```
/* Block printer output
// (C) Copyright Software Design & Engineering, 3/1/95
// All rights reserved
// rrvt 3/1/95      */




#include "stdafx.h"
#include "blkdpo.h"
#include "printer.h"




Blkdpo::Blkdpo() {
  hdr = 1;
  }


void Blkdpo::init(int linesPerPage, int charsPerLine) {
int i;

  noLinesInPage = linesPerPage; noTxtLines = linesPerPage - 2;

  linesPerHalfPage = noTxtLines / 2; noCharsInLine = charsPerLine;

  for (i = 0; i < MaxLines; i++) lines[i].pline = (char*) malloc(noCharsInLine + 2);
  }


/* Set header with name, time with the width in the call  */

void Blkdpo::set_header(Cchar* pname, Cchar* ptime, int width) {
int   nlng   = strlen(pname);
int   tlng   = strlen(ptime);
int   spaces = (width - tlng) / 2 - nlng - LeftMargin;
int   i;

  header = pname;

  for (i = 0; i < spaces; i++) header += ' ';

 header += ptime;

  spaces = width - spaces - nlng - tlng - LeftMargin - 9;

  for (i = 0; i < spaces; i++) header += ' ';

  reset_blkdpo(2);
  }


void Blkdpo::disable_header(void) {reset_blkdpo(0);}
void Blkdpo::reset_blkdpo(int no_hdr_lines) {

  current_line = 0; pos = 0; hdr = no_hdr_lines; noBlanks = 0;

  noTxtLines = noLinesInPage - no_hdr_lines; linesPerHalfPage = noTxtLines / 2;

  pline = lines[current_line].pline; blank = true;
  }
```

```
/* Output one character to a blocked printer output */

void Blkdpo::put(char ch) {
int i;

  if (ch == '\f') {
    if (pline > lines[current_line].pline) {

      *pline = 0; lines[current_line].blank = blank;

      pline = lines[++current_line].pline; blank = true; pos = 0;
      }

    noBlanks = 0;

    while (printer.printPage(this)) ;

    return;
    }

  if (ch != '\n') {

    if (pos >= noCharsInLine && !noBlanks) terminateLine();

    if (ch == ' ') noBlanks++;

    else {
      if (pos + noBlanks >= noCharsInLine) terminateLine();

      else
        for (i = 0; i < noBlanks; i++) {

          if (pos >= noCharsInLine) terminateLine();

          *pline++ = ' '; pos++;
          }

      *pline++ = ch; pos++; noBlanks = 0;

      if (blank && ch > ' ') blank = false;
      }
    }

  else {
    if (!current_line && blank) {pline = lines[current_line].pline; pos = 0;}

    else terminateLine();

    noBlanks = 0;
    }
  }
```

```
// terminate line

void Blkdpo::terminateLine(void) {

  *pline = 0; lines[current_line].blank = blank;

  pline = lines[++current_line].pline; blank = true; pos = 0;

  if (current_line >= noLinesInPage) printer.printPage(this);
  }






/* Ouput lines  */

int Blkdpo::output_lines(int pageno) {
int    i;
int    j;
int    nolines;
Line   t;
String stg;
int    offset;

  if (!isPage()) return 0;

  offset = pageno < 10 ? 3 : pageno < 100 ? 2 : pageno < 1000 ? 1 : 0;

  if (hdr != 0) {
    stg.format("%*s%s%*spage %i", LeftMargin, "", header.str(), offset, "", pageno);
    printer.printLine(stg); printer.printLine("");
    }

  nolines = find_break(); //nonff = nolines > 0 && nolines < noTxtLines;

  for (i = 0; i < nolines; i++) {
    stg.format("%*s%s", LeftMargin, "", lines[i].pline); printer.printLine(stg);
    }

  for (j = 0; i < current_line; i++)
    if (j || !lines[i].blank)
      {t = lines[i]; lines[i] = lines[j]; lines[j] = t; j++;}

  current_line = j; pline = lines[current_line].pline; blank = true; pos = 0;

  return current_line;
  }
```

```
/* Find best break  */

int Blkdpo::find_break(void) {
int i;
int no_blank_lines =  0;
int single_index   =  0;
int double_index   =  0;

int max_double         = -1;
int second_page_base;
int second_page_double =  0;
int t;
int no_lines;

  if (current_line < noTxtLines) return current_line;

  for (i = 0; i < current_line; i++) {
    if (lines[i].blank) no_blank_lines++;

    else if (no_blank_lines) {
      if (no_blank_lines == 1) {
        sngl_blank_lines[single_index].line_index     = i - 1;
        sngl_blank_lines[single_index].no_blank_lines = no_blank_lines;
        single_index++;
        }
            else {
        dbl_blank_lines[double_index].line_index     = i - no_blank_lines;
        dbl_blank_lines[double_index].no_blank_lines = no_blank_lines;
        double_index++;
        }

      no_blank_lines = 0; if (i > 2*noLinesInPage) break;
      }
    }

  dbl_blank_lines[double_index].line_index     = current_line;
  dbl_blank_lines[double_index].no_blank_lines = 0;
  double_index++;
  for (i = 0; i < double_index; i++) {
    if (dbl_blank_lines[i].line_index > noTxtLines) break;
    max_double = i;
    }

  if (max_double >= 0) {
    no_lines       = dbl_blank_lines[max_double].line_index;
    no_blank_lines = dbl_blank_lines[max_double].no_blank_lines;
    if (no_lines > linesPerHalfPage) return no_lines;

    second_page_base = no_lines + no_blank_lines;

    for (i = max_double + 1; i < double_index; i++) {
      t = dbl_blank_lines[i].line_index - second_page_base;
      if (t > noTxtLines) break;
      if (dbl_blank_lines[i].line_index > noTxtLines) return no_lines;
      second_page_double = t;
      }
    }

  no_lines = 0;

  for (i = 0; i < single_index; i++) {
    if (sngl_blank_lines[i].line_index > noTxtLines) break;
    no_lines = sngl_blank_lines[i].line_index;
    }

  if (no_lines > linesPerHalfPage) return no_lines;

  if (second_page_double > linesPerHalfPage)
        return dbl_blank_lines[max_double].line_index;

  return noTxtLines;
  }



/* close output, flushing buffer and terminating page */

void Blkdpo::close(void) {
char* p;

  while (printer.printPage(this)) ;

  if (printer.doubleSided && (printer.pageno & 1) == 0) {
    p = "\n\n\n                                     This page is blank.\n";     while (*p) put(*p++);
    printer.printPage(this);
    }
  }
```