```
/*************************************************************************
This ugly, sparsely-commented program is the source for text2pdf version
1.1.  It should be ANSI-conforming and compile on most platforms.  You'll
need to change LF_EXTRA to 1 for machines which write 2 characters for \n.
These include PCs, of course.

You may distribute the source or compiled versions free of charge.  You may
not alter the source in any way other than those mentioned above without
the permission of the author, Phil Smith <pns@cs.nott.ac.uk>.

Please send any comments to the author.

Copyright (c) Phil Smith, 1996

REVISION HISTORY

Version 1.1
11 Oct 96 Added handling of form-feed characters, removed need for tmp file,
          put reference to resources in each page (avoid bug in Acrobat),
          changed date format to PDF-1.1 standard.
12 Jun 96 Added check to avoid blank last page
12 Jun 96 Added LINE_END def to get round platform-specific \r, \n etc.
18 Mar 96 Added ISOLatin1Encoding option
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#ifndef SEEK_SET
#define SEEK_SET 0
#endif

#define LF_EXTRA 0  /* how many extra characters are written for \n */
                    /* change to 1 for PCs (where \n => <CR><LF>) */

#define LINE_END '\015'  /* CR used in xref table */
#define FF 12            /* formfeed character (^L) */

char *appname = "text2pdf v1.1";
char *progname = "text2pdf";

FILE *infile;
int pageNo = 0;
int pageObs[500];
int curObj = 5;  /* object number being or last written */
long locations[1000];

char font[256];
char *defaultFont = "Courier";
int ISOEnc = 0;
int doFFs = 1;
int tab = 8;
int pointSize = 10;
int vertSpace = 12;
int lines = 0;
int cols = 80;  /* max chars per output line */
int columns = 1;  /* number of columns */
```

```
/* Default paper is Letter size, as in distiller */
int pageHeight = 792;
int pageWidth = 612;

unsigned char buf[1024];
unsigned long fpos = 0;

void writestr(char *str) {
  /* Everything written to the PDF file goes through this function. */
  /* This means we can keep track of the file position without using */
  /* ftell on a real (tmp) file.  However, PCs write out 2 characters */
  /* for \n, so we need this ugly loop to keep fpos correct */

  fpos += strlen(str);
  while (*str) {
    if (*str == '\n') fpos += LF_EXTRA;
    putchar(*str++);
  }
}


void WriteHeader(char *title){

  struct tm *ltime;
  time_t clock;
  char datestring[30];

  time(&clock);
  ltime = localtime(&clock);

  strftime(datestring, 30, "D:%Y%m%d%H%M%S", ltime);

  writestr("%PDF-1.1\n");
  locations[1] = fpos;
  writestr("1 0 obj\n");
  writestr("<<\n");
  sprintf(buf, "/CreationDate (%s)\n", datestring); writestr(buf);
  sprintf(buf, "/Producer (%s (\\251 Phil Smith, 1996))\n", appname); writestr(buf);
  if (title) {sprintf(buf, "/Title (%s)\n", title); writestr(buf);}
  writestr(">>\n");
  writestr("endobj\n");

  locations[2] = fpos;
  writestr("2 0 obj\n");
  writestr("<<\n");
  writestr("/Type /Catalog\n");
  writestr("/Pages 3 0 R\n");
  writestr(">>\n");
  writestr("endobj\n");

  locations[4] = fpos;
  writestr("4 0 obj\n");
  writestr("<<\n");
  writestr("/Type /Font\n");
  writestr("/Subtype /Type1\n");
  writestr("/Name /F1\n");
  sprintf(buf, "/BaseFont %s\n", font); writestr(buf);
  if (ISOEnc) {
    writestr("/Encoding <<\n");
    writestr("/Differences [ 0 /.notdef /.notdef /.notdef /.notdef\n");
```

```
    writestr("/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef\n");
    writestr("/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef\n");
    writestr("/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef\n");
    writestr("/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef\n");
    writestr("/.notdef /.notdef /.notdef /.notdef /space /exclam\n");
    writestr("/quotedbl /numbersign /dollar /percent /ampersand\n");
    writestr("/quoteright /parenleft /parenright /asterisk /plus /comma\n");
    writestr("/hyphen /period /slash /zero /one /two /three /four /five\n");
    writestr("/six /seven /eight /nine /colon /semicolon /less /equal\n");
    writestr("/greater /question /at /A /B /C /D /E /F /G /H /I /J /K /L\n");
    writestr("/M /N /O /P /Q /R /S /T /U /V /W /X /Y /Z /bracketleft\n");
    writestr("/backslash /bracketright /asciicircum /underscore\n");
    writestr("/quoteleft /a /b /c /d /e /f /g /h /i /j /k /l /m /n /o /p\n");
    writestr("/q /r /s /t /u /v /w /x /y /z /braceleft /bar /braceright\n");
    writestr("/asciitilde /.notdef /.notdef /.notdef /.notdef /.notdef\n");
    writestr("/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef\n");
    writestr("/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef\n");
    writestr("/dotlessi /grave /acute /circumflex /tilde /macron /breve\n");
    writestr("/dotaccent /dieresis /.notdef /ring /cedilla /.notdef\n");
    writestr("/hungarumlaut /ogonek /caron /space /exclamdown /cent\n");
    writestr("/sterling /currency /yen /brokenbar /section /dieresis\n");
    writestr("/copyright /ordfeminine /guillemotleft /logicalnot /hyphen\n");
    writestr("/registered /macron /degree /plusminus /twosuperior\n");
    writestr("/threesuperior /acute /mu /paragraph /periodcentered\n");
    writestr("/cedilla /onesuperior /ordmasculine /guillemotright\n");
    writestr("/onequarter /onehalf /threequarters /questiondown /Agrave\n");
    writestr("/Aacute /Acircumflex /Atilde /Adieresis /Aring /AE\n");
    writestr("/Ccedilla /Egrave /Eacute /Ecircumflex /Edieresis /Igrave\n");
    writestr("/Iacute /Icircumflex /Idieresis /Eth /Ntilde /Ograve\n");
    writestr("/Oacute /Ocircumflex /Otilde /Odieresis /multiply /Oslash\n");
    writestr("/Ugrave /Uacute /Ucircumflex /Udieresis /Yacute /Thorn\n");
    writestr("/germandbls /agrave /aacute /acircumflex /atilde /adieresis\n");
    writestr("/aring /ae /ccedilla /egrave /eacute /ecircumflex\n");
    writestr("/edieresis /igrave /iacute /icircumflex /idieresis /eth\n");
    writestr("/ntilde /ograve /oacute /ocircumflex /otilde /odieresis\n");
    writestr("/divide /oslash /ugrave /uacute /ucircumflex /udieresis\n");
    writestr("/yacute /thorn /ydieresis ]\n");
    writestr(">>\n");
  }

  writestr(">>\n");
  writestr("endobj\n");

  locations[5] = fpos;
  writestr("5 0 obj\n");
  writestr("<<\n");
  writestr("  /Font << /F1 4 0 R >>\n");
  writestr("  /ProcSet [ /PDF /Text ]\n");
  writestr(">>\n");
  writestr("endobj\n");
}

long StartPage(){
  long strmPos;

  locations[++curObj] = fpos;
  pageObs[++pageNo] = curObj;
  sprintf(buf, "%d 0 obj\n", curObj); writestr(buf);
  writestr("<<\n");
  writestr("/Type /Page\n");
```

```
  writestr("/Parent 3 0 R\n");
  writestr("/Resources 5 0 R\n");
  sprintf(buf, "/Contents %d 0 R\n", ++curObj); writestr(buf);
  writestr(">>\n");
  writestr("endobj\n");

  locations[curObj] = fpos;
  sprintf(buf, "%d 0 obj\n", curObj); writestr(buf);
  writestr("<<\n");
  sprintf(buf, "/Length %d 0 R\n", curObj + 1); writestr(buf);
  writestr(">>\n");
  writestr("stream\n");
  strmPos = fpos;

  writestr("BT\n");
  sprintf(buf, "/F1 %d Tf\n", pointSize); writestr(buf);
  sprintf(buf, "1 0 0 1 50 %d Tm\n", pageHeight - 40); writestr(buf);
  sprintf(buf, "%d TL\n", vertSpace); writestr(buf);

  return strmPos;
}

void EndPage(long streamStart){
  long streamEnd;

  writestr("ET\n");
  streamEnd = fpos;
  writestr("endstream\n");
  writestr("endobj\n");

  locations[++curObj] = fpos;
  sprintf(buf, "%d 0 obj\n", curObj); writestr(buf);
  sprintf(buf, "%lu\n", streamEnd - streamStart); writestr(buf);
  writestr("endobj\n");
}

void WritePages(){
  int atEOF = 0;
  int atFF;
  int atBOP;
  long beginstream;
  int lineNo, charNo;
  int ch, column;
  int padding, i;

  while (!atEOF) {
    beginstream = StartPage();
    column = 1;
    while (column++ <= columns) {
      atFF = 0;
      atBOP = 0;
      lineNo = 0;
      while (lineNo++ < lines && !atEOF && !atFF) {
        writestr("(");
        charNo = 0;
        while (charNo++<cols &&
               (ch = getc(infile))!=EOF &&
               !(ch==FF && doFFs) &&
               ch!='\n') {
          if (ch >= 32 && ch <= 127) {
```

```
          if (ch == '(' || ch == ')' || ch == '\\') writestr("\\");
          sprintf(buf, "%c", (char)ch); writestr(buf);
        } else {
          if (ch == 9) {
            padding = tab - ((charNo - 1) % tab);
            for (i = 1; i <= padding; i++) writestr(" ");
            charNo += (padding - 1);
          } else {
            if (ch != FF) {
              /* write \xxx form for dodgy character */
              sprintf(buf, "\\%.3o", ch); writestr(buf);
            } else {
              /* don't print anything for a FF */
              charNo--;
            }
          }
        }
      }
      writestr(")'\n");

      /* messy stuff to handle formfeeds.  Yuk! */
      if (ch == EOF) atEOF = 1;
      if (ch == FF) atFF = 1;
      if (lineNo == lines) atBOP = 1;
      if (atBOP) {
        ch = getc(infile);
        if (ch == FF) ch = getc(infile);
        if (ch == EOF) atEOF = 1;
        else ungetc(ch, infile);
      }
      else if (atFF) {
        ch = getc(infile);
        if (ch == EOF) atEOF = 1;
        else ungetc(ch, infile);
      }
    }

    if (column <= columns) {
      sprintf(buf, "1 0 0 1 %d %d Tm\n",
              (pageWidth / 2) + 25, pageHeight - 40);
      writestr(buf);
    }
  }
  EndPage(beginstream);
 }
}

void WriteRest(){
  long xref;
  int i;

  locations[3] = fpos;
  writestr("3 0 obj\n");
  writestr("<<\n");
  writestr("/Type /Pages\n");
  sprintf(buf, "/Count %d\n", pageNo); writestr(buf);
  sprintf(buf, "/MediaBox [ 0 0 %d %d ]\n", pageWidth, pageHeight); writestr(buf);
  writestr("/Kids [ ");
  for (i = 1; i <= pageNo; i++) {sprintf(buf, "%d 0 R ", pageObs[i]); writestr(buf);}
  writestr("]\n");
```

```
  writestr(">>\n");
  writestr("endobj\n");

  xref = fpos;
  writestr("xref\n");
  sprintf(buf, "0 %d\n", curObj + 1); writestr(buf);
  /* note that \n is translated by writestr */
  sprintf(buf, "0000000000 65535 f %c", LINE_END); writestr(buf);
  for (i = 1; i <= curObj; i++) {
    sprintf(buf, "%.10ld 00000 n %c", locations[i], LINE_END);
    writestr(buf);
  }

  writestr("trailer\n");
  writestr("<<\n");
  sprintf(buf, "/Size %d\n", curObj + 1); writestr(buf);
  writestr("/Root 2 0 R\n");
  writestr("/Info 1 0 R\n");
  writestr(">>\n");

  writestr("startxref\n");
  sprintf(buf, "%ld\n", xref); writestr(buf);
  writestr("%%EOF\n");
}


void ShowHelp(){

  printf("\n%s [options] [filename]\n\n", progname);
  printf("  %s makes a 7-bit clean PDF file (version 1.1) from any input file.\n", progname);
  printf("  It reads from standard input or a named file, and writes the PDF file\n");
  printf("  to standard output.\n");
  printf("\n  There are various options as follows:\n\n");
  printf("  -h\t\tshow this message\n");
  printf("  -f<font>\tuse PostScript <font> (must be in standard 14, default: Courier)\n");
  printf("  -I\t\tuse ISOLatin1Encoding\n");
  printf("  -s<size>\tuse font at given pointsize (default %d)\n", pointSize);
  printf("  -v<dist>\tuse given line spacing (default %d points)\n", vertSpace);
  printf("  -l<lines>\tlines per page (default 60, determined automatically\n\t\tif unspecified)\n"
  printf("  -c<chars>\tmaximum characters per line (default 80)\n");
  printf("  -t<spaces>\tspaces per tab character (default 8)\n");
  printf("  -F\t\tignore formfeed characters (^L)\n");
  printf("  -A4\t\tuse A4 paper (default Letter)\n");
  printf("  -A3\t\tuse A3 paper (default Letter)\n");
  printf("  -x<width>\tindependent paper width in points\n");
  printf("  -y<height>\tindependent paper height in points\n");
  printf("  -2\t\tformat in 2 columns\n");
  printf("  -L\t\tlandscape mode\n");
  printf("\n  Note that where one variable is implied by two options, the second option\n  takes pr
  printf("  In landscape mode, page width and height are simply swapped over before\n  formatting,
  printf("\n%s (c) Phil Smith, 1996\n", appname);
}

int main(int argc, char **argv){
  int i = 1;
  int tmp, landscape = 0;
  char *ifilename = NULL;

  strcpy(font, "/");
  strcat(font, defaultFont);
```

```
infile = stdin;  /* default */

while (i < argc) {
  if (*argv[i] != '-') {  /* input filename */
    ifilename = argv[i];
    if (!(infile = fopen(ifilename, "r"))) {
      fprintf(stderr, "%s: couldn't open input file '%s'\n", progname, ifilename);
      exit(0);
    }
  } else {
    switch (*++argv[i]) {
    case 'h':
      ShowHelp();
      exit(0);
    case 'f':
      strcpy(font, "/");
      strcat(font, ++argv[i]);
      break;
    case 'I':
      ISOEnc = 1;
      break;
    case 'F':
      doFFs = 0;
      break;
    case 's':
      pointSize = atoi(++argv[i]);
      if (pointSize < 1) pointSize = 1;
      break;
    case 'v':
      vertSpace = atoi(++argv[i]);
      if (vertSpace < 1) vertSpace = 1;
      break;
    case 'l':
      lines = atoi(++argv[i]);
      if (lines < 1) lines = 1;
      break;
    case 'c':
      cols = atoi(++argv[i]);
      if (cols < 4) cols = 4;
      break;
    case '2':
      columns = 2;
      break;
    case 't':
      tab = atoi(++argv[i]);
      if (tab < 1) tab = 1;
      break;
    case 'A':
      switch (*++argv[i]) {
      case '3':
        pageWidth = 842;
        pageHeight = 1190;
        break;
      case '4':
        pageWidth = 595;
        pageHeight = 842;
        break;
      default:
        fprintf(stderr, "%s: ignoring unknown paper size: A%s\n", progname, argv[i]);
      }
```

```
      break;
    case 'x':
      pageWidth = atoi(++argv[i]);
      if (pageWidth < 72) pageWidth = 72;
      break;
    case 'y':
      pageHeight = atoi(++argv[i]);
      if (pageHeight < 72) pageHeight = 72;
      break;
    case 'L':
      landscape = 1;
      break;
    default:
      fprintf(stderr, "%s: ignoring invalid switch: -%s\n", progname, argv[i]);
    }
  }
  i++;
}

if (landscape) {
  tmp = pageHeight;
  pageHeight = pageWidth;
  pageWidth = tmp;
}

if (lines == 0) lines = (pageHeight - 72) / vertSpace;
if (lines < 1) lines = 1;
/* happens to give 60 as default */

WriteHeader(ifilename);
WritePages();
WriteRest();

return 0;
}
```