```
program driver (input, output);
{Reads a file called AFRWORDS of words arranged one to a line, each ending with a carriage    }
{return code, calculates all the hyphenation points in it, and stores the hyphenated version of word }
{in a file named HYPWORDS.                                                                    }
{Maximum length of a word is 64 characters. Any letters in audition are ignored. Letters may be }
{upper and lower case. Diacritics are represented a separate character before the. character which }
{they accent. Diaresis is " Circumflex is ^ Grave is ` Acute is '                             }
{Assume existence of type string = array[1..256] of char;                                     }
{Written by Quintin Gee, Computer Science Department,                                         }
{University of the Witwatersrand, Johannesburg, RSA.                                          }
{August 1987                                                                                  }
{Hardware used: Apple Macintosh connect to fileserver via AppleTalk network.                  }
{Software used: Lightspeed Pascal                                                             }
{Compiled version occupies 672 bytes of memory                                                }
{Uses procedures: afrhyp, insert                                                              }

var word        : string;          {unadulterated word to be tested                           }
 endofword      : integer;         {address of last character of word                         }
 optpoint       : integer;         {address of last character on line                         }
 source         : text;            {file of words to be tested                                }
 done           : text;            {file of resulting hyphenations                            }
 result         : integer;         {zero if no hyphenation point. value gives                 }
                                   {character number after which hyphenation                  }
                                   {can take place                                            }
 prevresult     : integer;         {previous hyphenation point                                }
 breaklist      : array[1..32] of integer; {list of all hyphenation points found              }
 i              : integer;         {counter for breaklist                                     }
 hyphen         : string;          {constant hyphen character                                 }
 entry          : boolean;         {true for initialisation, false after                      }

  procedure afrhyp (a : string; b, c : integer; d : boolean; var e : integer);
  external;

  {procedure insert                 is a system routine taking three arguments.              }
  {             (a      : string    constant hyphen stored as string                        }
  {             b       : string    word to be acted upon                                   }
  {             c       : integer); position in word for hyphen to be inserted              }

begin
reset(source, 'afrwords');
rewrite(done, 'hypwords');
hyphen := '-'
entry := true;
afrhyp(word, optpoint, endofword, entry, result);
entry := false;
repeat
   readln(source, word);
   endofword := length(word);
   i := 1;
   breaklist[1] := 0;
   prevresult := 0;
   for optpoint := 2 to length(word) - 2 do
      begin
      afrhyp(word,optpoint,endofword,entry,result);
      if ((result <> 0) and (result <> prevresult))
         then begin breaklist[i] := result + 1; prevresult := result; i := i + 1 end;
      end;
```

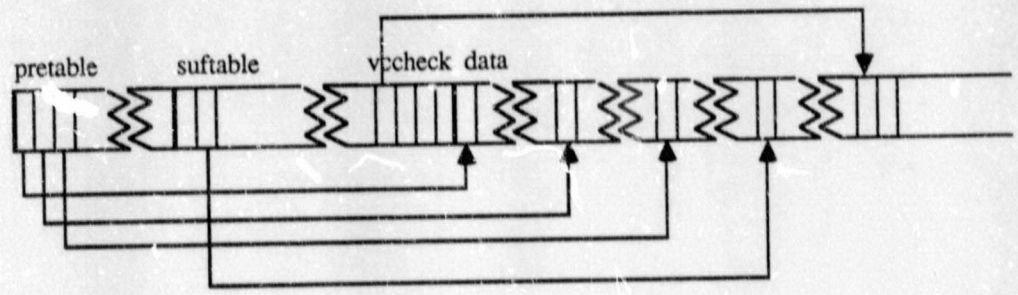
```
  if i <> 1 then repeat i := i - 1; insert(hyphen,word,breaklist[i]) until i = 1;
  writeln(done,word);
until eof(source);
end.
```

9.2 Loading the Tables

The various logic tables are stored in human-readable character format. Extracts are shown in the Appendix.

In order to make the tables more easily accessible to the procedure, this character is converted into a binary format by a program called *setuptable*. At the same time, the program also sets up the pointers to the various components in the tables. These are all contained in one record and written to a file. As a result, the hyphenation procedure is able to read one record using *read_tables* and sets up the locations of the tables at the same time.

Thus the first item in the record is the number which is the byte count of the first character of the prefixes beginning with A. The second item points to those beginning with B, and so on. The 27th item points to the suffixes ending with A etc. The 53rd to 56th items point to the locations where each of the 4 V/C tables starts.



```
program setuptable (input, output);
{Reads a file called SOURCETABLE                                                    }
{generates a file called BINARYTABLE which contains one record only.                }
{Written by Quintin Gee, Computer Science Department,                               }
{University of the Witwatersrand, Johannesburg, RSA.                                }
{August 1987                                                                        }
{Hardware used: Apple Macintosh connect to fileserver via AppleTalk network.        }
{Software used: Lightspeed Pascal                                                   }
{Compiled version occupies 858 bytes of memory                                      }

type
  index          = 1..6500;
  letter         = 'A'..'Z';
  pretables      = array[letter] of index;        {table of 26 numbers pointing to binary file  }
  suftables      = array[letter] of index;          {table of 26 numbers                        }
  vctables       = array[1..4] of index;            {table of 4 numbers                         }
  longstring     = packed array[index] of char;{binary file of character tables                 }
  datarecs       = record
                     pretable    : pretables;
                     suftable    : suftables;
                     vccheck     : vctables;
                     data        : longstring;
                   end;

var sourcefile   : text;
    datarec      : datarecs;
    destfile     : file of datarecs;
    count        : index;
    i            : letter;
    j            : integer
    ch           : char;

  procedure readsource;
  begin
  repeat read(sourcefile, ch); data[count] := ch; count := count + 1 until data[count - 1] = '*'
  end;

begin
reset(sourcefile, 'sourcetables');
rewrite(destfile, 'binarytable');
count := 1;
with datarec do
begin
for i := 'A' to 'Z' do begin pretable[i] := count; readsource end;
for i := 'A' to 'Z' do begin suftable[i] := count; readsource end;
for j := 1 to 4 do begin vccheck[j] := count; readsource end;
end;
write(destfile,datarec);
close(sourcefile);
close(destfile);
end.
```

Chapter 10 RESULTS

10.1 Performance

Research for the morphological aspects of this work was conducted on a word list supplied by Atex Systems using their Atex 7000 series computer with Document Processing System©.

The exact size of the procedure will vary according to the language of implementation, but when written in Lightspeed Pascal for the Apple Macintosh it occupied 6996 bytes. If it was programmed in Assembler it should use considerably less, and we expect that the tables and procedure would occupy about 6500 bytes. We compare this with the more popular word processors available for this computer. MacWrite occupies about 78 KB and Microsoft Word about 350 KB. This means that the hyphenation procedure would increase these by 9 % and 2 % respectively.

Speed of execution on the Apple Macintosh was measured using Ticktime and generated all the hyphens in 273 words in 235 seconds.

We attempted to incorporate the procedure in manufacturers' word processing software, but with limited success. In most cases the supplier was unwilling to provide information of suitable points in their code where the procedure could be incorporated. In some cases those contacted suggested that their coding internal systems were not available to external parties. In one or two cases the supplier was willing to cooperate in including this work as part of their software, but no results are yet forthcoming on this point.

The process of including a hyphenation procedure are as follows. There must exist, in the word processing software, a suitable line end handling 'hook', where an instruction can be inserted to transfer control to the procedure. The procedure must be coded in the native language used by the supplier, and return control to the originating point, or to some supplier-specified point where the assembly of the next line is started. The normalisation must be rewritten to strip unnecessary characters out of the word, and to translate them into the representation given in this work. Alternatively, the supplier's native representation may be used throughout, with a

little more handling added to some of the search routines. It is strongly suggested that the data tables be hard coded in the program, the pointers being calculated manually before compilation. This avoids special disc handling, other than using overlaying for the procedure if required.

If there is insufficient space in memory to accommodate both the procedure and its tables, then a scatter storage method could be adopted, such as those mentioned in Malone (1978), for storing the tables on disc and accessing them only when needed.

10.2 Extended Word List

Two word lists were used for a test. The first consisted of the 357 highest frequency words in Afrikaans, as supplied to Primary School teachers by the Johannesburg College of Education. There were no errors in hyphenation of this list.

The second list consisted of 20 000 words supplied by the Computer Centre of the University of the Witwatersrand. Of these, 8 % of the words were found to be incorrectly hyphenated, of which the following is a typical sample demonstrating the actual occurrence and the corrected version.

Wrong	Right
om-ska-ke-lingsa-lgorit-me	oms!akelings-al-go-ritme
noem-en-swaar-di-ge	noe-mens-waardige
suk-se-spers-en-ta-sie	sukses-persentasie
in-lig-ting-snet-werk	inligtings-netwerk
in-lig-tings-hul-pbron-ne	inligtingshulp-bronne
be-ken-dma-king	bekend-making
te-le-ksma-sjie-ne	teleks-masjiene

It can be seen that the major problem is with the interface between the various components of a compound word. This was as expected.

10.2 Continuous Paragraph

As a suitable test for presentation, we show below 1 John 2 : 1-11 extracted from Die Nuwe Testament en die Psalms, 1979-vertaling, Bybelgenootskap van Suid-Afrika, Cape Town, 1979. This is reprinted below, retyped by us.

> Dit skrywe ek aan julle, my liewe kinders, dat julle nie moet sondig nie. En as een van ons sondig – ons het Jesus Christus, die *regverdige, as ons *voorspraak by die Vader. [2]Hy is die *versoening vir ons sondes; en nie net vir óns sondes nie, maar ook vir dié van die hele wêreld. [3]As ons die gebooie van God gehoorsaam, weet ons daaraan dat ons Hom ken. [4]Iemand wat sê: „Ek ken Hom", maar nie sy gebooie gehoorsaam nie, is 'n leuenaar, en die *waarheid is nie in hom nie. [5]Wie sy woord egter gehoorsaam – in hom het die liefde van God werklik sy doel volkome bereik. Hieraan weet ons dat ons in Hom is. [6]Wie beweer dat hy in Hom bly, behoort self ook te lewe soos Jesus gelewe het.
>
> [7]Geliefdes, dit is nie 'n nuwe gebod wat ek aan julle voorskrywe nie, maar 'n ou gebod wat julle van die begin af gehad het. Die ou gebod is die boodskap wat julle gehoor het. [8]Tog is dit ook 'n nuwe gebod wat ek aan julle voorskrywe. Dat dit werklik so is, is te sien in Jesus en ook in julle, want die duisternis is aan die verbygaan, en die ware lig skyn alreeds. [9]As iemand beweer dat hy in die lig is, maar hy haat sy broer, is hy nog steeds in die duisternis. [10]Wie sy broer liefhet, bly in die lig, en daar is niks wat hom laat struikel nie. [11]Maar wie sy broer haat, is in die duisternis en lewe in die duisternis en weet nie waar hy gaan uitkom nie, omdat die duisternis hom blind gemaak het.

We note an error in the original typesetting; the period at the end of verse 2 was not printed at the end of the line as it should be, but was followed by a space.

This sample contains the following typographic features.

Punctuation	period, comma, en dash, semicolon, colon, double close quote, double inferior open quote.
Special characters	superior star. superior numerals, circumflex, acute, 'n.

After apply-ing the Afrikaans hyphenation algorithm discussed here to every word we get the following. All the hyphens detected have been inserted in the continuous text.

> Dit skry-we ek aan jul-le, my lie-we kin-ders, dat jul-le nie moet son-dig nie. En as een van ons son-dig – ons het Je-sus Chris-tus, die *reg-ver-di-ge, as ons *voor-spraak by die Va-der. [2]Hy is die *ver-soe-ning vir ons son-des; en nie net vir óns son-des nie, maar ook vir dié van die he-le wê-reld. [3]As ons die ge-booie van God ge-hoor-saam, weet ons daar-aan dat ons Hom ken. [4]Ie-mand wat sê: „Ek ken Hom", maar nie sy ge-booie ge-hoor-saam nie, is 'n leue-naar, en die *waar-heid is nie in hom nie. [5]Wie sy woord eg-ter ge-hoor-saam – in hom het die lief-de van God werk-lik sy doel vol-ko-me be-reik. Hier-aan weet ons dat ons in Hom is. [6]Wie be-weer dat hy in Hom bly, be-hoort self ook te le-we soos Je-sus ge-le-we het.
>
> [7]Ge-lief-des, dit is nie 'n nu-we ge-bod wat ek aan jul-le voor-skry-we nie, maar 'n ou ge-bod wat jul-le van die be-gin af ge-had het. Die ou ge-bod is die bood-skap wat jul-le ge-hoor het. [8]Tog is dit ook 'n nu-we ge-bod wat ek aan jul-le voor-skry-we. Dat dit werk-lik so is, is te sien in Je-sus en ook in jul-le, want die duis-ter-nis is aan die ver-by-gaan, en die wa-re lig skyn al-reeds. [9]As ie-mand be-weer dat hy in die lig is, maar hy haat sy broer, is hy nog steeds in die duis-ter-nis. [10]Wie sy broer lief-het, bly in die lig, en daar is niks wat hom laat stru-kel nie. [11]Maar wie sy broer haat, is in die duis-ter-nis en le-we in die duis-ter-nis en weet nie waar hy gaan uit-kom nie, om-dat die duis-ter-nis hom blind ge-maak het.

In this passage no incorrect hyphens have been inserted, and there are no missing hyphens.

We note the word LEUENAAR. It would seem to have a syllable break LEU-EN but a hyphenation point LEUE-NAAR.

Chapter 11 IMPROVEMENTS

11.1 Linguistics

Since the syllable identification performed here has been partly based on well-formed rules and partly based on a pragmatic approach, many rare words have been included as 'non-conforming' in the tables. It could be argued that this is an unnecessary overhead when the objective is to make the tables as small as possible. One way to reduce their size would therefore be to eliminate these rare items, at the cost of occasionally experiencing an incorrect hyphenation. This may well be acceptable in the case of a newspaper, where the breadth of subject matter is wide. In most other work, however, much of the vocabulary depends on the subject being written about and so word frequency order is quite volatile after the first 500 or so common words. If one of main subject words is incorrectly hyphenated, its relatively high frequency in the passage concerned may contribute to a disproportionate error rate.

Additional saving of table space could be obtained by eliminating prefixes and suffixes that hyphenate correctly when V/C checking is applied to them. Thus -LI-KE and MO-NO- are typical CVCV strings which could be tackled by the syllable structure tables without resort to special prefix or suffix tables.

It is clear that further work is necessary on compound boundaries, in particular the features that identify the Consonant/Vowel, Vowel/Consonant, and Consonant/-Consonant divisions at boundaries.

We assumed that because the circumflex affected phonetic value, this accent should be included when setting up the hyphenation tables. However, we found that it did not occur in the tables. It should therefore now be dropped, as it appears unnecessary.

11.2 Implementation

We expected the tables to occupy a few hundred bytes of memory, and thus be written into the procedure directly. The exceptions have made the tables so large that a separate file system had to be constructed to handle the resulting tables. Several approaches can be adopted to reduce their size, and combinations of approach may be possible. The tables can be condensed further by the following actions.

1. Eliminate the search letter key. Thus instead of looking in prefix table M for a match with MONO and MONOF, we store only the ONO and ONOF.

2. Store the table elements using a hash coding technique, or tree storage. Some recent work has been performed in this area, notably Chang (1986) but his method soon becomes unwieldy with a large base of words. A report by Holman (1987) has extended this to a pragmatic approach useful for up to 5 000 words.

3. Limit exceptions to words that occur in practice. This has been covered in 11.1.

4. Reduce the tables to bit coding. It should be possible to use 6 bits to represent each table character, which would save 25% of the space, at the cost of some extra programming instructions.

The second area where implementation can have a significant effect on performance is that of programming. The restrictions of the Lightspeed Pascal compiler for the Macintosh do not permit, for example, indexed variables to be passed to procedures. Thus instead of writing

```
search(table.pretable[hword[front]],front,found);  or      search(i,back–2,found);
```

we are forced to provide a dummy variable

```
i := table.pretable[hword[front]];                  or     datum := back – 2;
search(i,front,found);                                     search(i,datum,found);
```

Another deficiency of Pascal is that there is no way to escape from a loop operation, when there are two or more possible termination events. We have to create a dummy flag which is set by either of the events, and is then tested to determine whether to exit from the loop.

We believe that the algorithm can be made considerably more succinct by using LISP, but unfortunately at the expense of being able to communicate the logic to general implementors in South Africa.

11.3 Extension to other languages

The method we have described of tackling the hyphenation of Afrikaans can also be applied to the handling of other languages. In general, they are characterised by the term coined by Humboldt as **agglutinative**; that is, they combine morphemes and roots together into larger words in contrast to non-agglutinative languages which leave them as separate words in a sentence. Examples are German, Russian and Dutch. The same method could also be applied to other European languages with a much simplified approach. For example, the English prefix table has only 33 components with 37 exceptional cases.

11.4 Other uses

We believe that the algorithm can be applied in other text processing areas since the procedure undertakes some morphological analysis of the language. This is exactly the requirement for one implementation of a spelling checker, in order to condense the spelling dictionary and provide more efficient working. Other applications in a similar area can also be envisaged e.g. thesauri.

Text compression relies on efficient coding of features with different relative frequencies. The main methods given by Witten (1987) rely on whether the features are known beforehand (non-adaptive coding), or must be calculated as the text to be compressed is processed (adaptive coding). In a given language, the morphology is a fixed feature and can be automatically recognised using the algorithm we have suggested. This can help make the coding system more efficient.

The same analysis can, in a wider context, be useful as a small part of natural language processing, forming a bridge between the phonemics and syntax.

APPENDIXES

CONDENSED PREFIX TABLES

ABRA90ADR90AB/DA/E/I/O/U92AGUUR90ALEER90ALO"I91ALO90ANALF90
ANESTETIKUM90ANIOCN90ANORG90ASOF90ASOOK90AANDAG93AANDIK93
AANDRA93AANDUI93AANDEEL93AANDI/OEN93AANDRUK93AANDRING93
AANDRYF93AANDURF93AANDENKING93AAND90AAN93AARTS95A"E91
AFRIC/K90AFERESE90AFASIE90AFONIE91AF92AGTER95ALLO94AMB/FI94
ANALF90ANATT90ANA-M/R/S93ANTE-NNE94ANTI-EK94APO-S93
ARGE-N94ARGI-E94*BEATNIK90BEDWELM92BEDWA/I/ONG92
BEDB/D/J/K/L/S/T/V/W90BE"E92BEE/F90BEIAARD90BEID/GE90BEIER90
BEITS90BEKAF90BEKFL90BEKK/S/V90BEKPR90BEKRIEM90BELBREI90BELG90
BELHAM90BELK/L/R/S/T/V90BENDE90BENG/J/N/S/T90
BERB/D/G/K/L/R/S90BESKIN90BESS90BESTIAL90BESWIL90BETSJ90BETWE90
BEU90BE92BIO-PSIE93BIBB90BIDD/S90BIE90BIGG90BILA92BIN@E92BINO92
BISA/E/O92BITU92BI-E/L/N/S/T92BRAGI95BYLAE92BYLANGE92BYL@E92
BYLEER92BYLEGGING92BYLOOP92BYLYN92BYSTER90BYTEL/R92BYTEND92
BYTERIG92BYTOON92BYTREK92BY-L/T92**DAAR94DERI/C90DERMAF90
DERMB/K/O/S90DESEM/N/R90DESILL93DESINF/T93DESI90DESK/TR90
DESOL90DEUR-E94DEEG/L/M/R/S90DEFTIG90DEIK90DEKBL90
DEKG/H/K/M/P/S/T/V90DEKLAAG90DEKOES90DEKRIET90DELF/G/S/T/W90
DEM/PP90DEND/K/N/S/T90DERM90DEUG/N/S/T90DER/S93DE92DIAR90
DISPENSASIE93DISEN90DISPENS90DIA/S93*EENDS/V90EEN-SD/G/K/L93
EKSA/E/I/O90EKSPAN90EKSPEK90EKSPERT90EKSPI90EKSTERI/R93
EKSTRAD/H93EKSTRO/U93EKSTA/R90EKSTER90EKWI94ENDO-SS94ERFENIS92
ERTJIE90ERN/TS90ER-D/E/F/I/O/U92ETNO94**GEO93GE"E/I/U92
GEKLIKKLAK92GESPESI92GEKLIK90GEKH/K/S90GELD/L90GEMM/S90GENRE90
GENS/T90GERE"EL90GERF/M90GESP/TE90GESTIK90GE-E/I/U92
*HEKSA-AN95HEMI94HEREK/N93HERDS90HERE/F90HERO"90HEROUT90HER93
HETERO-PS96HIER-TS94HIPER95HIPO-ST94HOMO94*IDIO94