APPLEII
BASICによるテキスト・エディタ

BATE

■T.KY.■

文字列の形をしたプログラムやデータの編集ができる『テキスト・エディタ』をBASICで書きました．テキスト・エディタは，特に成積処理など大量のデータを扱かうときには大変便利ですから，まだ使っていない方にはぜひ使っていただきたいと思います．

はじめに

テキスト・エディタを作った動機の１つは次のとおりです．成積処理などのテープやディスクを使って，比較的大量のデータを扱かうプログラムでも，マイコンに関しては，これまでに発表されたものは，個々の応用プログラムで，キーボードからの入力や，入力されたデータの編集（修正，部分的抹消，挿入など）を行なっているのが普通のようです．

このようなプログラムの編集機能は，一般に低く，入力が正しく行なわれたかどうかを，確かめたり，修正したりするための，充分な機能を持っていません．

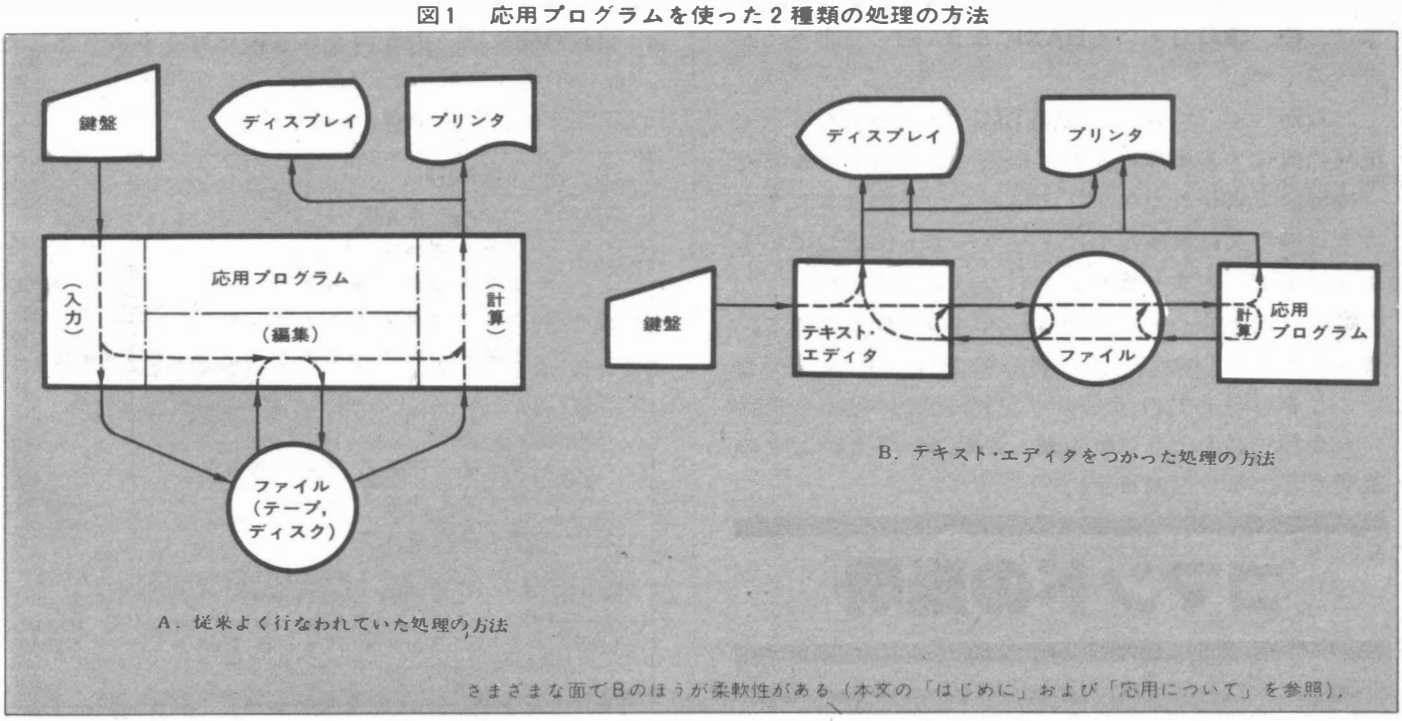
また，このようなプログラムを作るたびに，データ入力に関するプログラム（部分）を書くのは，労多く益少ないと考えられます．

汎用のデータ入力・編集プログラムを作れば，このような問題は解決されます．それが『テキスト・エディタ』です(図１)．

もう１つの動機は，APPLEII にエディタが付属していなかったことです．ただし，いまはお金を出せば買うことができます．

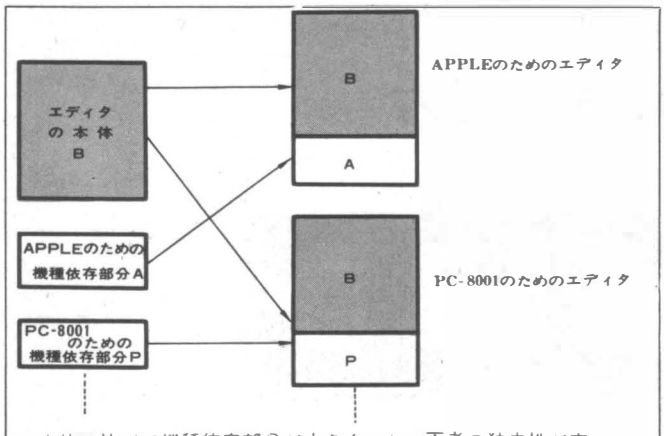
そういうわけでエディタを作ることにしましたが，作るにあたって次のようなことを目標にしました．

図１　応用プログラムを使った２種類の処理の方法



さまざまな面でBのほうが柔軟性がある（本文の「はじめに」および「応用について」を参照）．

図2　異なる機種で同じエディタをつかう方法



本体に比べて機種依存部分が小さく，かつ両者の独立性が高いほど，容易に多種多様な計算機に移植することができる．

❶1,000行程度のテキスト（プログラム，データ）を作ってテープやディスクにしまったり，テープやディスクにしまったテキストを編集し，再びしまったりすることができること．
❷容易につかえること．
❸エディタ自体が容易に作れて，その修正も容易であること．
❶エディタの1部を書き替えるだけで，いろいろな機種（もちろんミニコンや大型機も含みます）で使えること(図2)．
❺『安全性』に気を配ること．

APPLE II，PETなどのマイコンでは，機械語のプログラムよりBASIC のプログラムの方が楽に，気易く扱うことができます．

したがって，❷はエディタをBASIC で書くことを支持していると考えられます．

また，BASIC以外の高級言語が普及していない現在では，❸，❹のためにも**BASICな**BASICで書くのが良いのです．

したがって，エディタはBASICで，しかもなるべく機種に依存する機能は使わずに書くよう心がけました（機械語で書かれたエディタはいくつか発表されています（参考文献参照）し，メーカーでも供給しているようですが，あまり普及していないようです)．

❺の『安全性』というのは，入力の操作を誤って，せっかく入力したデータを壊してしまう危険が少ないことをいいます．

安全性にどのように気を配ったかは，各コマンドの説明のところで触れます．

コマンドの説明

次に，このエディタ（以下BATE——BAsic Text Editorと呼びます）の機能について説明します．

あとの使用例と見くらべるとわかり易いと思いますが，理解するための最善の方法は使ってみることです．

BATEはメモリの中に**テキスト**（＝行がいくつか並んだもの）をたくわえ，そのうちの1行を**ポインタ**が指しています(図3)．

BATEの実行を開始した直後はテキストは空ですが，仮の行が1行あることにして，ポインタはそこを指しています．

エディタは一般に**コマンド**を幾つか持っていて，エディタの実行を終了させるコマンドを除けば，それらを任意の順序で使うことができます．

BATEのコマンドは，

❶ポインタの指す位置を変えるもの．
❷ポインタの指す行に対して作用するもの．
❸ポインタの指す行の次の行から作用するもの．
❹その他．

の4つに分けられます。

以下では，コマンドの綴りと引数，その意味をコマンドごとに述べます．

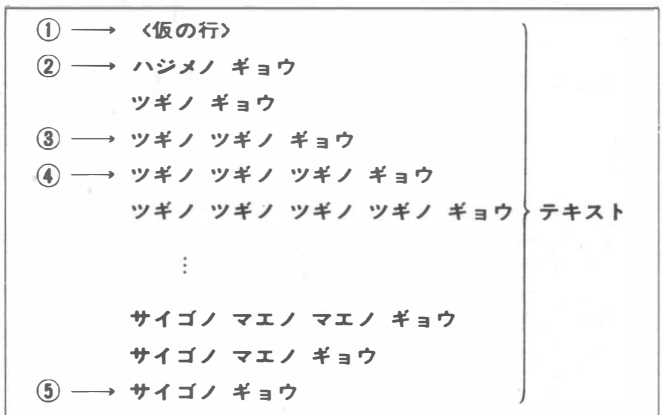
コマンドの綴りは，省略できる部分を小文字で書きました．たとえばChangeAllは**CHANGEALL**または**CA**とタイプします．

以後，記号↵によって RETURN または CR を表わすことにします。

各コマンドは，コマンド名をタイプしたあとに↵を入力し，次に引数のあるものは引数に続けて↵を入力します（普通のエディタではコマンド名と引数を同じ行にタイプしますが，BATEでは，引数の型や数の異なるコマンドを同じINPUT文で入力できるように，↵をはさむことにしました)．

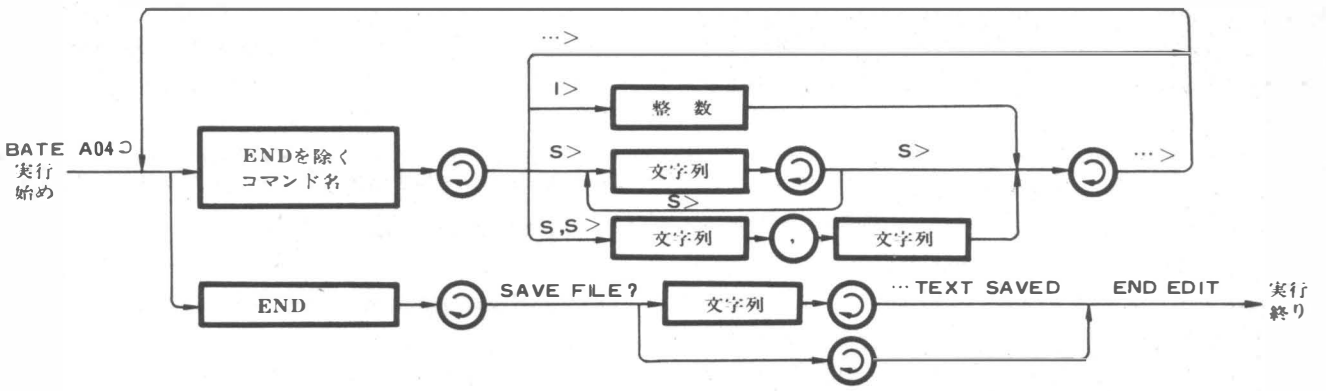
引数の欄には，引数のないものには0を書きました．

図3　ポインタ



ポインタは①～⑤のようにテキストの任意の行を指すことができる．(ポインタは1本である)．①はBATEの実行開始時の位置．ポインタが③の位置にあるときに**Bottom**を実行すると⑤，**Down 1**を実行すると④，**Top**を実行すると①，**Up 2**を実行すると②にポインタは移る．

図4　DATEの構文



矢線の上の文字列はBATEが出す．四角および丸のなかは人間がタイプしてやる．

また，整数の引数をとるものは i，文字列の引数をとるものは s，コンマで区切られた文字列の引数を2つとるものは s_1，s_2 のように書きました．

文字列はコンマなどを含まなければ引用符を付ける必要はありません．

なお，ＢＡＴＥはコマンドの入力を要求するときは〝>〟，整数（Integer）の入力を要求するときは〝I>〟，文字列（String）の入力を要求するときは〝S>〟で催促します(図4)．

(1) ポインタの位置を変えるコマンド

いずれも図3を参照されたい．

Bottom 0:

テキストの最後の行をポインタで指すようにします。

Down i:

ポインタを現在より i 行下にずらします($i>0$)．

もし，はじめポインタが指していた行の下に i 行以下しかテキストがなかったときは，〝END␣OF␣FILE〟と表示して，効果はBottomと同じです．

Top 0:

テキストの最初の行より1つ前に仮の行があることにして，そこをポインタで指すようにします．この仮の行は，テキストを入力する前からある仮の行に対応します．

Up i:

ポインタを現在より i 行上にずらします（$i>0$)．初めにポインタが指していた行の上に $i-1$ 行以下しかテキストがなかったときは，〝BEGINNING␣OF␣FILE〟と表示して，効果はTopと同じです．

〔注意〕　これらのコマンドの実行にはポインタが移動する行数に比例する時間がかかります．APPLE IIの実数BASICの場合，100行/秒くらいの速度です．また，APPLE IIの場合，DownおよびUpは引数の入力をもとめている時点で引数を入れずにリターンをかえすと，そのコマンドの実行を中止することができます．

(2) ポインタの指す行に対して作用するコマンド

以下のコマンドはポインタの位置を変えません．

Change s_1, s_2:

ポインタの指す行に最初に表われる文字列 s_1 を文字列 s_2 で置き替えます．s_1 が空列のときは行の始めに s_2 が挿入されます（使用例1．参照)．

わずかな誤りのある行を修正するときに使います．

Replace s:

ポインタの指す行の内容を s で置き替えます．誤りの多い行を修正するときに使います．また，Insertによっては空行（文字を含まない行）を入力することはできませんが，Replaceで空行を作ることができます(使用例5．参照)．

(3) ポインタの指す行の次から作用するコマンド

Changeall s_1, s_2:

ポインタの指す行の次からテキストの終りまでに表われるすべての文字列 s_1 を文字列 s_2 で置き替えます．ポインタの位置は変りません（Verifyコマンドの説明を参照)．

〔注意〕　処理するテキストの行数が多かった場合，Changeallは最も実行時間のかかるコマンドです．途中で短気をおこさないように．

DELETE i:

ポインタの指す行の次の行から連続 i 行を抹消します．i が6以上のときは，BATEは i の値が正しいかどうかを尋ね，■またはYESとこたえたときだけ抹消を行ないます（これは，誤って大きな数を入力したり，鍵盤のチャタリングで大きな数が入ってしまったときに，必要なデータが壊されてしまうのを防ぐためです．〝DELETE〟とfull spellでなければ

ならないのも同様の理由からです). ポインタの位置は変りません.

Find *s*:

ポインタの指す行の次の行から最後までの間で文字列*s*で始まる行を逐次さがし, それがあればそこにポインタを移します.

なければ"STRING NOT FOUND"と表示し, ポインタの位置は変りません.

*s*として空列を入力する(すなわち?だけを入力する)と, もっとも最近Findコマンドで指定された文字列をさがします.

したがって, ポインタの指す行の次から*n*番目にある文字列が現われる場所をさがすということを, 比較的簡単に行なえます.

BASICのプログラムを扱う場合(APPLE IIではBASICのプログラムのファイル中間言語で書かれているので, テキスト・エディタで直接それを扱うことはできませんが), 行の始めの部分, すなわち行番号で行をさがせると便利です.

このコマンドを使えばそれを行なうことができます(使用例3.参照).

Insert *s**:

ポインタの指す行の次に, 新たな行を?で区切って任意の数だけ挿入するためのコマンドです.

挿入をやめたいときは空行を入力します(すなわち?を2つ続けて入力します). 最後の空行は挿入されません.

コンマなどの特殊文字を入力することはできません(プログラムの説明の行番号1630のところを参照). このコマンドの実行後, ポインタは最後に挿入された行を指します(使用例4.参照).

〔注意〕 Insertの引数*s**は, 上に説明したような任意個の文字列のならびを表します.

(4) その他のBATEコマンド

以下のコマンドはポインタの位置を変えません(ただし, ENDについてはポインタの位置は問題になりませんが).

END 0:

BATEの実行を終了したいときに使います. このコマンドを入力すると, BATEは?は"SAVE FILE ?"と聞いてきます. そこで?をすぐ入力するとただちにBATEの実行は終りますが, 空でない文字列に続けて?を入力すると, その文字列を名前とするファイルにテキスト全体が出力されたうえで実行を終了します(full spellにしたのは,"END"を入力した後はいずれにしてもBATEの実行を続けることができないため, 誤ってこのコマンドを入力することは危険だからです.

また,"SAVE FILE ?"ときくのはファイルをSaveするのを忘れていないかどうかを確認するためです).

List *i*:

現在ポインタが指している行から始まる*i*行をリストします. *i*>20のときには20行ごとにリストを中断しますので?を入れてください.

ポインタが指している行以降に*i*行未満しかテキストが存在しないときは, 最後までリストしたあと(1行あけて)"END OF FILE"と表示します.

APPLE IIの場合, CONTROL C をタイプすることによってリストを中止することができます(CONTROL を押しながらCを押す).

なお, ポインタがTopを指しているときは*i*−1行しかリストされません.

READ *s*:

文字列*s*を名前とするファイルからテキストを入力します. それまでにメモリにテキストがあっても, それは消されます.

READの実行後, ポインタは入力された最後の行を指します(full spellにしたのは,"READ"がすでにあるテキストを消してしまう『危険な』コマンドだからです).

Save *s*:

文字列*s*を名前とするファイルにテキスト全体を出力します. *s*として空列を入力する(すなわち?だけを入力する)と, もっとも最近READコマンドで入力したファイルに出力されます(したがって, そのファイルのもとの内容は壊されます. すでにある名前のファイルを指定したときも同じです). 編集中に事故が起こってテキストが失なわれてしまうのを避けるため, ディスクのあるシステムではときどきSaveをしましょう.

Verify ON/OFF:

BATEには"verify on"と"verify off"の2つのモードがあって, このコマンドでそれを切り替えます.

verify onモードでは, 各コマンドの実行後にポインタが指している行の内容が表示されます.(ただし, ポインタがtopを指しているときは表示されません).

また, Change allコマンドを実行したとき, 変更のあった行の内容を(変更された回数だけ)表示します.

verify offモードでは, これらは表示されません.

(5) APPLE II BASIC/DOSのコマンド

APPLE IIでBATEを使うと, 上のコマンドのほかにAPPLE IIのコマンドの幾つかをBATEのコマンドと同じようにして使うことができます.

たとえば, CATALOG, DELETE ファイル名, PR# *n* などが使えます(この場合のDELETEと, BATEのDELETEコマンドは無関係です).

PR# *n* を使うことによって, テキストの一部分または全体のリストをプリンタに出力することができます.

また, ディスクからファイルをREADした後にデ

ィスクをとりかえて，CATALOGをとってからSaveすることによって，データ・ファイルを別のディスクに移すこともできます．

しかし，逆にInsertなどでこれらのBASIC/DOSコマンドと同じ文字列を入力するときには，引用符〝を付けなければなりません．

使用例

BATEの使用例をあげます．下線を引いた箇所が鍵盤から入力した文字です．例はいずれもVerify on モードの場合です．

1．空列のChange

```
0  REM  PROGRAM          -- (1)
>CHANGE↵
S, S>, 10↵               -- (2)
100  REM  PROGRAM        -- (3)
```

(1)はポインタが指す行の元の内容，(2)において，s_1＝〝 〃，s_2＝〝10〃，(3)は変更されたポインタが指す行の内容です．

2．テキストをすべて抹消すること

```
>TOP↵
>DELETE↵
1>99999↵
99999  LINES? (Y/N) Y↵
END  OF  FILE
```

3．Findの連続

```
>TOP↵
>LIST↵
1>99↵
 TAKE WA
TABERARENAI
GA
TAKENOKO WA
TABERARERU.

END  OF  FILE
>FIND↵
S>TA↵
TABERARENAI              -- (1)
>F↵
S>↵
TAKENOKO WA              -- (2)
>F↵
S>↵
TABERARERU.
```

Findコマンドで〝TA〃をさがすと，最初の行は空白で始まっているので見付からず，2行目が見付かる(1)．もう1度さがすと4行目が見付かる(2)．次も同様．

4．InsertとChangeの組み合わせ

```
>INSERT↵
S>Dear  Mr.  Apple.↵
S>You  are  very  expensive↵
S>so  I can not bye  you ↵   -- (1)
S>↵                          -- (2)
so  I  can  not  bye  you.
>CHANGE↵
S, S>bye, buy↵
so  I  can  not  buy  you.
```

Insertの直後はポインタは最後に挿入した行を指すので，(1)でミス・タイプをしたことを↵を入力した直後に気付いたときは，空列を入れ(2)，Changeでミスを直します（その後Insertで続きをタイプする）．

5．閉じた使用例1（すでにあるファイル編集）

BASICのレベルからBATEを呼び出し，再びBASICのレベルへ戻るまでの『閉じた』例を示します．まず，すでにあるファイルを編集する場合です．

```
] LOAD BATE↵
] RUN↵
BATE A04
>READ↵
FILE NAME>ASCII↵
          :
          :
70  END
>TOP↵
>LIST↵
1>99↵
10  REM  EBCDIC  CODE
20  FOR  I=0  TO  255
30  IF  I-I/8*8=0  THEN  50
40    PRINT
50    PRINT  I;" ";CHR$(I);"  "
60  NXT  I
70  END
END  OF  FILE
```

```
>DOWN↵
I>1↵
10  REM  EBCDIC  CODE
>REPLACE↵
10  REM  EBCDIC  CODE
->10 REM --ASCII CODE--↵
10  REM  --ASCII  CODE--
>DOWN↵
I>2↵

30      IF I-I/8*8=0 THEN 50
>CHANGE↵
S, S>"I/8*8", "INT(I/8)*8"↵
30      IF I-INT(I/8)*8=0 THEN 50
>FIND↵
S>60↵
60  NXT  I
>CHANGE↵
S, S>X, EX↵
60  NEXT  I
>TOP↵
>LIST↵
I>7↵
10  REM  --ASCII  CODE--
20  FOR  I=0  TO  255
30      IF I-INT(I/8)*8=0 THEN 50
40      PRINT
50      PRINT  I; "␣"; CHR$(I); "␣␣"
60  NEXT  I
70  END
>SAVE↵
FILE␣NAME>↵
        ⋮
        ⋮
TEXT  SAVED
>END↵
SAVE  FILE?↵
END     BATE
```

6．閉じた使用例2（新たなファイルを作ること）

次に，新たなファイルを作る，閉じた例を示します．

```
]LOAD␣BATE↵
]RUN↵
BATE  A04
>INSERT↵
S>マエノ  レイワ↵
S>スデニ  アル  ファイルヲ↵
S>ヘンシュウスル  レイデシタガ↵
S>コンドノワ  アラタナ  ファイルヲ↵
S>ツクル  モノデス↵
S>↵
ツクル  モノデス
>SAVE↵
FILE  NAME>レイノセツメイ↵
        ⋮
        ⋮
TEXT  SAVED
>END↵
SAVE  FILE?↵
END  BATE
```

応用について

始めに書いたように，エディタを使えばプログラムやデータをテープやディスクにしまったり，いったんしまったプログラムやデータを取り出して編集したのち，再びしまったりすることができます．

そこで，成績処理や統計処理などの応用プログラムは，（ディスク・ベースの場合には入力すべきファイルの名前だけを鍵盤から読み込んで），テープやディスクから入力するようにすれば良いわけです．

BATEで作ったファイルには，最後のデータの次の行に〝/*〟が書かれるので，応用プログラムはそれを使ってデータが誤っていないかどうかを検出することができます．

また，応用プログラムは，その出力をテープやディスクに出力し，かつ，データの終りに〝/*〟を書くようにすれば，BATEを使ってその出力の一部を取り出してみたり，それを編集して別のプログラムの入力ファイルとして使ったりすることができます(図1参照)．

成績処理のばあい，たとえば次のようなファイルをエディタで作ります．

```
1-1␣␣1␣RINGO␣ZIRO      --組，番号，名前
0                       --英語の点
100                     --数学の点
50                      --国語の点
1-1␣␣2␣AIGAN␣DOBUTU    --組，番号，名前
60                      --英語の点
50                      --数学の点
70                      --国語の点
 ⋮                       ⋮
 ⋮                       ⋮
```

これをエディタ（のListコマンド）を使って正しく入力されたかどうか確かめたあと，成績処理のプログラムを走らせます．

成績処理のプログラムは上のファイルを入力し，ディスプレイ（またはプリンタ）とファイルに出力します．

処理の途中でエラーが起こったときはあわてずに原原をよく調べ，データに誤まりがあるとわかったときは，再びエディタを使ってファイルを直します．

必要なだけのこの操作を繰り返すことができます．

具体的な応用例については，機会があれば別に述べたいと思っています．

プログラムとその説明

ミニ・フロッピー・ディスク付きのAPPLE IIのためのBATEのプログラム・リストと，簡単な説明をします．

行100で名前と版（A 04）を表示します．行200は例外（＝入力の誤り，プログラムに虫がいたときの配列添字の範囲から外れなど）が起ったとき行50000に飛ぶことを指定しています．

行300の〝〟は空列ではなく〝CONTROL D〟です．

行530のTXT＄はテキストを入れる配列，行580のPTRはそれを1対1に指す『ポインタ』からなる配列（PTR（n）＝iのときPTR（n）はTXT＄（i）を指していると考がえる），行550～555のPL，PUは，PTRのなかを指す『ポインタ』です．

プログラムの実行中，これらの関係は図5のようになっています（前節まで『ポインタ』と呼んできたものは，プログラムの上には存在せず，それが指す行はPLが指す行より1つ前の行だということに注意してください）．

したがって，ポインタを下げる（Down, Bottom）ときには図6のようにPTR(PU)，PTR(PU＋1)，……の内容をPTR(PL)，PTR(PL＋1)，……へコピーしてPL，PUをずらせば良く，ポインタをあげるとき（Up, Top）はその反対のことをすれば良いわけです．

ただ，Down，Upの場合には指定されただけの行数が存在しないことがあるので，多少の注意を要します．

さて，行1023～1027では，verify onモードでポインタが先頭になければポインタの指す行を表示します．行1500～1670はファイルへの出力をおこなう副プログラムですが，これについてはSaveのところで説明します．行2000～4910は行1030で入力したコマンド（名）を解析し，それぞれのコマンド・インタープリタに分岐します，行4850では，コマンドが空列のときはもう一度コマンドを読みに行っています（この行があるためにAPPLE IIではBASIC/DOSのコマンドが〝COMMAND　ERROR！〟といっておこられずに使えるのです）．

行11000～11170ではBottomコマンドの処理をしますが，これについてはすでに簡単に触れたので説明は省略します．

行12000～12810ではChangeコマンドの処理をします．topにいるときChangeをしようとすると，行12120のメッセージが出力されます．

行12140でポインタの値をCPTに入れ，行12150でさ

図5　TXT＄，PTR，PL，PUの関係（例）

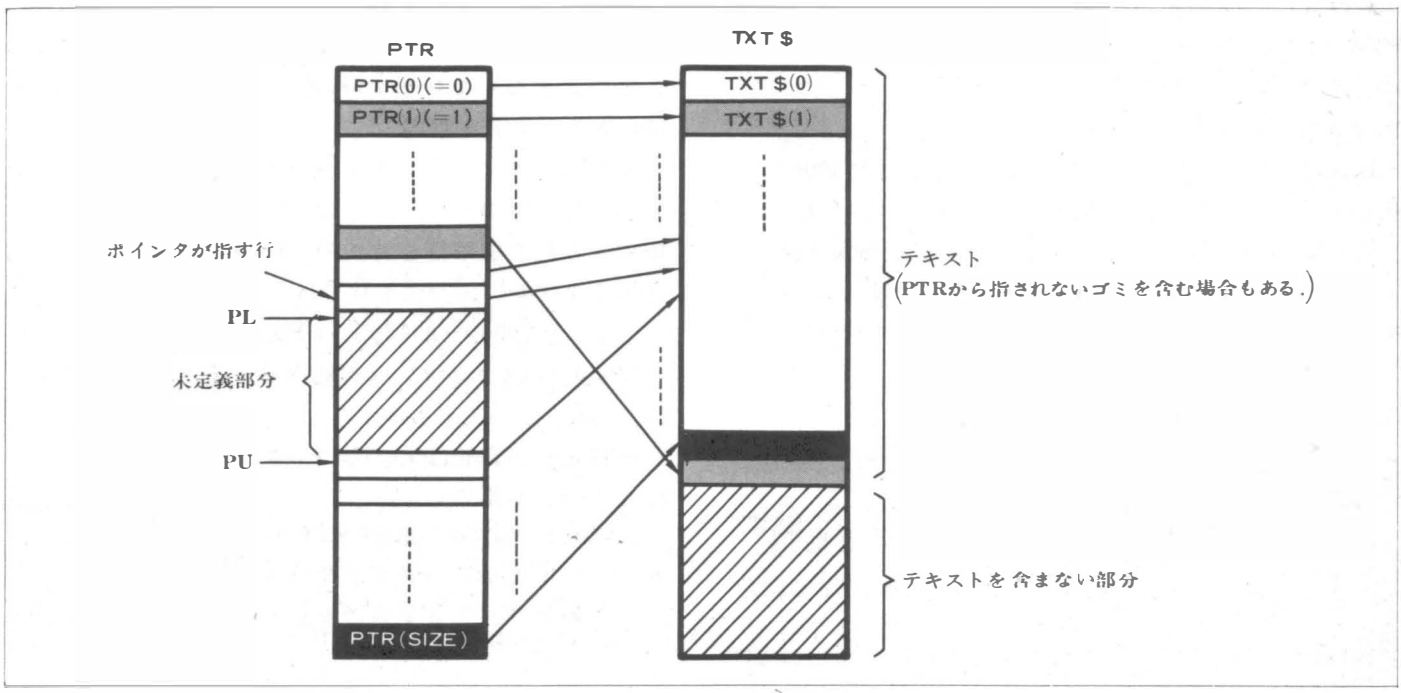
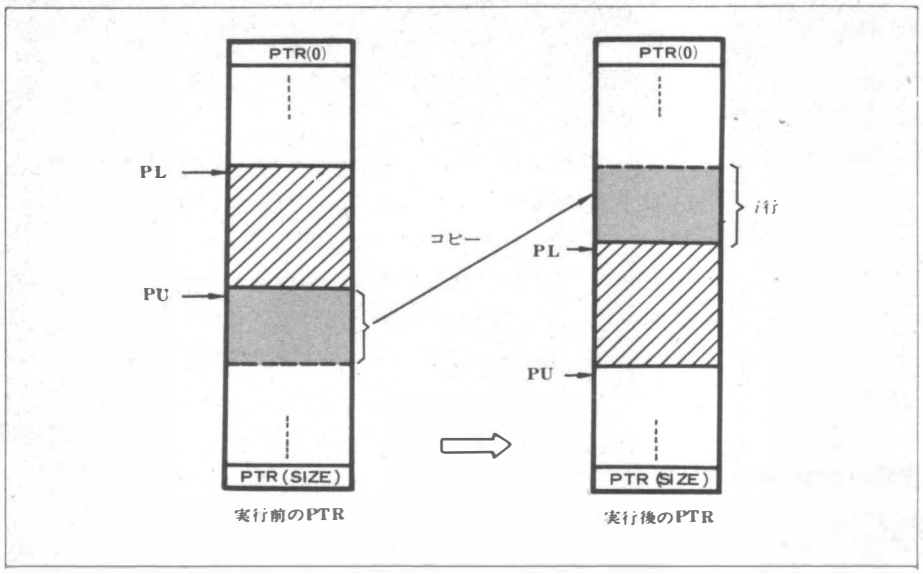


図6　Down i の実行



がすべき文字列の長さをLに入れ，行12160～12180でさがします．

見付かれば行12500へ抜けますが，それ以降の部分で本質的なのは行12800です．行12510～12710はＬＥＦＴ＄（・，０），RIGHT＄（・，０）がとれない(おこられる）のでしかたなく付けたのです．

LEFT＄（・，０），RIGHT＄（・，０）がおとなしく空列を返してくれれば，こんな面倒なことはしなくてすむのですが．

行13000～13530はDownコマンド処理ですが，これも説明は省きます．

行15000～15250はFindコマンドの処理です．行15110～15120でPTR（PU）からPTR（SIZE）までが指すテキストの中を逐次さがし，さがしている文字列が見付かれば行15200～15240でポインタをそこへ移します．

行18000～18900はInsertコマンドの処理をします．InsertはTXT＄のあいた場所に新たなテキストを入れ，PTR（PL），PTR（PL＋１），……でそれを指すようにし，終ったらＰＬを新たに入れた行数のぶんだけズラせば良いわけです．

Insertしたため行数が限界を越えると，行18500でエラー・メッセージを出します．

このようなときにはいったんSaveして善後策を練ってください（いったんSaveしてREADすると，編集中にDELETEした行数のぶんだけ余裕ができます)．

行21000～21510はListの処理をします．ポインタの指す行だけ特別扱いをする必要が生じています．

行27000～27160はReplaceコマンドの処理ですが，特に説明の必要はないでしょう．

行28000～28210はSave コマンドの処理です．実際に仕事をするのは行1500～1670の副プログラムなので，ここでその説明をします．

行1610～1620でファイルを書き込み可能な状態にします（これらはAPPLE独特のコマンドです)．

行1615～1648でテキストをファイルへ出力し，行1650でファイルを閉じます（これもAPPLEだけのコマンド．

ただし，ここでファイル名を書かせているのは人間のため)．

ところで，BATEでコンマなどの特殊文字を含む文字列を扱えない（Insertコマンドの説明を参照）理由は，INPUT文でコンマを含む引用符なしの文字列を入力することができないからですが，行1630と1644を，

```
PRINT CHR$(34); TXT$(PTR(I)); CHR$(34)
```

のようにすると，Saveコマンドで出力したファイルを再びREADしてもコンマなどを正しく入力することができます．

その代わり，それを別のプログラムで数として，またはいくつかの数や文字列が並らんだものとして読むことはできなくなります(機能の向上についてのSave Quoteコマンドを参照)．

なお，上のようにしてもあいかわらず引用符〝は扱えません，処理系によっては，TXT＄（PTR(I)）の中に表われる引用符を２つの引用符で置き替えてやれば扱えますが，APPLE IIではだめです．

さて，行29000～29150ではTopコマンド，行31000～31530ではUpコマンドの処理をしますが，説明は省略します．

行32000～32160はVerifyコマンドの処理ですが，これも説明は不要でしょう．

行36000～36150はDELETEコマンドを処理しますが，これはただＰＵをズラすだけで済みます．

そのかわり，ＴＸＴ＄のなかにゴミができるわけです．

行37000～37530はREADコマンドを処理します．行37110～37120でファイルを読み込み可能な状態にして．行37125～37170で入力します．

ここでも限界を越える行数が入力される可能性があるので，それをチェックしています．正常に入力が終れば行37510でファイルを閉じます．

行38000～38510はChangeallコマンドの処理ですが，これはChange，Findと似ているので説明は省きます．

行39000～39140はＥＮＤコマンドの処理です．テキストの出力が必要なきときはSaveコマンドと同じ副プログラムを呼んでいます．行50000以下は例外が起こったときの処理をする部分です．

移植について

BATEは，文字列変数を持ち，LEFT$，MID$，RIGHT$といった関数を持っていれば，実数BASICの上にでも整数BASICの上にでも移植することができます．(上のプログラムはなりゆき上実数BASICの上に作ったのですが，その結果必要もないところにコストのかかる実数を使うはめになり，整数BASICのほうが良かったと思っています．APPLE IIでは，実数BASICの上で作ったファイルをLOADすると自動的に実数ＢＡSICに入ってしまうので，このようなとき不便です)．

APPLEII以外の機械や，テープ・ベースのAPPLE IIに移植するには，場合によっては次の点を直さなければなりません．

❶標準のBASICでは，名前は１文字の英字，またはそれに１文字の数字が付いたものですが，上のプログラムでは，２文字以上の英字列をつかっていますから，これを１対１に対応する別の名前に置き替えなければなりません．

❷行番号として４桁以下の数しか許されないBASICの上へ移植するときなどは，行番号のつけかえが必要です．

上のプログラムの行番号はほとんど10の倍数になっているので，10で割って同じ番号になったところだけ付け替えるのが簡単でしょう（GOTO文やGOSUB文の行き先も忘れずに直しましょう)．

❸行200および50000～50025は，実行中に例外が発生したときにエディタから抜けてしまうのを避けるために付けた部分です．そのためにいくつかのコマンドの実行を中止することが可能になっていますが，それ以外に特に問題点はないので，除いてもさしつかえありません．

例外処理の機能を持たないBASIC の上に移植するときは，この部分は削ってください．

❹行520で，配列の大きさを決める変数(実は定数)の値を定義していますが，ＲＡＭの少ない場合にはもっと小さくする必要があるでしょう．

❺行1500～1670ではテキストをファイルへ出力していますが，ここは機種に依存するので書き直す必要があります．また，行37000～37530ではテキストをファイルから入力していますが，出力の出合と同様ここも書き直す必要があります．

ファイルの入出力に関係して行300でD$を定義していますが，他の機種へ移すときはこの行は必要ありません．

機能の向上について

BATEのコマンド体系はまだ充分とはいえません．次のようなコマンドを加えることを考えていますが，まだ最終的な決定をしていません．

BATEを拡張して使いたい方のために書いておきます．

Append　s　：

ポインタを指す行の行末に文字列sを付けたします．

Find String　s　：

文字列sを含む行をさがします．Findと違って，sは行の途中にあっても良いわけです．

Find Line　s　：

行全体がsと一致する行をさがします．BATE ではコンマが入力できないので，それが扱うテキストはごく短い行の集まりになりがちで，Findを使ってみた結果それよりはFind Lineのようなコマンドのほうが便利なように思われます．なぜなら

1

という行をさがしたいとき，Findだと，

100

のような行も見付かってしまうからです．

Merge　s　：

sはファイル名．READコマンドに似ていますが，すでにあるテキストを消さず，Insertのようにポインタの指す行の次に挿入します．

Save Quote　s　：

Saveとほぼ同じですが，特殊文字を含む文字列をBATEのREADコマンドで読めるように引用符をつけて出力します．このコマンドを作れば，引用符をのぞく特殊文字を扱うことができるようになります(Insert では，引用符を付ければ引用符以外の文字は入れられます)が，SaveとSave Quoteを使い分けるのはわずらわしく，間違いのもとです．

BATEの欠点とその対策

BATEはBASICで書かれているうえ速さに格別の考慮を払っていないので，Top，Changeallといったコ

マンドの実行はかなり遅いという欠点があります．しかし，Top，Bottomなどに関していえば，速さはミニ・フロッピー・ディスクに比べれば充分速いといえます．

したがって，大量のデータを扱うときにはREAD，Saveにかかる時間のほうがはるかに問題です(APPLE IIでは5～20行/秒くらいの速さです)．

この点はテキストのディスク/テープ上の表現を変えたり，入出力の部分を書き直したりすればある程度改善されるでしょうが，根本的な解決はハード・ディスクなどより高速な媒体の導入によるしかないでしょう．

コンマなどの入力が難かしいのも，BASICのINPUT文を使ったゆえの欠点です．しかし，この点はINPUT文を機種に依存した入力文で書き替えることで解決可能のはずです（たとえばPC-8001ならLINE INPUT文でたちどころに解決します)．

また，参考文献にあげたエディタとは違って，メモリが充分あることを前提としているので，RAMが16KB未満のシステムではこのまま使うのは困難だと思います．必要度の低いコマンド(たとえばChangeall)を削ることで，より小さなシステムでも使えるようになるでしょう．

おわりに

始めに述べた5つの目標は達成することができたと思います．BATEは上に述べたように幾つか欠点を持っていますが，それにもかかわらず，エディタがない場合に比べると，大変便利です．

この便利さは，(そして不便な点も)使ってみなければわかりませんから，まだエディタを使ったことのない方にはもちろん，ほかのエディタを使っている方にも使ってみていただきたいと思います．

◘参考文献

1）和田英一：〝マイクロエディタ〞，bit臨時増刊，マイクロコンピュータのプログラミング，pp.44-53（8080のテキスト・エディタのリストとその解説）

2）小川　尚：〝6800用テキスト・エディタ〞，マイコン，'77年12月号，pp.19-28

3）野村浩郷：〝6800プログラム・テクニック3テキスト・エディタの作成とMT-2フォーマットの設計〞，インターフェース，'78年10月号，pp.113-122

4）野村浩郷：〝6800のプログラム開発システム〞，インターフェース，'79年1月号，pp.25-74

5）秋山貞夫：使いやすさを徹底的に追求した8080用セルフ・アセンブラ，インターフェース'79年1月号，pp.75-123

BATE プログラム リスト

```
]LIST
10  REM ************************
20  REM BATE (BASIC TEXT EDITOR)
22  REM
25  REM     AUTHOR: T.KY.
30  REM
100  PRINT "BATE A04"
120  REM
140  REM ---IMPLEMENTATION DEPEND
     ENT PART
200  ONERR  GOTO 50000
300 D$ = "": REM <CONTROL>D
500  REM ************************
510  REM  --CONSTANT
520 SIZE = 2000
525  REM  --GLOBAL VARIABLES
530  DIM TXT$(SIZE)
540  REM PU,PL:POINTERS TO PTR
550 PL = 0
555 PU = SIZE + 1
560  REM MAX:END-OF-TEXT POINTER
570 MAX = 0
580  DIM PTR(SIZE)
595 VRFY = 1: REM VERIFICATION MO
    DE
1000  REM ************************
1010  REM COMMAND READER
1020  REM
1023  IF VRFY = 0 THEN 1030
1025  IF PL = 0 THEN 1030
1027  PRINT TXT$(PTR(PL - 1))
1030  INPUT ">";C$
1040  GOTO 2000
1500  REM ************************
1510  REM SUBROUTINE SAVE
1520  REM
1600  PRINT D$;"OPEN ";NAME$
1610  PRINT D$;"WRITE ";NAME$
1615  IF PL = 0 THEN 1641
1620  FOR I = 0 TO PL - 1
1630  PRINT TXT$(PTR(I))
1640  NEXT I
1641  IF PU > SIZE THEN 1648
1642  FOR I = PU TO SIZE
1644  PRINT TXT$(PTR(I))
1646  NEXT I
1648  PRINT "/*"
1650  PRINT D$;"CLOSE ";NAME$
1660  PRINT "TEXT SAVED"
1670  RETURN
2000  REM ********************
2010  REM COMMAND ANALYZER
2020  REM
2100  IF C$ = "B" OR C$ = "BOTTOM
     " THEN 11000
2200  IF C$ = "C" OR C$ = "CHANGE
     " THEN 12000
2300  IF C$ = "D" OR C$ = "DOWN" THEN
     13000
2500  IF C$ = "F" OR C$ = "FIND" THEN
     15000
2800  IF C$ = "I" OR C$ = "INSERT
     " THEN 18000
```

```
3100  IF C$ = "L" OR C$ = "LIST" THEN
    21000
3700  IF C$ = "R" OR C$ = "REPLAC
    " THEN 27000
3800  IF C$ = "S" OR C$ = "SAVE" THEN
    28000
3900  IF C$ = "T" OR C$ = "TOP" THEN
    29000
4100  IF C$ = "U" OR C$ = "UP" THEN
    31000
4200  IF C$ = "V" OR C$ = "VERIFY
     THEN 32000
4600  IF C$ = "DELETE" THEN 36000

4700  IF C$ = "READ" THEN 37000
4800  IF C$ = "CA" OR C$ = "CHANG
    EALL" THEN 38000
4840  IF C$ = "END" THEN 39000
4850  IF C$ = "" THEN 1000
4900  PRINT "COMMAND ERROR!"
4910  GOTO 1000
11000  REM **********************
11010  REM BOTTOM COMMAND
11020  REM
11100  IF PU > SIZE THEN 1000
11110 N = SIZE - PU + 1
11120  FOR I = 0 TO N - 1
11130 PTR(PL + I) = PTR(PU + I)
11140  NEXT I
11150 PL = PL + N
11160 PU = PU + N
11170  GOTO 1000
12000  REM ***********************
12010  REM CHANGE COMMAND
12020  REM
12100  INPUT "S,S)";SRC$,DST$
12110  IF PL <  > 0 THEN 12140
12120  PRINT "ILLEGAL OPERATION"
12130  GOTO 1000
12140 CPT = PTR(PL - 1)
12150 L =  LEN (SRC$)
12160  FOR I = 1 TO  LEN (TXT$(CP
    T)) - L + 1
12170  IF  MID$ (TXT$(CPT),I,L) =
    SRC$ THEN 12500
12180  NEXT I
12190  PRINT "STRING NOT FOUND"
12200  GOTO 1000
12500 LL = I - 1
12510 RL =  LEN (TXT$(CPT)) - I -
    L + 1
12520  IF LL > 0 AND RL > 0 THEN
    12800
12530  IF RL > 0 THEN 12700
12540  IF LL > 0 THEN 12600
12550 TXT$(CPT) = DST$
12560  GOTO 1000
12600 TXT$(CPT) =  LEFT$ (TXT$(CP
    T),LL) + DST$
12610  GOTO 1000
12700 TXT$(CPT) = DST$ +  RIGHT$
    (TXT$(CPT),RL)
12710  GOTO 1000
12800 TXT$(CPT) =  LEFT$ (TXT$(CP
    T),LL) + DST$ +  RIGHT$ (TXT
    $(CPT),RL)
12810  GOTO 1000
13000  REM ********************
13010  REM DOWN COMMAND
13020  REM
13050  INPUT "I)";N
13110  FOR I = 0 TO N - 1
13115  IF PU + I > SIZE THEN 1350
    0
13120 PTR(PL + I) = PTR(PU + I)
13140  NEXT I
13150 PL = PL + N
13160 PU = PU + N
13170  GOTO 1000
13500  PRINT "END OF FILE"
13510 PL = PL + I
13520 PU = PU + I
13530  GOTO 1000
15000  REM ************************
    **
15010  REM FIND COMMAND
15020  REM
15100  INPUT "S)";STRG$
15105  IF STRG$ = "" THEN 15110
15107 S$ = STRG$
15110  FOR I = PU TO SIZE
15115  IF S$ =  LEFT$ (TXT$(PTR(I
    )), LEN (S$)) THEN 15200
15120  NEXT I
15140  PRINT "STRING NOT FOUND"
15150  GOTO 1000
15200  FOR N = PU TO I
15210 PTR(PL + N - PU) = PTR(N)
15220  NEXT N
15230 PL = PL + I - PU + 1
15240 PU = I + 1
15250  GOTO 1000
18000  REM ************************
    *
18010  REM INSERT COMMAND
18020  REM
18030  FOR PL = PL TO PU - 1
18040  IF MAX < SIZE THEN 18100
18050  PRINT "STORAGE OVERFLOW"
18060  PRINT "PLEASE SAVE"
18070  GOTO 1000
18100  INPUT "S)";STRG$
18110  IF STRG$ = "" THEN 1000
18120 MAX = MAX + 1
18130 PTR(PL) = MAX
18150 TXT$(MAX) = STRG$
18160  NEXT PL
18170 PL = PU
18900  GOTO 1000
21000  REM ************************
    **
21010  REM LIST COMMAND
21020  REM
21105  INPUT "I)";N
21109  IF PL = 0 THEN 21118
21110  PRINT TXT$(PTR(PL - 1))
21115  IF N = 1 THEN 21510
21118  FOR I = 0 TO N - 2
21125  IF PU + I > SIZE THEN 2140
    0
21127  IF I -  INT (I / 20) * 20 <
    19 THEN 21130
21129  INPUT SW$
21130  PRINT TXT$(PTR(PU + I))
21160  NEXT I
21165  GOTO 21510
21400  PRINT
21500  PRINT "END OF FILE"
21510  GOTO 1000
27000  REM ************************
    *
27010  REM REPLACE COMMAND
27020  REM
27100  REM LOOP
```

BATE プログラム リスト

```
27110  IF PL = 0 THEN 1000
27120  PRINT TXT$(PTR(PL - 1))
27130  INPUT "->";STRG$
27150 TXT$(PTR(PL - 1)) = STRG$
27160  GOTO 1000
28000  REM **********
28010  REM SAVE COMMAND
28020  REM
28100  INPUT "FILE NAME>";NM$
28110  IF NM$ = "" THEN 28200
28120 NAME$ = NM$
28200  GOSUB 1500
28210  GOTO 1000
29000  REM **********
29010  REM TOP COMMAND
29020  REM
29090  IF PL = 0 THEN 1000
29100  FOR I = 1 TO PL
29110 PTR(PU - I) = PTR(PL - I)
29120  NEXT I
29130 PU = PU - PL
29140 PL = 0
29150  GOTO 1000
31000  REM **********
31010  REM UP COMMAND
31020  REM
31100  INPUT "I>";N
31105  IF N <  = 0 THEN 1000
31110  FOR I = 1 TO N
31120  IF PL < I THEN 31500
31130 PTR(PU - I) = PTR(PL - I)
31140  NEXT I
31150 PL = PL - N
31160 PU = PU - N
31170  GOTO 1000
31500  PRINT "BEGINNING OF FILE"
31510 PL = PL - I + 1
31520 PU = PU - I + 1
31530  GOTO 1000
32000  REM **********
32010  REM VERIFY COMMAND
32020  REM
32100  INPUT "ON/OFF?";SW$
32110  IF SW$ <  > "ON" THEN 3214
       0
32120 VRFY = 1
32130  GOTO 1000
32140  IF SW$ <  > "OFF" THEN 321
       00
32150 VRFY = 0
32160  GOTO 1000
36000  REM **********
36010  REM DELETE COMMAND
36020  REM
36100  INPUT "I>";N
36102  IF N < 6 THEN 36110
36103  PRINT N;" LINES?(Y/N)"
36104  INPUT SW$
36106  IF SW$ = "Y" OR SW$ = "YES
       " THEN 36110
36108  GOTO 1000
36110 PU = PU + N
36120  IF PU <  = SIZE THEN 1000
36130  PRINT "END OF FILE"
36140 PU = SIZE + 1
36150  GOTO 1000
37000  REM **********
37010  REM READ COMMAND
37020  REM
37100  INPUT "FILE NAME>";NAME$
37110  PRINT D$;"OPEN ";NAME$
37120  PRINT D$;"READ ";NAME$
37125 PU = SIZE + 1
37130  FOR I = 0 TO SIZE
37140  INPUT TXT$(I)
37150  IF TXT$(I) = "/*" THEN 375
       00
37160 PTR(I) = I
37170  NEXT I
37175  PRINT D$;"CLOSE ";NAME$
37180  PRINT "STORAGE OVERFLOW"
37190 PL = SIZE
37200  GOTO 37510
37500 PL = I
37505 MAX = I - 1
37510  PRINT D$;"CLOSE ";NAME$
37530  GOTO 1000
38000  REM **********
38010  REM CHANGEALL COMMAND
38020  REM
38100  INPUT "S,S)";SRC$,DST$
38110 L =  LEN (SRC$)
38120 N = MAX
38130 CNT = 0
38140  FOR J = PU TO SIZE
38150  FOR I = 1 TO  LEN (TXT$(PT
       R(J))) - L + 1
38155  IF  MID$ (TXT$(PTR(J)),I,L
       ) <  > SRC$ THEN 38300
38160 CNT = CNT + 1
38170 LL = I - 1
38180 RL =  LEN (TXT$(PTR(J))) -
      I - L + 1
38190  IF LL > 0 AND RL > 0 THEN
       38240
38200  IF RL > 0 THEN 38230
38202  IF LL > 0 THEN 38210
38204 TXT$(PTR(J)) = DST$
38206  GOTO 38260
38210 TXT$(PTR(J)) =  LEFT$ (TXT$
      (PTR(J)),LL) + DST$
38215  GOTO 38260
38230 TXT$(PTR(J)) = DST$ +  RIGHT$
      (TXT$(PTR(J)),RL)
38235  GOTO 38260
38240 TXT$(PTR(J)) =  LEFT$ (TXT$
      (PTR(J)),LL) + DST$ +  RIGHT$
      (TXT$(PTR(J)),RL)
38260 I = I + L - 1: REM BAD ASSI
      GNMENT
38270  IF VRFY = 0 THEN 38300
38280  PRINT TXT$(PTR(J))
38300  NEXT I
38310  NEXT J
38500  PRINT CNT;" STRINGS CHANGE
       D"
38510  GOTO 1000
39000  REM **********
39010  REM END COMMAND
39020  REM
39100  INPUT "SAVE FILE?";NAME$
39110  IF NAME$ = "" THEN 39130
39120  GOSUB 1500
39130  PRINT "END BATE"
39140  END
50000  REM **********
50010  REM BREAK PACKAGE
50020  REM
50022  PRINT
50024  PRINT "*** ERROR ***"
50025  GOTO 1000
```