```
cbf_failnez (cbf_count_datablocks(cif, &blocks))
for (blocknum = 0; blocknum < blocks;  blocknum++)
{ /* start of copy loop */
  cbf_failnez (
    cbf_select_datablock(cif, blocknum))
  cbf_failnez (
    cbf_datablock_name(cif, &datablock_name))
  cbf_failnez (
    cbf_force_new_datablock(cbf, datablock_name))
  if ( !cbf_rewind_category(cif) ) {
    cbf_failnez (
      cbf_count_categories(cif, &categories))
    for (catnum = 0; catnum < categories; catnum++) {
      cbf_select_category(cif, catnum);
      cbf_category_name(cif,&category_name);
      cbf_force_new_category(cbf, category_name);
      cbf_count_rows(cif,&rows);
      cbf_count_columns(cif,&columns);
      /*  Transfer the column names from cif to cbf */
      if ( ! cbf_rewind_column(cif) ) {
      do {
        cbf_failnez(
          cbf_column_name(cif, &column_name))
        cbf_failnez(
          cbf_new_column(cbf, column_name))
      } while ( ! cbf_next_column(cif) );
      cbf_rewind_column(cif);
      cbf_rewind_row(cif);
      }
      /* Transfer the rows from cif to cbf */
      for (rownum = 0; rownum < rows; rownum++) {
        cbf_failnez (
          cbf_select_row(cif, rownum))
        cbf_failnez ( cbf_new_row(cbf))
        cbf_rewind_column(cif);
        for (colnum = 0; colnum < columns; colnum++) {
          cbf_failnez (cbf_select_column(cif, colnum))
          if ( ! cbf_get_value(cif, &value) ) {
            cbf_failnez (
              cbf_select_column(cbf, colnum))
            cbf_failnez ( cbf_set_value(cbf, value))
          } else {
            void * array;
            int binary_id, elsigned, elunsigned;
            size_t elements,elements_read, elsize;
            int minelement, maxelement;
            unsigned int cifcompression;

            cbf_failnez (
              cbf_get_integerarrayparameters(cif,
              &cifcompression, &binary_id, &elsize,
              &elsigned, &elunsigned, &elements,
              &minelement, &maxelement))
            if (array=malloc(elsize*elements)) {
              cbf_failnez (
                cbf_select_column(cbf,colnum))
              cbf_failnez (
                cbf_get_integerarray(
                cif, &binary_id, array, elsize,
                elsigned, elements, &elements_read))
              cbf_failnez(
                cbf_set_integerarray(
                cbf, compression, binary_id, array,
                elsize, elsigned, elements))
              free(array);
            } else {
              fprintf(stderr,
              "\nFailed to allocate memory %d bytes",
                elsize*elements);
               exit(1);
            }
          }
        }
      }
    }
  }
```

Fig. 5.6.5.3. Listing of the main code fragment in the program *cif2cbf* for conversion of an ASCII CIF to a CBF file.

Borenstein, 1996) headers within binary data value text fields. The default is headers for CIFs, no headers for CBFs. *-d* specifies the use of digests. The default is the MD5 digest (Rivest, 1992) when MIME headers are selected. *-e* specifies one of the standard MIME encodings: BASE64 (the default) or quoted-printable; or a non-standard decimal, hexadecimal or octal encoding for an ASCII CIF; or 'none' for a binary CBF. *-b* specifies the direction of mapping of bytes into words for decimal, hexadecimal or octal output, marked by '>' for forwards or '<' for backwards as the second character of each line of output, and in # comment lines. The default is backwards.

5.6.5.3. *cif2cbf*

The test program *cif2cbf* (Fig. 5.6.5.3) uses the same command-line options as *img2cif*, but accepts either a CIF or a CBF as input instead of an image file. The heart of the code is a series of nested loops. The outermost loop scans through all the data blocks. The next innermost loop scans through all the categories within each data block. The first of the two next innermost loops scans through the columns to copy the column headings. Then the second of these two loops scans through the rows. Finally, the innermost loop scans through the columns for each row.

We are grateful to Frances C. Bernstein for her helpful comments and suggestions.

References

ADSC (1997). *Quantum 4R specifications.* Poway, CA, USA: Area Detector Systems Corporation. http://www.adsc-xray.com/Q4techspecs.html.

Freed, N. & Borenstein, N. (1996). *Multipurpose Internet Mail Extensions (MIME) part one: format of internet message bodies.* RFC 2045. Internet Engineering Task Force. http://www.ietf.org/rfc/rfc2045.txt.

MAR Research (1997). *The mar345 image plate detector.* Norderstedt: MAR Research GmbH. http://www.marresearch.com/products.mar345.html.

Moffat, A., Bell, T. C. & Witten, I. H. (1997). *Lossless compression for text and images. Int. J. High Speed Electron. Syst.* **8**, 179–231.

Rivest, R. (1992). *The MD5 message-digest algorithm.* RFC 1321. Internet Engineering Task Force. http://www.ietf.org/rfc/rfc1321.txt.

Tosic, O. & Westbrook, J. D. (2000). *CIFParse. A library of access tools for mmCIF*. Reference guide. http://sw-tools.pdb.org/apps/CIFPARSE-OBJ/cifparse/index.html.