が入り，データがまだつづくときは $n=63$
- BUFFER[I] $I=1\sim63$ はデータ用
- BUFFER[J] $J=64, 65, 66$ は 2 bit 1単位の型の情報をいれる.

第3図　　MT#2 の構成

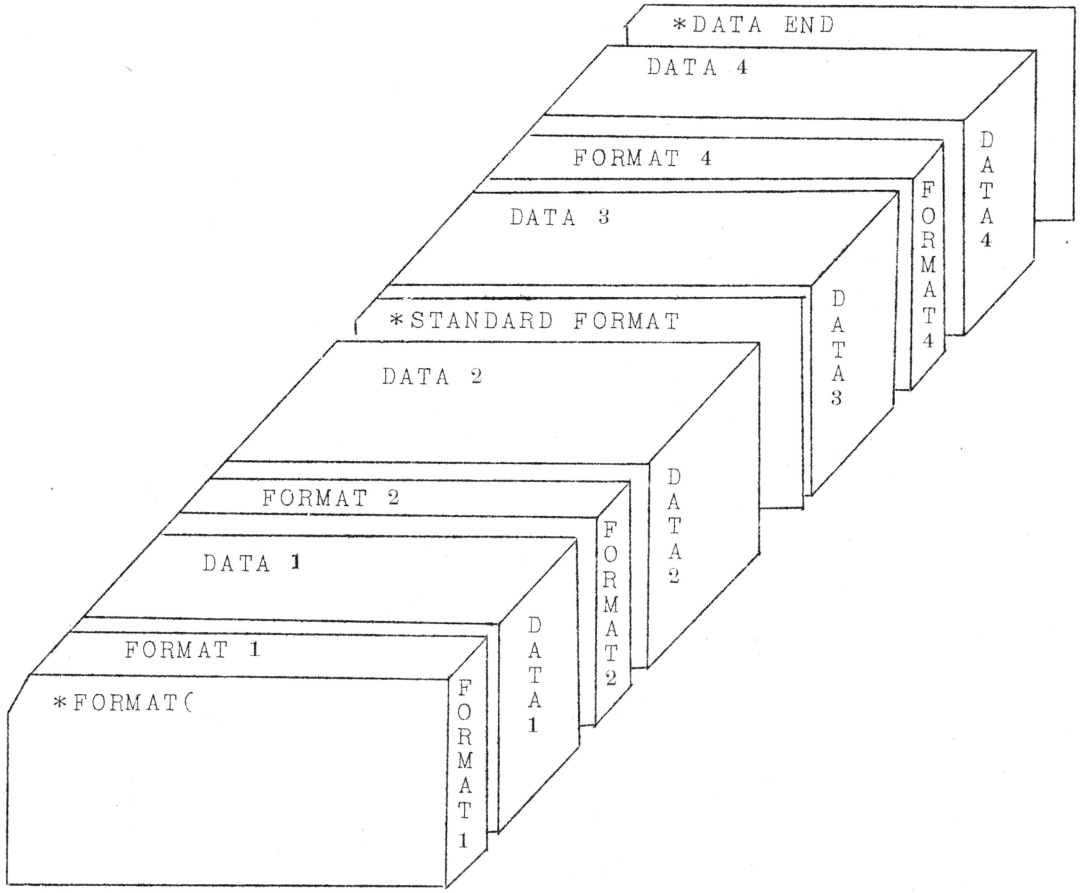


SM は segment mark
FM は file mark

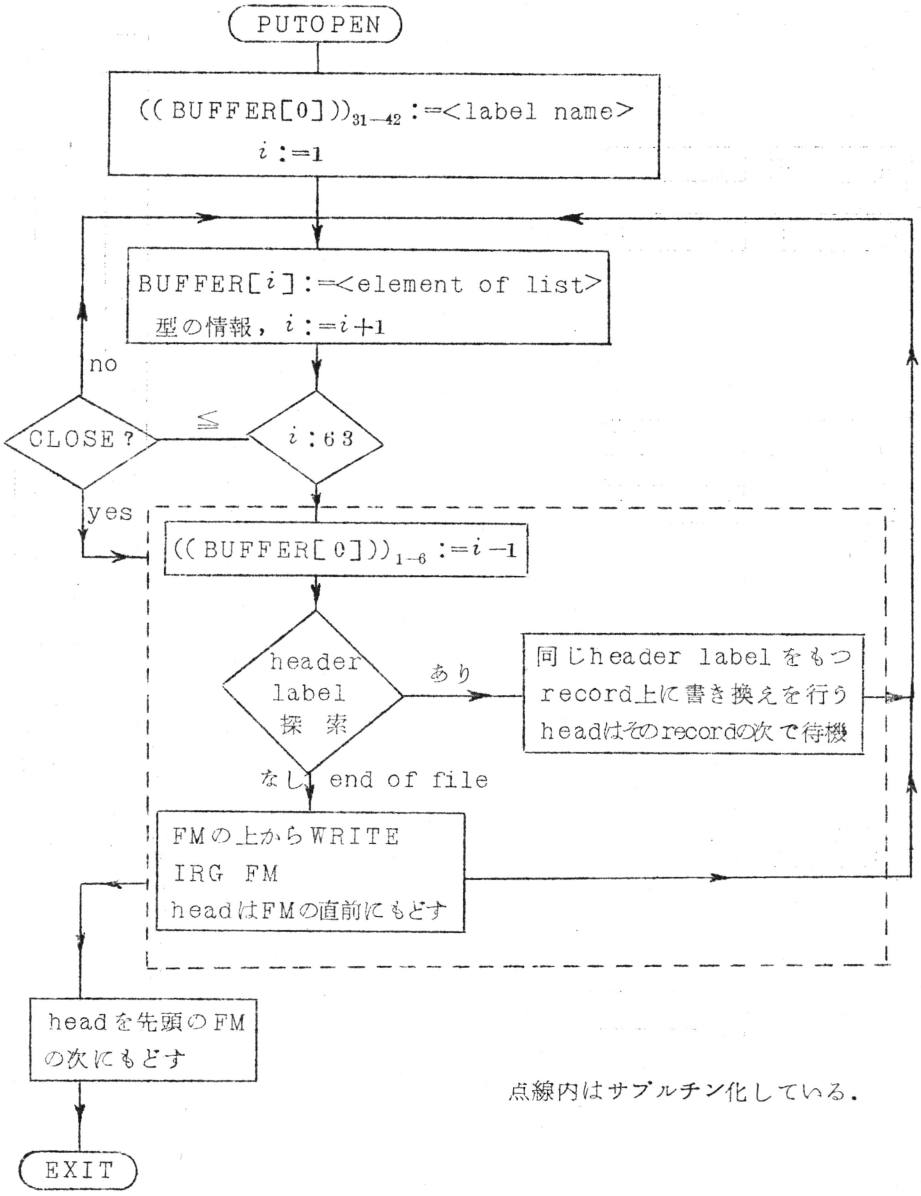
第４図　　データリード管理プログラム



第５図　データカードの構成

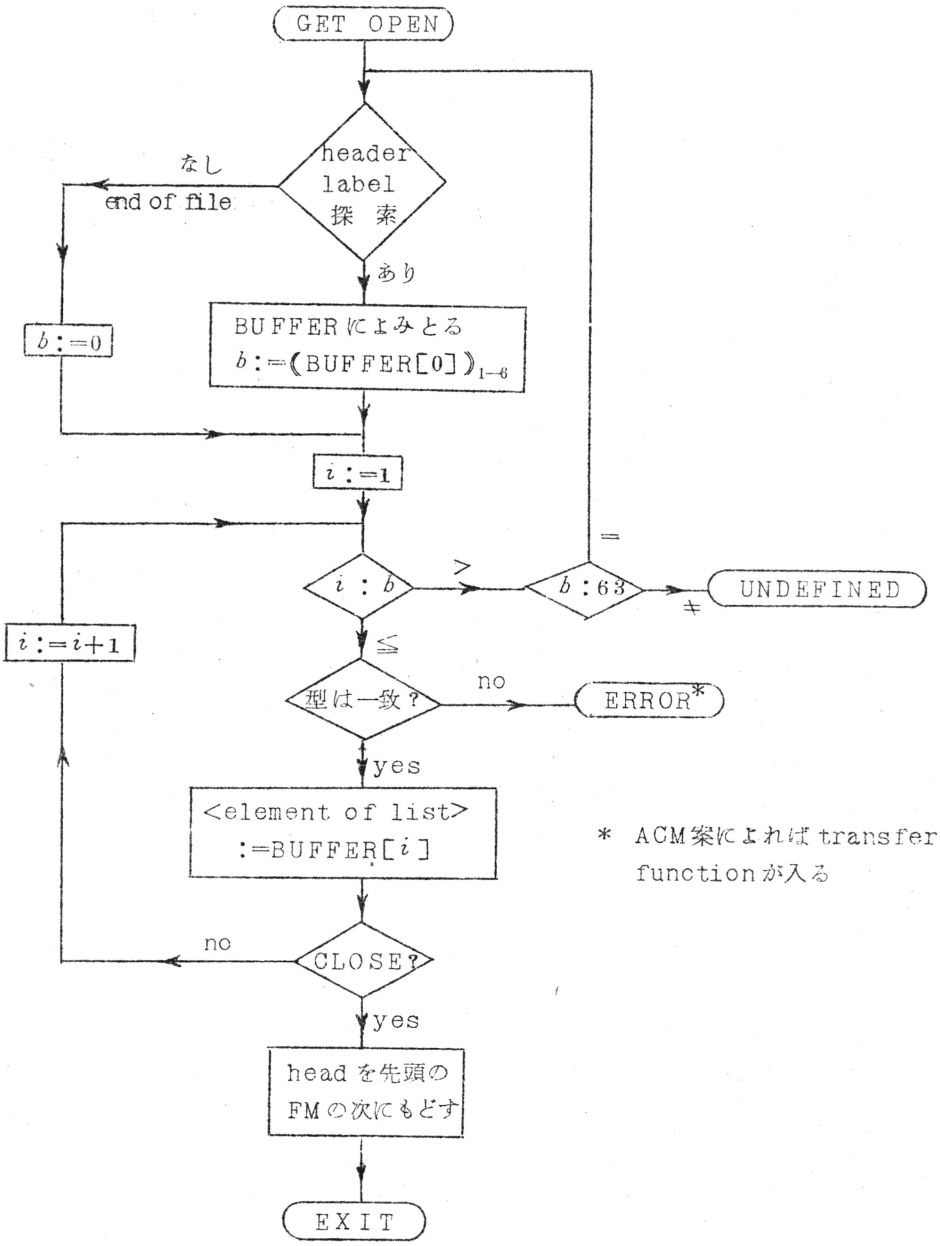


第６図　PUT管理プログラム



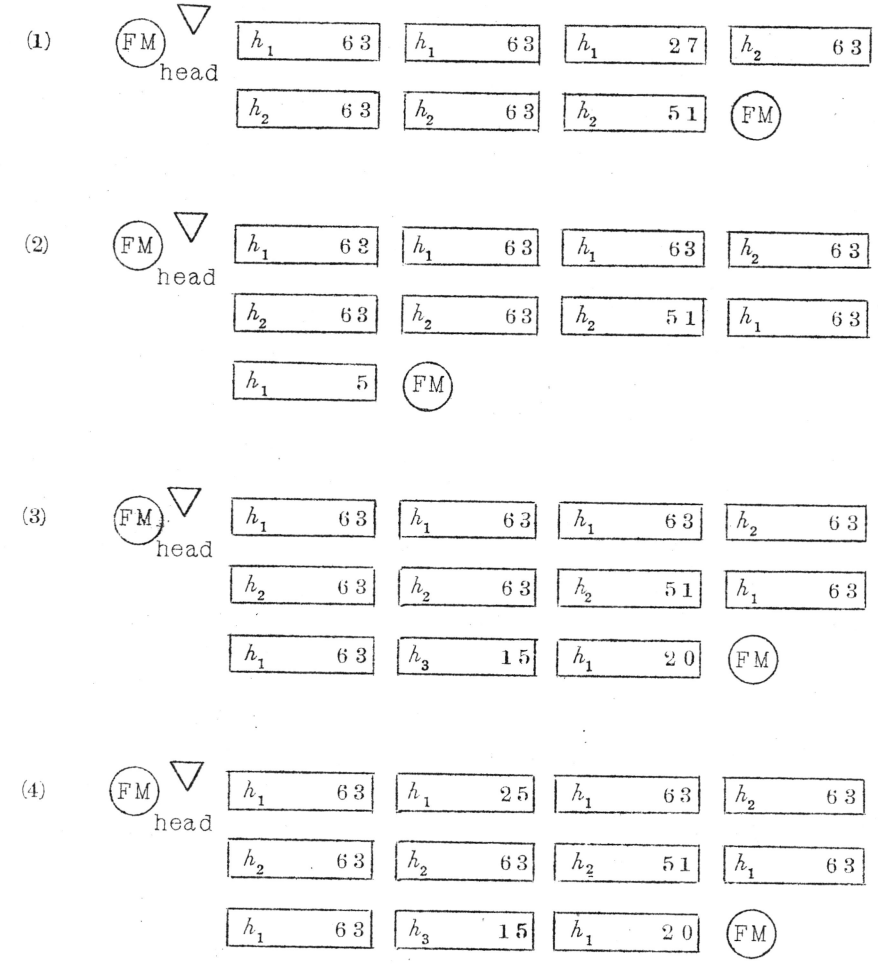
点線内はサブルチン化している．

第7図　GET管理プログラム

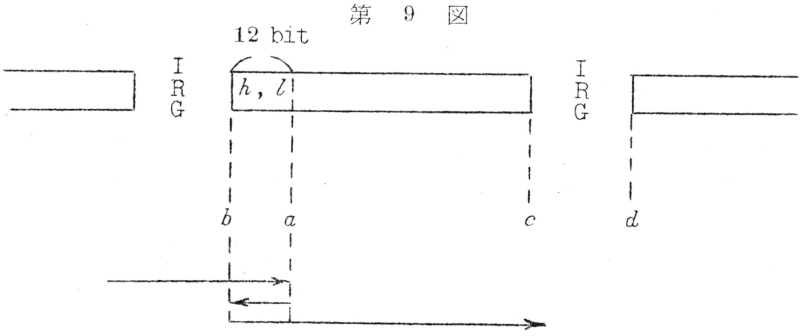


* ACM案によればtransfer functionが入る

第８図　　ＰＵＴ の 例



第　９　図



本 PDF ファイルは 1965 年発行の「第 6 回プログラミング―シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html