2．字句解析部の高速化について

—ソフトウェアの調整技法—

東京工業大学 総合情報処理センター

白濱　律雄・前野　年紀

要旨

Pascalコンパイラの字句解析部を改良して、処理時間を約35％削減した。

コンパイラ自身を翻訳するために要するCPU時間のうち、半分を字句解析部が消費していた。この字句解析部は、一文字毎に或る手続きを呼び出し，それが原始プログラムを一字ずつ入力して、各種の検査・作業のうえで，呼び出し側に渡す、という構造をしていた。

ところが、よく調べてみると、文字の数の分だけ実行されるにもかかわらず、文字単位の入力は遅く、手続き呼び出しもかなり手間がかかる。

そこで、一行単位で読み込み、各種の検査はデータ構造で吸収してしまうことにより、一文字毎には手続きを呼び出す必要がない形に改造したところ、字句解析部の処理速度は約3.5倍に向上し、プログラムもすっきり仕上がった。

overallで35％短縮

かつては、空白の読み飛ばしに時間がかかっていたが、改良により、ほとんど目立たなくなった。

§0　背景

PASCAL 8000[3,4]は、Nägeli のTrunk コンパイラをもとに、疋田、石畑[5]らによって開発され、更にA.A.E.C.(Australian Atomic Energy Commission)で改造されたコンパイラである。Pascalは、言語がよくまとまっていて、手元の処理系も安定して動いていることから、重宝している。

コンパイラ自身がPascalで書かれているため、保守・改造も難しくない。約6,300行の原始プログラムを翻訳するCPU時間は、約14秒である。しかし、TSSを介して作業する時の能率を考えると、翻訳速度の改善が望まれた。

この種の処理系では、空白の読み飛ばしに時間がかかっていることが多いことから[6]、測定してみたところ、全処理時間の半分を字句解析部が消費していることがわかった。

そこで、コンパイラ保守用の道具の部品としても使えるものなので、字句解析部を作り直すことにした。従来、字句解析部の作り方として、オートマトン化は論じられても、本文中で試みたような入力インタフェースを扱ったものは少ない。

一行入力という標準Pascal[2]にはない機能を利用したが、構造もすっきりしたものが出来上がり、処理速度も約3.5倍に向上した。

以下の順に沿って報告する。

第一部では、AAEC版の字句解析部（以後旧版と呼ぶ）の機能を説明する。第二部では、その構造と測定結果を記し、問題点を指摘する。第三部で改善案を挙げ、具体化された字句解析部（以後新版と呼ぶ）を説明し，その測定結果を旧版のものと比較する。第四部は、今回得られた知見のまとめと、苦言・提言の部である。

この研究には、東京工業大学総合情報処理センターのHITAC M-180を使用した。

§1 Pascalコンパイラの字句解析部の機能

1) 字句解析部の一義的機能

字句解析部は、外部からの入力文字に対する前処理を受け持っている。

一般に言語は各々固有の語彙を有するが、現実の計算機で使用可能な文字集合には限りがあり、そのために、言語の一つの記号が複数の文字で表現(金物表現)される場合には、逆変換の必要があるからである。

また、名前や数は、言語の定義では個々の文字からなる複合体であるが、構文解析では単体として扱うので、まとめて一つの記号とする。注釈や空白、順序番号等、構文解析に不要な情報を除外する仕事もある。

字句解析部は、通常多数の文字を扱うので、性能についての配慮が必要である。

2) Pascalプログラムの金物表現

入力である原始プログラムの形式の定義は、付録を参照されたい。入力中には、行末・入力末を意味する文字が含まれているとし、各々eol, eofで表わす。

原始プログラムは拡張記号の列である。拡張記号とは、構文解析の要素である「記号」と「分離子」である。記号は、名前、区切り語、数、文字列、区切り記号のいずれかである。数、名前及び区切り語の相互の間には、分離子を置かねばならない。分離子とは、空白、注釈、行末のいずれかである。

区切り語は名前と同じ形をしている。区切り語と同一の名前は使用できない。(予約語方式である。)

行はリスティングのための出力単位であり、構文上の意味はない。一つの記号が二行以上にまたがることはない。

注釈本体の頭部にあり、通貨記号(¥)で始まる翻訳選択子列により、選択子の指定は随時更新される。そこで選択子Uが指定されたならば、解除されるまで、各行の73文字目以降は無視される約束になっている。

定義から、次の二点がいえる。

(i) 拡張記号は、先頭の一文字で大分類可能である。

(ii) 区切り記号の長さは、高々二文字までである。

3) 字句解析部の副次的機能

PASCAL 8000の字句解析部の、他の部分との関係を図1に示す。

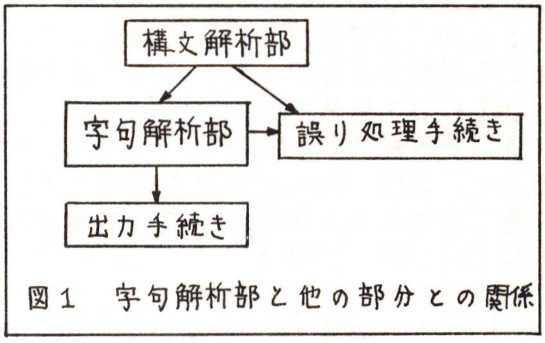

図1 字句解析部と他の部分との関係

字句解析部の第一の機能は、記号の切り出しである。他にも以下の作業をする。

①リスティングの準備と行末の検出。原始プログラムを表示する指定があった場合、または、誤りが発見された場合には、行の内容を表示する。そのために、字句解析部は、入力した文字を出力バッファに保存し、次の行を読む直前に「出力手続き」を起動する。

②誤り位置の保存。行中での文字の位置を明らかにしておく。翻訳中に誤りが発見されると、「誤り処理手続き」が呼ばれ、誤りの位置と種類が保存される。行の内容を出力した後、誤りの発見された位置に、"@"と誤りの種類が表示される。

③入力末の検出。構文的に正しいプログラムの処理中には起きないことであるが、入力末に達した時には、翻訳を終らせる。

§2 PASCAL8000の字句解析部の構成

1) 内部仕様

PASCAL8000では、§1で述べた諸機能を、二つの手続きINSYMBOLとNEXTCHとに分担させている。

INSYMBOLは、構文解析部から呼ばれ、記号の切り出しと翻訳選択子の値設定を行なう。他の全ての作業はNEXTCHで行なう。

①NEXTCH

NEXTCHは、INSYMBOLの要求に応じて、外部から一文字ずつ入力して渡す。また、並行して以下の作業を行なう。

(i) 余地があることを確認しながら、リスティングのために、入力した文字を出力用バッファに入れる。このバッファのインデクスが、誤り検出時に保存される。

(ii) U選択子への対応。論理的な行の長さ(72またはバッファ長)を越える部分については、行末まで、読み込んで(i)の作業は行なうが、INSYMBOLには渡さない。論理的な行の長さは、INSYMBOLで翻訳選択子Uの処理時に設定される。

(iii) 行末・入力末の検出。行末を検出したならば、出力手続きを呼ぶ。標準Pascalの仕様では、eolとeofは文字としては入力されない。eolの代りに空白が渡される。行末を確認するための標準論理型関数(eoln)がある。eofがあるべき位置で入力を試みた場合の結果は、保証されない。入力末の確認も標準関数で行なう。

eolの検査は、文字を入力する前に行なっているが、結果はいったん変数に保存され、次回NEXTCHが呼ばれた時点で再検査される。真ならば、出力手続きを起動し、その後eofを検査して、真ならば翻訳を打ち切る。

行末を検出しても出力手続きの起動を一文字分遅らせるのは、記号の直後に行末がある場合を想定しているからである。行末相当の空白をINSYMBOLに渡して、いったんその記号を区切らせ、その行の構文解析を済ませる。その後で出力手続きを呼ぶようにしなければ、誤りが発見されても、警告が次の行に対して出てしまう。また、リスティングが抑制されている時ならば、誤りのあった行の内容が表示されない可能性もある。

この他、文字列が一行中に納まっていることを確認するために、INSYMBOL中の文字列読み込み部でも、上記変数を利用している。

②INSYMBOL

字句解析部の副次的作業(リスティングや警告)は全てNEXTCHで行なっているので、INSYMBOLでは記号の切り出しに専念できる。その骨格を図2に示す。

先ず空白を読み飛ばし、次いで順次四つの場合に分ける。CHTYPEは、英字ならばLETTER、数字ならばDIGIT、その他の文字ならばSPCHAR、という三通

```
1: while CH=' ' do NEXTCH;
   if CHTYPE[CH]=LETTER then [(i) 名前、区切り語]
else if ('0'≤CH) and (CH≤'9') then [(ii) 数]
else if (ord(CH)≤73) or (190≤ord(CH)) then [(iii) 不当な文字]
else case ord(CH) of [(iv) 区切り記号、文字列、注釈] ;
```

図2 旧版のINSYMBOLの骨格

りに文字を分類するための配列である。

(i)は名前または区切り語の読み込み部である。英字で始まる記号は、つづり読み込み用配列(長さは八文字分)に移される。予約語表は、語の長さにより区分けされており、読み込んだつづりの長さ(八文字以上ならば八)により範囲を定め、検索される。

(ii)は数の読み込み部である。

(iii)の場合は、Pascalでは定義されていない文字である。(文字の内部コードはEBCDICである。)

(iv)では、特殊文字をcase文で分岐した後、個々別々に扱う。文字列の処理もこの一部である。また、"("に続いて"*"が来れば注釈であるから、"*"と")"とが連続して現れるまで読み捨てて、最初(名札1)に戻り、次の記号を読み込む。注釈頭部の翻訳選択子の処理も行なう。U選択子が指定されたならば、論理的な行長を再設定する。

2) 測定

字句解析部が隘路になっていることが多い[7,8]ので、INSYMBOL、NEXTCH、一字読みの各々(中で使われる手続きは含む)の呼び出しをループさせて、時間を測定した(表1)。測定用入力データは、Pascalコンパイラの原始プログラムである。

全処理時間	13.9秒
INSYMBOL	6.9秒
NEXTCH	5.8秒
一文字読み	3.0秒

表1 旧版の測定結果

予想通り、字句解析部が全処理時間の半分を占めることが確認された。しかも、記号の切り出しではなく、単なる文字の取り込みに時間がかかっていることが判明した(注1)。

手続き呼び出しに要する時間を測定したところ、引数無しの場合でも、一回当たり6.0μ秒かかることがわかった(注2)。NEXTCHは、(行末としての空白を含む)文字の数だけ、つまり約22万回呼ばれるので、呼び出し時間だけで約1.3秒かかっているわけである(§3表2参照)。NEXTCHを展開すれば、この時間は節約できる。ただし、INSYMBOL中では、NEXTCHが41ヶ所参照されているので、そのどれを展開するかを決めるためには調査が必要である。

原始プログラムは、「行」を単位として扱うのが自然である。ところが、NEXTCHは文字単位で入力して行を組み立てている。行単位で入力できれば、速くできそうである。

次の第三部では、処理対象のコンパイラ原始プログラムの調査の後、改善案を説明し、測定結果を旧版と比較する。

(注1) NEXTCHは、24行しかない。

(注2) FORTRANでは、10μ秒である。

§3 高速化のための改善案

字句解析部の高速化を、二つの方向から考える。一つは、入力データである原始プログラムの特性であり、もう一つは、処理の方法である。

1) 処理の対象について

ねらいをつけるために、Pascalコンパイラの原始プログラム中の文字(表2)と記号(表3)の出現頻度を調べた。

ファイルは可変長形式であり、行の後部の空白は除かれている。それでも空白の割合が四割弱になるのは、段付けのためである。行頭の空白は、合計62,192個あり、これを取り除くと、処理時間は約1.6秒短縮された。

文字列は無視し得る程度しかないので、半分弱を占める英字は、名前または区切り語(以後、名前風語と総称する)に使われていることになる。

以上のことから、字句解析部では、空白の読み飛ばしと、名前風語の読み込みを高速化すべきであるといえる。

文字数	211,592
空白	78,525 (37.1%)
英字	102,932 (48.6%)
数字	4,260 (2.0%)
特殊文字	25,875 (12.2%)
行数	6,283

表2 コンパイラ原始プログラム中の文字の出現頻度

名前	13,870 (33.6%)
区切り語	5,900 (14.3%)
数	1,835 (4.4%)
区切り記号	19,136 (46.4%)
文字列	537 (1.3%)
計	41,278

表3 コンパイラ原始プログラム中の記号の出現頻度

2) 処理方法について

①空白の読み飛ばしについて

文字列の中以外では、空白は、分離する機能を果した後、無視してよいわけである。位置に無関係な方策として、次のようなことが考えられる。

(i)空白を読み飛ばす標準手続きを設ける。

(ii)NEXTCHを二種類持つ。一方は従来通りだが、他方は、先ず空白の読み飛ばしを行なう。

また、行頭、行中、行後部に分けて考えるならば、次のようなことが考えられる。

(iii)行頭:ファイル自体に段付け情報を持たせる。次の行の先頭まで読み飛ばす標準手続き(readln)を拡張して、次の行の空白でない最初の文字まで読み飛ばすような標準手続きを新たに設ける。等。

(iv)行中:行中に現れる空白は少ないので、特に対策を考える必要はない。

(v)後部:可変長形式のファイルを標準的に用いて、後部の空白は除く。

可能な限り早い段階で空白を無視する方が効果は大きい。

最初に発表された段階[1]では、eolという特別な文字があったが、標準Pascalでは、行末は文字としては空白、ということになっている。これでは、もっともよく現れる空白の読み飛ばしの最中に、関数eolnを呼び出す必要があって処理が遅くなる。

②名前風語の読み込みと予約語の分離

(i) PASCAL 8000では、名前風語は八文字まで有効である。読み込み中に、一文字毎に長さの検査をするのは無駄であるので、一行の最大長のバッファに読み込んで、記号が切れた時点で有効部分だけ使用するようにする。切れ目はもちろんCHTYPEを利用して検出する。

(ii) 旧版のように、予約語の表をつづりの長さで区分けすると、二文字または三文字の予約語の種類が多いため、検索回数が多くなる。先頭の一文字で区分けすれば、大きな山はなくなり、検索回数を減らすことができる。さらに、つづりの先頭文字によっては、検索を省略することができる。

ハッシュ法を用いることも考えるべきである。予約語の各々に異なる値を与えるハッシュ関数があったとする。その値域をインデクスとする配列（ハッシュ表）を用意して、各々の予約語から算出される要素には予約語（文字列）へのポインタを、他の要素には無効ポインタを入れておく。読み込まれたつづりからハッシュ表を索き、無効ポインタが得られたら名前であるとし、それ以外ならば文字列の比較をする。こうすることによって、時間がかかる文字列の比較の回数を削減できる。

③入力方法について

旧版は、単なる入力が隘路になっていた（表1）。NEXTCHが遅いのも、まとめれば一行に一度で済ませられることを各文字毎に行なっていたためである。標準Pascalにはないが、手元の処理系には、一行を配列にまとめて読み込む機能があるので、試してみたところ、同じデータを0.6秒で読み終えることがわかった。

一行読みの機能としては、以下の三点を満たせばよい。

(i) バッファ領域外を壊さない。

(ii) 行の長さがわかる。読む前でも後でもよい。

(iii) eofを検出できる。読む前でも、試みて失敗でも、行の長さが負という形でも、方法は問わない。

こういう入力があれば、一文字毎にしていた処理のうち、出力バッファへの移動やバッファのあふれの予防などは不要になる。

さらに、行末を意味する文字を、入力の最後の文字の次に「番兵」として立てて置くことにより、（空白の読み飛ばしを含めて）特に行末の検査をしなくても、自然に切り出しが止まるようにすることができる。U選択子の処理は、この番兵を置く位置の変更で対応できる。

④INSYMBOLのオートマトン化

記号の切り出しが有限オートマトンで実現できることは知られている[9]。ところが、旧版は、case文を使えば済むところでif-then-elseを重ねて使用しており、忠実にオートマトンを実現しているとは言い難い。記号の先頭文字による大分類をすべきである。case文を使えば、検査の順番を気にする必要もない。負担増無く、場合分けを増やすこともできる。

⑤インライン展開

呼び出しの手間を削減するため、NEXTCHをインライン展開する。空白と名前風語の処理で全呼び出し回数の86%（表2）を占めるので、少なくともこの二ヶ所は展開すべきである。

3）新版の構成

新版の字句解析部は、二つの手続きINSYMBOLとGETLNとからなる。従来はNEXTCHが一字ずつ入力していたのに対して、GETLNが一行ずつ取り込む点が大きな違いである。

①入力バッファの構造

外部から一行ずつ、入出力兼用バッファに読み込む。eof、行の長さは読み込む前に調べられる。これにより、eolの検査，出力バッファへの埋め込み，行長の検査は一行に一度で済む。

行の最後の文字の次に，行末を示す文字eolを置く。ただし、U選子が指定されていて、かつ行長が73以上の場合には、73文字目にeolを埋める（図3）。

このeolとしては、Pascalで使用しない文字を選び、名前風語、数、区切り記号に対しては、分離子として働くようにしておく。次回、INSYMBOLが呼び出された時に、eolが記号の開始位置で検出される。この時点では、この行の構文解析は終っているので、出力手続きを呼び、次の行を読めばよい。

以上により，次の文字を取り込むためには、単にバッファのインデクスを進めるだけでよい。

ただし、注釈と文字列の処理だけは、本来任意の文字を書ける場所であるから、行末は、文字eolによってではなく、位置によって確認しなければならない。

②GETLN

以下の通り順に実行される。

(i)出力手続きの起動（前の行の後仕末）。
(ii)eofの検査。真ならば翻訳を打ち切る。
(iii)新しい行の長さを調べ、長過ぎれば翻訳を打ち切る。
(iv)入出力兼用バッファに読み込む。
(v)行の長さとU選択子の指定の有無により位置を決定し、eolを埋める（図3）。リスティングには出力すべき文字かも知れないので、埋め込む位置の文字は保存しておく。
(vi)バッファのインデクスを先頭に合わせる。

ここで行頭の空白を読み飛ばしてもいいのであるが、新版ではINSYMBOLで行なっても速度は変らないため、そちらに任せる（図4参照）。

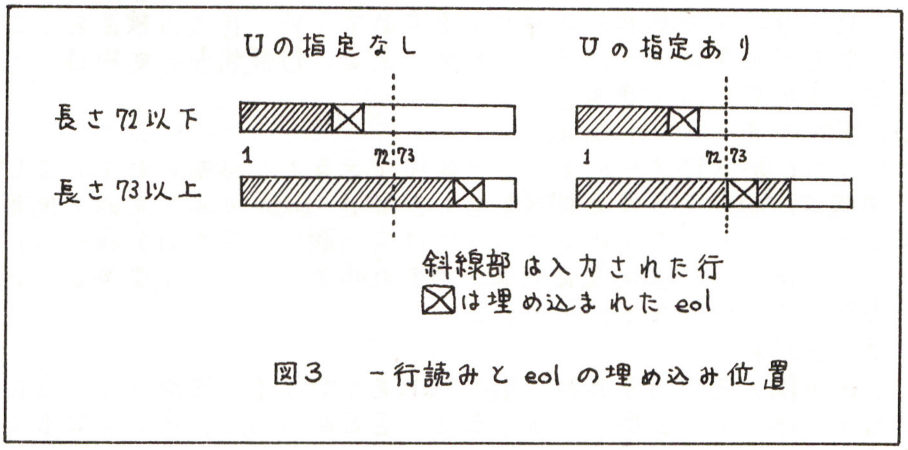

図3　一行読みとeolの埋め込み位置

③INSYMBOL

機能的には旧版と変らない。記号の切り出しと翻訳選択子の値設定を行なう。

文字は、個々別々としてではなく、類別で使用されることが多い。類別は、文字に対して類を与える配列を用いるのが速い[9]。我々は、次の八種類に文字を分類する。

(i) DIGIT ---------- 数
(ii) MUSTBEID ---- H、J、K、Q、X、Y、Z
(iii) LETTER ------ (i)以外の英字
(iv) SINGLE ------- +、−、*、/、=、,、;、@、)
(v) DOUBLE -------- <、>、.、:、(
(vi) QUOTE -------- '
(vii) EOL ---------- eol (内部コード∅)
(viii) INVALID ----- (i)〜(vii)以外の全ての文字

INSYMBOLの骨格を図4に示す。

```
1: while LINE[CP]=␣ do CP:=CP+1;
   case CHTYPE[LINE[CP]] of
        処　理 ;
```

図4　新版のINSYMBOLの骨格

空白の読み飛ばしの後、字類により多分岐する。

(i) DIGIT：数の読み込みを行なう。

(ii) MUSTBEID：これらの文字で始まる予約語はない。従って名前である。

(iii) LETTER：予約語であるかも知れない名前風語である。予約語表は、つづりの頭文字により区分けされている。索表にハッシュ法(先頭の四文字の内部コードを或る整数で割った余りを使う。)も試みたが、頭文字による区分けと大差はなかった。

(iv) SINGLE：一文字だけからなる区切り記号である。

(v) DOUBLE：次の文字を調べる必要がある特殊文字である。一文字目による記号の種別を表から求めた後、分岐し、二字目を調べる。注釈は、この中で扱われる。注釈の読み飛ばし中は、位置によりeolを確認する。eolならば、GETLNを呼ぶ。この行の最後の記号に対する構文検査は、既に終っているからである。

(vi) QUOTE：文字列の読み込み部である。eolは位置で確認する。検出しても、警告は出すが、GETLNを呼ばない。構文検査は終っていないからである。

(vii) EOL：内部コードがEBCDICであるため、通常使用されない∅を用いている。記号の先頭でeolが検出されたら、GETLNを呼ぶ。前回の呼び出し時にeolが分離子として働き、記号は切れ、その記号の処理は終っているからである。

字類は、記号の分類の他に、数と名前風語の終りの検出にも用いられる。例えば、数の読み込みは、図5のようになる。名前風語の読み込みでは、三種の字類との比較を図6のように行なっている。字類にDIGIT<MUSTBEID<LETTER<…のように順序をつけてあるからである。

```
repeat 処　理 ; CP:=CP+1
until CHTYPE[LINE[CP]]≠DIGIT
```

図5　数の読み込み

```
repeat 処　理 ; CP:=CP+1
until CHTYPE[LINE[CP]]>LETTER
```

図6　名前風語の読み込み

4）測定と比較

新版を、旧版と同様の方法で測定した（表3）。

字句解析の部分（INSYMBOL およびそれが呼び出す手続き）で、4.9秒削減したことになる。削減された処理時間の内訳は、次のように考えられる。

(i) 入力単位が文字から行になったことによる
2.4秒（＝3.0秒－0.6秒）

(ii) NEXTCHの呼び出しの22万回が、GETLNの呼び出し六千回に変ったことによる　1.3秒（＝0.6μ秒×（22万－六千））

(iii) NEXTCHで毎字行なっていた各種検査が、各行一度に減ったことによる
0.6秒　（推定）

(iv) 予約語の検索法の改善による　0.3秒　　（計4.6秒）

INSYMBOL	2.0秒
GETLN	0.8秒
一行読み	0.6秒

表3　新版の測定結果

INSYMBOL 自体が、1.1秒から1.2秒へと遅くなったようにみえるのは、旧版ではNEXTCHに任せていた文字の取り込みが、新版ではバッファのインデクスを進めるという形でINSYMBOL中に移動したためである。また、入力文字の参照が、単純変数に対してではなく、配列の要素に対して行なわれるようになった点も、INSYMBOLの負担増の一つである。各々小さいことではあるが、文字の数だけ起きることでもあり、無視はできない（増加した分は、0.4秒程度と見積られる）。

§4　まとめ

Pascalコンパイラの字句解析部の処理法を、一文字単位の考え方から、一行単位の処理法に変更して、高速化に成功した。一行入力を利用したことと関連して、一文字毎に行なっていた検査は、一行に一回で済ませられるようになった。また、従来多種の処理をしていた一字読み手続きが簡略化されたため、インライン展開が楽にできるようになり、呼び出しの手間をなくすことができた。

これらの結果として、コンパイラの処理時間を35％削減できた。またプログラムそのものも、旧版と比べて、整理された形に仕上がっている。追試が、FACOM 230-45S（東工大・佐渡氏）及びFACOM230-38（東大・近山氏）で行なわれ、45％～50％の削減という結果を得ている。ただし、これらは固定長形式のファイルを使用しているため、空白の割合が大きい。

字句解析部を遅くしていた原因を見直してみると、第一に、一行読みは当然備わっているべき機能であったのに、一字読みしかなく、しかも遅かったことがある。実行時ルーチンは一行単位で読んでいるので，現在標準手続きreadが毎字実行時ルーチンを呼び出しているのを、行末に達しないうちは直接一文字取り込むようにすれば、高速化は可能である。

第二に、行末の仕様がある。かつてのPascal[1]のように、行末は検出しやすい文字として存在させるべきである。

第三に、呼び出しの手間の事がある。6μ秒という時間は、FORTRANでの10μ秒に比べれば短かいが，構造的プログラミングの立場から見ると、長過ぎる。ソフトウェア的改善とともに、ハードウェアの改善を強く望みたい。

さて、上記の改善が全て実現されたとしても、旧版は新版より速くはならない。字句解析部の機能分担の仕方が悪いことが、根本原因だからである。処理内容に適した単位で、つまり文字ではなく行で、大きく受け渡すべきであった。構文解

析部に対して字句解析部が存在する理由と同じである。

アセンブラ言語を使って字句解析部を作っていれば、レコード入力とtranslateのような専用命令を自然に使うので、今回の話はなかったであろう。しかし、高級言語で書いても、これだけの性能は得られるのである。旧版（またはその原典）の作成者が定石に通じていれば、あるいは調整用の道具（FORTUNE[6]の類）を用いていれば、欠陥は容易に発見され、既に取り除かれていたと思われる。

一方、今回の作業で考えさせられたのは、ファイル中に占める空白の割合である[10]。空白を圧縮したファイルの仕様を決めて、どこでも使えるようにすることをメーカに期待する。

最後に。移植を考えて作ったソフトウェアは、機械独立性を強調するあまり、ただ移植しただけでは能率が悪くなっているのではないだろうか。今回の場合で言えば、一部の責任はTrunkコンパイラにあるが、他方、字句解析部は外部とのインタフェースを受け持つ基本的部分であるから、移植者も調整は不可欠と考えるべきである。これからは、もともと環境の変化に強いソフトウェアを作っていく必要があると考えている。

謝辞

私共が日々お世話になっている処理系の作成者である、疋田輝雄、石畑清、Gordon Cox、Jeffrey Tobiasの諸氏に感謝する。

参考文献

[1] N. Wirth : 'The Programming Language Pascal', Acta Informatica, Vol.1, No.1 (1971).

[2] K. Jensen, N. Wirth : 'PASCAL User Manual and Report', Lecture Notes in Computer Science, Vol.18, Springer-Verlag (1975).

[3] G. Cox, J. Tobias : 'PASCAL 8000 Reference Manual', Australian Atomic Energy Commission (1978).

[4] T. Hikita, K. Ishihata : 'PASCAL 8000 Reference Manual', University of Tokyo (1976).

[5] 疋田輝雄：「コンパイラのキットを用いたPASCALの移植」, 日経エレクトロニクス, No.149 (1976).

[6] D. Ingalls : 'The Execution Time Profile as a Programming Tool', Design and Optimization of Compilers, Prentice-Hall (1972).

[7] D. Comer : 'MOUSE4 : An Improved Implementation of RATFOR Preprocessor', Software—Practice and Experience, Vol.8, No.1, John Wiley & Sons (1978).

[8] A. C. Hartmann : 'A Concurrent Pascal Compiler for Minicomputers', Lecture Notes in Computer Science, Vol.50, Springer-Verlag (1977).

[9] D. Gries : 'Compiler Construction for Digital Computers', John Wiley & Sons, (1971).

[10] 木村泉、飯島淳子、辻尚史：「ファイルの内容に関する実例研究」, 第17回プログラミング・シンポジウム報告集, 情報処理学会 (1976).

付録 Pascalの金物表現

字句解析部への入力（原始プログラム）の形式をＡＮ記法により以下に示す。

＜原始プログラム＞ ＝＝ ＜拡張記号＞… eol eof

＜拡張記号＞ ＝＝ ＜記号＞ | ＜分離子＞

＜記号＞ ＝＝ ＜名前＞ | ＜区切り語＞ | ＜数＞ | ＜区切り記号＞ | ＜文字列＞

＜分離子＞ ＝＝ ＜空白＞… | ＜注釈＞ | eol

＜名前＞ ＝＝ ＜英字＞ [＜英字＞ | ＜数字＞]…

＜区切り語＞ ＝＝ AND | ARRAY | BEGIN | ⋯ （以下省略）

＜数＞ ＝＝ ＜数字＞… [. ＜数字＞…] [E [+ | -] ＜数字＞…]

＜区切り記号＞ ＝＝ + | - | * | / | := | . | , | ; | : | = | <> |
< | <= | >= | > | (|) | (. | .) | @ | ..

＜文字列＞ ＝＝ ' ＜文字列本体＞ '

＜注釈＞ ＝＝ (* [＜翻訳選択子列＞] ＜注釈本体＞ *)

＜翻訳選択子列＞ ＝＝ $ (＜英字＞ (+ | -)) { , }…

- 文字列本体は、eol，eof 以外の文字の列であり、引用符(')は、もし含まれるならば、連続して偶数個ずつ現われる。
- 注釈本体は、eof 以外の文字の列であり、アスタリスク(*)と右カッコが連続して現われることはない。

本 PDF ファイルは 1979 年発行の「第 20 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