NOFILL Program

Lynne A. Price
Patrick Milligan

BNR INC.,
subsidiary of Bell-Northern Research, Ltd.

Below is the source for a SAIL program called NOFILL. This program reads an ASCII text file and outputs a file that can be processed by TEX in order to typeset a listing of the original file. Line spacing is preserved and characters which have special meaning to TEX (*e.g.*, backslash and braces) do not cause problems. This program is very useful to documenters of TEX macros who wish to prepare lengthy examples of corresponding input and output. Preparation of documents that include sample programs in various programming languages is also simplified with this program. For example, the listing shown here was itself generated by NOFILL. Following the program listing is some test data that can be used to verify this code and the corresponding output.

Source of NOFILL:

```
begin "nofill"
comment This program generates a TeX input file that typesets ASCII
  text files (e.g., TeX macro source).  The output file  has the form
          {\ty\save9\hbox{A}
          \def \#  {{\ty\char`043}}  % Defines for special characters
              .
              .
              .
          \def \up {{\ty\char`136}}
          \hbox{first line}
          \hbox{second line}
              .
              .
              .
          \hbox{last line}
          \hbox{}
          }
  The call to \ty on the first line is assumed to select an appropriate font,
  usually a fixed-width typwriter font.  The width of any character (the
  letter "A", for instance) can then be used to space special characters
  from other fonts.  Box 9 is used to save this width.
  The text of each line is copied directly from the input file to the output
```

```
   file with special characters changed to macro calls (for example, a space
   in the input is output as "\ " and a backslash is output as "\\".  When
   long input lines occur, or when an input line contains several special
   characters that expand to macros with long names, one \hbox call may
   be output over several lines in the output file.  The  macros used to output
   special characters are controlled by the SAIL macro "chartable" defined
   below. Users at different sites or with different applications may wish to
   modify this table.
   ;

define crlf = "^15&^12";
define cr = "^15" ;  define lf = "^12" ;
define normal = "0", foundcr = "1", foundtab = "2", special = "3"  ;

comment Output lines are broken after maxlen characters (or after the
   first control sequence that extends past maxlen characters).  This cutoff
   makes output files easier to read on terminals with narrow screens and
   also prevents the generation of TeX input lines longer than the maximum
   150 characters;
define maxlen = "60" ;

require "<><>" delimiters ;
comment The following table defines the processing performed on each input
   character.  The SAIL macro "action" has three parameters: the name of the
   ASCII character, the name of an action to be performed when the character
   is encountered (used to control the case statement in the main program loop),
   and, for special characters, a string to be output when the character is
   encountered.  The order of actions in this table is significant--when
   changes are made, the order must be preserved.
   ;
define chartable=<
   action(<NUL>,<special>,<"\up @">),
   action(<^A (SOH)>,<special>,<"\up A">),
   action(<^B (STX)>,<special>,<"\up B">),
   action(<^C (ETX)>,<special>,<"\up C">),
   action(<^D (EOT)>,<special>,<"\up D">),
   action(<^E (ENQ)>,<special>,<"\up E">),
   action(<^F (ACK)>,<special>,<"\up F">),
   action(<^G (BEL)>,<special>,<"\up G">),
   action(<^H (BS)>,<special>,<"\up H">),
   action(<^I (HT)>,<foundtab>,<"\up I">),
   action(<^J (LF)>,<special>,<"\up J">),
   action(<^K (VT)>,<special>,<"\up K">),
   action(<^L (FF)>,<special>,<"\up L">),
   action(<^M (CR)>,<foundcr>,<"\up M">),
   action(<^N (SO)>,<special>,<"\up N">),
   action(<^O (SI)>,<special>,<"\up O">),
   action(<^P (DLE)>,<special>,<"\up P">),
   action(<^Q (DC1)>,<special>,<"\up Q">),
   action(<^R (DC2)>,<special>,<"\up R">),
   action(<^S (DC3)>,<special>,<"\up S">),
   action(<^T (DC4)>,<special>,<"\up T">),
   action(<^U (NAK)>,<special>,<"\up U">),
```

```
  action(<^V (SYN)>,<special>,<"\up V">),
  action(<^W (ETB)>,<special>,<"\up W">),
  action(<^X (CAN)>,<special>,<"\up X">),
  action(<^Y (EM)>,<special>,<"\up Y">),
  action(<^Z (SUB)>,<special>,<"\up Z">),
  action(<ESC>,<special>,<"\up [">),
  action(<FS>,<special>,<"\up \\">),
  action(<GS>,<special>,<"\up ]">),
  action(<RS>,<special>,<"\up \^">),
  action(<US>,<special>,<"\up \_">),
  action(<space>,<special>,<"\ ">),
  action(<!>,<normal>,<"">),
  action(<">,<normal>,<"">),
  action(<#>,<special>,<"\#">),
  action(<$>,<special>,<"\$">),
  action(<%>,<special>,<"\%">),
  action(<&>,<normal>,<"">),
  action(<'>,<special>,<"\'">),
  action(<(>,<normal>,<"">),
  action(<)>,<normal>,<"">),
  action(<*>,<normal>,<"">),
  action(<+>,<normal>,<"">),
  action(<,>,<normal>,<"">),
  action(<->,<normal>,<"">),
  action(<.>,<normal>,<"">),
  action(</>,<normal>,<"">),
  [10] action(<digits>,<normal>,<"">),
  action(<:>,<normal>,<"">),
  action(<;>,<normal>,<"">),
  action(<less than>,<normal>,<"">),
  action(<=>,<normal>,<"">),
  action(<greater than>,<normal>,<"">),
  action(<?>,<normal>,<"">),
  action(<@>,<normal>,<"">),
  [26] action(<upper case letters>,<normal>,<"">),
  action(<[>,<normal>,<"">),
  action(<\>,<special>,<"\\">),
  action(<]>,<normal>,<"">),
  action(<^>,<special>,<"\^">),
  action(<_>,<special>,<"\_">),
  action(<`>,<special>,<"\`">),
  [26] action(<lower case letters>,<normal>,<"">),
  action(<{>,<special>,<"\{">),
  action(<|>,<special>,<"\|">),
  action(<}>,<special>,<"\}">),
  action(<~>,<special>,<"\~">),
  action(<DEL>,<special>,<"\up ?">)
  >;
define action(a,b,c) = <b>;
preloadfXwith chartable;
integer array states[0:127] ;
redefine action(a,b,c) = <c> ;
preloadfXwith chartable;
```

```
string array strings[0:127] ;

external integer !SKIP! ;

string infile, outfile, nextchar;
integer chan, dbreak, deof, table, state, outcnt, incnt, i, spacestotab ;
boolean skipnext ;

procedure cntprint(string s); comment Output a string and count its length;
  begin
    outcnt := outcnt + length(s) ;
    print(s) ;
  end ;

!SKIP! := TRUE ;
print("NOFILL",crlf) ;
while  !SKIP! do begin
  print ("Input file name? ");
  infile := intty;
  lookup(chan,infile,deof);
  chan := openfile(infile, "ROE") ;
  if  !SKIP! then begin
    infile := infile&".TEX";
    lookup(chan,infile,deof);
    chan := openfile(infile, "ROE") ;
    if !SKIP! then print(infile," bad. Try again...",crlf);
    end ;
  end;
setinput(chan,1,dbreak, deof);
print("Output file (Default: NOFILL.TEX) ? ");
if (outfile:=intty) = "" then outfile:="NOFILL.TEX" ;
setprint(outfile,"F") ;

print("{\ty",crlf) ;
print("\def \#  {{\ty\char`043}}", crlf) ;
print("\def \$  {{\ty\char`044}}", crlf) ;
print("\def \%  {{\ty\char`045}}", crlf) ;
print("\def \`  {{\ty\char`015}}", crlf) ;
print("\def \\  {{\ty\char`134}}", crlf) ;
print("\def \^  {{\ty\char`017}}", crlf) ;
print("\def \_  {{\ty\char`032}}", crlf) ;
print("\def \~  {{\ty\char`016}}", crlf) ;
print("\def \{  {{\ty\char`173}}", crlf) ;
print("\def \|  {{\ty\char`174}}", crlf) ;
print("\def \}  {{\ty\char`176}}", crlf) ;
print("\def \~  {{\ty\char`024}}", crlf) ;

print("\def \up {{\ty\char`136}}", crlf) ;

print(crlf, "\hbox{") ;

setbreak(table := getbreak,"","", "I") ;
deof := 0 ;
```

NOFILL Program

```
nextchar := input(chan, table) ;
incnt:= 1 ; comment incnt counts characters on input line ;
outcnt:= length("\hbox{") ;  comment outcnt counts characters on output line ;
while NOT deof do begin
  if outcnt > maxlen then begin
    print("\!" & crlf) ;
    outcnt := 0 ;
    end ;
  skipnext := FALSE ;  comment used for "lookahead" in cr-lf pairs ;
  case states[nextchar] of begin
    [normal]
      begin
        cntprint(nextchar) ;
      end ;
    [special]
      begin
        cntprint(strings[nextchar]) ;
      end ;
    [foundcr]
      begin
        nextchar := input(chan, table) ;
        if nextchar = lf then begin
          print("}", crlf, "\hbox{") ;
          incnt := 0 ;
          outcnt := length("\hbox{") ;
          end
        else begin
          cntprint(strings[cr]) ;
          skipnext := TRUE ;
          end ;
      end ;
    [foundtab]
      begin
        spacestotab := 8 - ((incnt - 1) MOD 8) ;
        for i := 1 step 1 until spacestotab do cntprint("\ ") ;
        incnt := incnt + spacestotab - 1 ;
      end
    end ;
  if NOT skipnext then nextchar := input(chan, table) ;
  incnt := incnt + 1 ;
  end ;
print("}}") ;
end "nofill";
```

Test data for NOFILL:

ASCII Test:

```
↑A↑B↑C↑D↑E↑F↑G↑H        ↑J↑K↑L↑M↑N↑O↑P↑Q↑R↑S↑T↑U↑V↑W↑X↑Y↑Z↑[↑\↑]↑^↑_
 !"#$%&' ()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~↑?
```

```
Tab test:

12345678901234567890123456789012345678901234567890
.       .       .       .       .       .       .       (tabs)
a       b       c       d       e       f       g
aa      bb      cc      dd      ee      ff      gg
aaa     bbb     ccc     ddd     eee     fff     ggg
aaaa    bbbb    cccc    dddd    eeee    ffff    gggg
aaaaa   bbbbb   ccccc   ddddd   eeeee   fffff   ggggg
aaaaaa  bbbbbb  cccccc  dddddd  eeeeee  ffffff  gggggg
aaaaaaa bbbbbbb ccccccc ddddddd eeeeeee fffffff ggggggg
.       .       .       .       .       .       .       (spaces and tabs)
a       b       c       d       e       f       g
 a       b       c       d       e       f       g
  a       b       c       d       e       f       g
   a       b       c       d       e       f       g
    a       b       c       d       e       f       g
     a       b       c       d       e       f       g
      a       b       c       d       e       f       g
.       .       .       .       .       .       .       (spaces)
.               .               .               .       (multiple tabs)
```

Test data output from NUFILL:

```
{\ty
\def \#  {{\ty\char`043}}
\def \$  {{\ty\char`044}}
\def \%  {{\ty\char`045}}
\def \`  {{\ty\char`015}}
\def \\  {{\ty\char`134}}
\def \^  {{\ty\char`017}}
\def \_  {{\ty\char`032}}
\def \'  {{\ty\char`016}}
\def \{  {{\ty\char`173}}
\def \|  {{\ty\char`174}}
\def \}  {{\ty\char`176}}
\def \~  {{\ty\char`024}}
\def \up {{\ty\char`136}}

\hbox{ASCII\ Test:}
\hbox{}
\hbox{\up A\up B\up C\up D\up E\up F\up G\up H\ \ \ \ \ \ \ \ \!
\up J\up K\up L\up M\up N\up O\up P\up Q\up R\up S\up T\up U\up V\!
\up W\up X\up Y\up Z\up [\up \\\up ]\up \^\up \_}
\hbox{\ !"\#\$\%\&\'()*+,-./0123456789:;<=>?}
\hbox{@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]\^\_}
\hbox{\`abcdefghijklmnopqrstuvwxyz\{\|\}\~\up ?}
\hbox{}
\hbox{}
```

NOFILL Program

```
\hbox{Tab\ test:}
\hbox{}
\hbox{12345678901234567890123456789012345678901234567890}
\hbox{.\ \ \ \ \ \ \ .\ \ \ \ \ \ \ .\ \ \ \ \ \ \ .\ \ \ \ \ \ \ \!
.\ \ \ \ \ \ \ .\ \ \ \ \ \ \ .\ \ \ \ \ \ \ (tabs)}
\hbox{a\ \ \ \ \ \ \ b\ \ \ \ \ \ \ c\ \ \ \ \ \ \ d\ \ \ \ \ \ \ \!
e\ \ \ \ \ \ \ f\ \ \ \ \ \ \ g}
\hbox{aa\ \ \ \ \ \ bb\ \ \ \ \ \ cc\ \ \ \ \ \ dd\ \ \ \ \ \ \!
ee\ \ \ \ \ \ ff\ \ \ \ \ \ gg}
\hbox{aaa\ \ \ \ \ bbb\ \ \ \ \ ccc\ \ \ \ \ ddd\ \ \ \ \ eee\!
\ \ \ \ \ fff\ \ \ \ \ ggg}
\hbox{aaaa\ \ \ \ bbbb\ \ \ \ cccc\ \ \ \ dddd\ \ \ \ eeee\ \ \ \ \!
ffff\ \ \ \ gggg}
\hbox{aaaaa\ \ \ bbbbb\ \ \ ccccc\ \ \ ddddd\ \ \ eeeee\ \ \ \!
fffff\ \ \ ggggg}
\hbox{aaaaaa\ \ bbbbbb\ \ cccccc\ \ dddddd\ \ eeeeee\ \ ffffff\!
f\ \ gggggg}
\hbox{aaaaaaa\ bbbbbbb\ ccccccc\ ddddddd\ eeeeeee\ fffffff\ g\!
ggggggg}
\hbox{.\ \ \ \ \ \ \ .\ \ \ \ \ \ \ .\ \ \ \ \ \ \ .\ \ \ \ \ \ \ \!
.\ \ \ \ \ \ \ .\ \ \ \ \ \ \ .\ \ \ \ \ \ \ (spaces\ and\ ta\!
bs)}
\hbox{a\ \ \ \ \ \ \ b\ \ \ \ \ \ \ c\ \ \ \ \ \ \ d\ \ \ \ \ \ \ \!
e\ \ \ \ \ \ \ f\ \ \ \ \ \ \ g}
\hbox{\ a\ \ \ \ \ \ \ b\ \ \ \ \ \ \ c\ \ \ \ \ \ \ d\ \ \ \ \ \ \!
\ e\ \ \ \ \ \ \ f\ \ \ \ \ \ \ g}
\hbox{\ \ a\ \ \ \ \ \ \ b\ \ \ \ \ \ \ c\ \ \ \ \ \ \ d\ \ \ \ \ \!
\ \ e\ \ \ \ \ \ \ f\ \ \ \ \ \ \ g}
\hbox{\ \ \ a\ \ \ \ \ \ \ b\ \ \ \ \ \ \ c\ \ \ \ \ \ \ d\ \ \ \ \!
\ \ \ e\ \ \ \ \ \ \ f\ \ \ \ \ \ \ g}
\hbox{\ \ \ \ a\ \ \ \ \ \ \ b\ \ \ \ \ \ \ c\ \ \ \ \ \ \ d\ \ \ \!
\ \ \ \ e\ \ \ \ \ \ \ f\ \ \ \ \ \ \ g}
\hbox{\ \ \ \ \ a\ \ \ \ \ \ \ b\ \ \ \ \ \ \ c\ \ \ \ \ \ \ \!
d\ \ \ \ \ \ \ e\ \ \ \ \ \ \ f\ \ \ \ \ \ \ g}
\hbox{\ \ \ \ \ \ a\ \ \ \ \ \ \ b\ \ \ \ \ \ \ c\ \ \ \ \ \ \!
\ d\ \ \ \ \ \ \ e\ \ \ \ \ \ \ f\ \ \ \ \ \ \ g}
\hbox{.\ \ \ \ \ \ \ .\ \ \ \ \ \ \ .\ \ \ \ \ \ \ .\ \ \ \ \ \ \!
\ \ .\ \ \ \ \ \ \ .\ \ \ \ \ \ \ .\ \ \ \ \ \ \ (spaces)}
\hbox{.\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ .\ \ \ \ \ \ \ \ \ \ \ \ \ \!
.\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ .\ \ \ \ \ \ \ (multiple\ tabs\!
)}
\hbox{}
\hbox{}}
```

```
(* Generate TEX input file to typeset ASCII text (i.e. TEX macro source) *)
(* Written using Hedrick's TOPS-20 Native Mode Pascal compiler *)

Program NoFill(Input,Output);

Const
    MaxLen = 60;                (* Maximum length of output line *)
    LF = 012B;                  (* ASCII Line Feed *)

Type
    CharClass = (Printing, Quoted, Control, HTab, Return);

Var
    NextChar:   Char;
    CharType:   Array [0..177B] of CharClass;
    InCount:    Integer;
    OutCount:   Integer;
    I:          Integer;        (* Scratch counter *)


Procedure Initialize;           (* Initialize CharType array... *)

Begin (*Initialize*)

    (* Start of ASCII Control Characters: *)

    For I := 000B to 010B Do            (* NUL to BS *)
        CharType[I] := Control;

    CharType[011B] := HTab;             (* TAB *)

    CharType[012B] := Control;          (* LF *)
    CharType[013B] := Control;          (* VT *)
    CharType[014B] := Control;          (* FF *)

    CharType[015B] := Return;           (* CR *)

    For I := 016B to 037B Do            (* SO to US *)
        CharType[I] := Control;

    (* End of ASCII Control Characters *)

    CharType[040B] := Quoted;           (* Space *)
    CharType[041B] := Printing;         (* ! *)
    CharType[042B] := Printing;         (* " *)
    CharType[043B] := Quoted;           (* # *)
    CharType[044B] := Quoted;           (* $ *)
    CharType[045B] := Quoted;           (* % *)
    CharType[046B] := Printing;         (* & *)
    CharType[047B] := Quoted;           (* ' *)

    For I := 050B to 132B Do            (* ( to Z *)
        CharType[I] := Printing;
```

```
    CharType[133B] := Printing;          (* [ *)
    CharType[134B] := Quoted;            (* \ *)
    CharType[135B] := Printing;          (* ] *)
    CharType[136B] := Quoted;            (* ^ *)
    CharType[137B] := Quoted;            (* _ *)
    CharType[140B] := Quoted;            (* ` *)

    For I := 141B to 172B Do             (* a to z *)
        CharType[I] := Printing;

    CharType[173B] := Quoted;            (* { *)
    CharType[174B] := Quoted;            (* | *)
    CharType[175B] := Quoted;            (* } *)
    CharType[176B] := Quoted;            (* ~ *)
    CharType[177B] := Control;           (* DEL *)

End; (*Initialize*)


Procedure OutChar(InChar: Char);

Var
    Spaces:     Integer;        (* Spaces needed to expand a TAB *)
    NextChar:   Char;           (* Used for lookahead in CR-LF pairs *)

Begin (*Output*)

    InCount := InCount + 1;

    Case CharType[Ord(InChar)] of

        Printing:   Begin
                        Write(InChar);
                        OutCount := OutCount + 1;
                    End;

        Quoted:     Begin
                        Write('\',InChar);
                        OutCount := OutCount + 2;
                    End;

        Control:    Begin
                        Write('\up ');
                        OutCount := OutCount + 4;
                        InCount := InCount - 1;
                        If Ord(InChar) > 100B Then Begin (* DEL is Special *)
                            OutChar(Chr(Ord(InChar) - 100B));
                        End Else Begin
                            OutChar(Chr(Ord(InChar) + 100B));
                        End;
                    End;
```

```
        HTab:       Begin
                        Spaces := 8 - ((InCount - 1) MOD 8);
                        For I := 1 to Spaces Do Write(`\ `);
                        OutCount := OutCount + 2*Spaces;
                        InCount := Incount + Spaces - 1;
                    End;

        Return:     Begin
                        Read(NextChar);
                        If Ord(NextChar) = LF Then Begin
                            WriteLn(`}`);
                            Write(`\hbox{`);
                            OutCount := 6;      (* Length of \hbox{ *)
                            InCount := 0;
                        End Else Begin
                            Write(`\up M`);
                            OutCount := OutCount + 5;
                            OutChar(NextChar);
                        End;
                    End;

    End; (*Case*)

End; (*OutChar*)


Begin (*NoFill*)

    Initialize;                 (* Set up CharType array *)

    WriteLn(`{\ty`);

    WriteLn(`\def \#  {{\ty\char``043}}`);
    WriteLn(`\def \$  {{\ty\char``044}}`);
    WriteLn(`\def \%  {{\ty\char``045}}`);
    WriteLn(`\def \`` {{\ty\char``015}}`);
    WriteLn(`\def \\  {{\ty\char``134}}`);
    WriteLn(`\def \^  {{\ty\char``017}}`);
    WriteLn(`\def \_  {{\ty\char``032}}`);
    WriteLn(`\def \`  {{\ty\char``016}}`);
    WriteLn(`\def \{  {{\ty\char``173}}`);
    WriteLn(`\def \|  {{\ty\char``174}}`);
    WriteLn(`\def \}  {{\ty\char``176}}`);
    WriteLn(`\def \~  {{\ty\char``024}}`);

    WriteLn(`\def \up {{\ty\char``136}}`);

    Write(`\hbox{`);
    OutCount := 6;              (* Length of \hbox{ *)
    InCount := 0;

    While Not(Eof(Input)) Do
    Begin
```

```
        If OutCount > MaxLen Then Begin
            WriteLn('\!');
            OutCount := 0;
        End;

        Read(NextChar);
        OutChar(NextChar);

    End;

WriteLn('}}');

End.
```