```
/* 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"



Ln* Lines::resetLines() {
int j;

  for (; i < curX; i++) if (!lines[i].blank) break;

  for (j = 0; i < curX; i++, j++) lines[j] = lines[i];

  curX = j;  ln = resetCur();  return ln;
  }


Blkdpo::Blkdpo() : noCharsInLine(0), noLinesInPage(0), pline(0), noTxtLines(0),
                   blank(false), outputHdr(true) { }


void Blkdpo::init(int linesPerPage, int charsPerLine) {
  noCharsInLine = charsPerLine;
  noLinesInPage = linesPerPage;
  lines.init();
  pline         = 0;
  header.clear();
  outputHdr     = true;
  blank         = false;
  noBlanks      = 0;
  reset_blkdpo(0);
  }
```

```
/* Set header with name, time */

void Blkdpo::set_header(TCchar* pname, TCchar* ptime) {          //, int width
int   nlng   = _tcsclen(pname);
int   tlng   = _tcsclen(ptime);
int   width  = printer.noCharsPerLine();
int   spaces = (width - tlng) / 2 - nlng - LeftMargin;
int   i;

  header = pname;

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

  header += ptime;

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

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

  reset_blkdpo(true);
  }


void Blkdpo::disable_header(void) {reset_blkdpo(true);}



void Blkdpo::reset_blkdpo(bool hdr) {

  lines.resetCur();                           // current_line = 0;
  outputHdr = hdr; noBlanks = 0;

  noTxtLines = noLinesInPage - 2;

  ln = lines.cur(); pline = &ln->s;            //  pline = getNewLine();
  }
```

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

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

  if (ch == _T('\f')) {
    if (pline > lines.curLine()) {

      ln->blank = blank;                        //lines[current_line].blank = blank;

      ln = lines.nextCur(); pline = &ln->s;     //current_line++; pline = getNewLine();
      }

    noBlanks = 0;

    while (printer.printPage(this)) continue;   return;
    }

  if (ch == _T('\n')) {terminateLine();   noBlanks = 0;  return;}

  if (lines.curPos() >= noCharsInLine && !noBlanks) terminateLine();

  if (ch == _T(' ')) {noBlanks++; return;}

  if (lines.curPos() + noBlanks >= noCharsInLine) terminateLine();

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

      if (lines.curPos() >= noCharsInLine) terminateLine();

      lines.put(_T(' '));
      }


  lines.put(ch); noBlanks = 0;                                    // *pline += ch; pos++;

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



// terminate line

void Blkdpo::terminateLine(void) {

  ln->blank = blank;              // lines[current_line].blank = blank;

  ln = lines.nextCur();  pline = &ln->s;   blank = true;      // current_line++; pline = getNewLine();

  if (lines.gotApage(noLinesInPage)) printer.printPage(this);
  }
```

```
/* Ouput lines  */

bool Blkdpo::output_lines(int pageno) {
int    i;
int    nolines;
Ln*    p;
String stg;
int    offset;

  if (!lines.isPage()) return false;

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

  if (outputHdr) {
    stg.format(_T("%*s%s%*spage %i"), LeftMargin, _T(""), header.str(), offset, _T(""), pageno);
    printer.printLine(stg); printer.printLine(_T(""));
    }

   Break brk(lines);  nolines = brk.find(noTxtLines, noLinesInPage);

  for (i = 0, p = lines.startLoop(); i < nolines && p; i++, p = lines.nextNode()) {
    stg.format(_T("%*s%s"), LeftMargin, _T(""), p->s.str()); printer.printLine(stg);
    }

  ln = lines.resetLines();  pline = &ln->s;   return true;
  }




struct BlankLn {
int line_index;
int no_blank_lines;

  BlankLn() : line_index(0), no_blank_lines(0) { }
  };


class BlankLines {
int     nBlkLns;
BlankLn blkLns[MaxLines];

public:

  BlankLines() : nBlkLns(0) {}

  void     add(int lnX, int nBlnks);
  BlankLn* findMax(int maxLn);
  bool     isSecondBrk(BlankLn& firstBrkLn, int noTxtLines);
  };
```

```
/* Find best break  */

int Break::find(int noTxtLines, int noLinesInPage) {
int        linesPerHalfPage = noTxtLines / 2;
int        i;
int        no_blank_lines =  0;
int        firstBrk;
BlankLines sngl;
BlankLines dbl;

  if (curX < noTxtLines) return curX;

  for (i = 0; i < curX; i++) {

    if (lns.lines[i].blank) no_blank_lines++;

    else if (no_blank_lines) {
      if (no_blank_lines == 1) sngl.add(i, no_blank_lines);
      else                     dbl.add(i, no_blank_lines);

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

  BlankLn* maxDbl = dbl.findMax(noTxtLines);

  if (maxDbl) {
    firstBrk       = maxDbl->line_index;

    if (firstBrk > linesPerHalfPage) return firstBrk;

    if (dbl.isSecondBrk(*maxDbl, noTxtLines)) return firstBrk;

    if (sngl.isSecondBrk(*maxDbl, noTxtLines)) return firstBrk;
    }

  firstBrk = 0;

  BlankLn* maxSngl = sngl.findMax(noTxtLines);

  if (maxSngl) {
    firstBrk = maxSngl->line_index;

    if (firstBrk > linesPerHalfPage) return firstBrk;

    if (dbl.isSecondBrk(*maxSngl, noTxtLines))  return firstBrk;

    if (sngl.isSecondBrk(*maxSngl, noTxtLines)) return firstBrk;
    }

  return noTxtLines;
  }
```

```
// Add a break to the array

void BlankLines::add(int lnX, int nBlnks) {
  blkLns[nBlkLns].line_index     = lnX - nBlnks;
  blkLns[nBlkLns].no_blank_lines = nBlnks;
  nBlkLns++;
  }


// Find the second break maximum break that starts at firstBrk and has fewer than noTxtLines

bool BlankLines::isSecondBrk(BlankLn& firstBrkLn, int noTxtLines) {
int      firstBrk         = firstBrkLn.line_index;
int      no_blank_lines   = firstBrkLn.no_blank_lines;
int      second_page_base = firstBrk         + no_blank_lines;
int      second_page_end  = second_page_base + noTxtLines;
BlankLn* maxScnd          = findMax(second_page_end);
int      secondBrk        = maxScnd ? maxScnd->line_index : 0;

  return secondBrk > firstBrk && secondBrk - second_page_base > noTxtLines/2;
  }


// find maximum break that is less than maxLn (maximum line)

BlankLn* BlankLines::findMax(int maxLn) {
int i;

  if (nBlkLns == 0) return 0;

  for (i = 0; i < nBlkLns; i++) if (blkLns[i].line_index > maxLn) break;

  return i > 1 ? &blkLns[i-1] : 0;
  }



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

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

  while (printer.printPage(this)) continue;

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

```
  #if 0
void Blkdpo::isMagicLine() {
String& s = lines.cur()->s;

  if (s.find(_T("Blkdpo::terminateLine")) >= 0) {
    int x = blank;
    }

  if (ln && ln->s != s) {
    int y = 1;
    }
  }
#endif
```

```
//345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
//2
//3
//4
//5
//6
//7
//8
//9
//0
//1
//2
//3
//4
//5
//6
//7
//8
//9
//0
//1
//2
//3
//4
//5
//6
//7
//8
//9
//0
//1
//2
//3
//4
//5
//6
//7
//8
//9
//0
//1
//2
//3
//4
//5
//6
//7
//8
//9
//0
//1
//2
//3
//4
//5
//6
//7
//8
//9
//0
```

```
//1
//2
//3
//4
//5
//6
//7
//8
//9
//0
//1
//2
//3
//4
//5
//6
//7
//8
//9
//0
//1
//2
//3
//4
//5
//6
//7
//8
//9
//0
//1
//2
```