```
                         PROGRAM  CIF_IN
C
C....A.. Define the data variables
         include        'ciftbx.cmn'
         logical        f1,f2,f3
         character*32   name
         character*80   line
         real           cela,celb,celc,siga,sigb,sigc
         real           x,y,z,u,numb,sdev
         data           cela,celb,celc,siga,sigb,sigc /6*0.0/

C....B.. Assign the CIFtbx files
         f1 = init_( 1, 2, 3, 6 )

C....C.. Request dictionary validation check
         if(dict_('cif_core.dic','valid'))  goto 100
         write(6,'(/a/)') ' Requested Core dictionary not present'

C....D.. Open the CIF to be accessed
100      name='test.cif'
         if(ocif_(name))      goto 120
         write(6,'(a///)')  ' >>>>>>>>> CIF cannot be opened'
         stop
C....E.. Assign the data block to be accessed
120      if(.not.data_(' '))       goto 200
         write(6,'(/a,a/)') ' Access items in data block  ',bloc_

C....F.. Extract some cell dimensions; test all is OK
         f1 = numb_('_cell_length_a', cela, siga)
         f2 = numb_('_cell_length_b', celb, sigb)
         f3 = numb_('_cell_length_c', celc, sigc)
         if(.not.(f1.and.f2.and.f3)) write(6,'(a)') ' Cell lengths missing!'
         write(6,'(a,6f10.4)') ' Cell ',cela,celb,celc,siga,sigb,sigc

C....G.. Extract space group notation (expected char string)
         f1 = char_('_symmetry_cell_setting', name)
         write(6,'(a,a/)') ' Cell setting   ',name(1:long_)

C....H.. Get the next name in the CIF and print it out
         f1 = name_(name)
         write(6,'(a,a/)') ' Next data name in CIF is   ',name

C....I.. List the audit record (possible text line sequence)
         write(6,'(a)') ' Audit record'
140      f1 = char_('_audit_update_record', line)
         write(6,'(a)') line
         if(text_)  goto 140

C....J.. Extract atom site data in a loop
         write(6,'(/a)') ' Atom sites'
160      f1 = char_('_atom_site_label', name)
         f2 = numb_('_atom_site_fract_x', x, sx)
         f2 = numb_('_atom_site_fract_y', y, sy)
         f2 = numb_('_atom_site_fract_z', z, sz)
         f3 = numb_('_atom_site_U_iso_or_equiv', u, su)
         write(6,'(1x,a4,4f8.4)') name,x,y,z,u
         if(loop_)  goto 160
C
         goto 120
200      continue
         end
```

Fig. 5.4.9.1. Sample program *CIF_IN*. See text for explanation.

`'#< not in dictionary'`. This applies to both looped and single data items.

A more complex example of writing a CIF is given in the program *cif2cif* available with the *CIFtbx* release. A similar program that reads a CIF and writes an XML file is *cif2xml*, also available with the *CIFtbx* release.

5.4.11. Error-message glossary

The *CIFtbx* routines will generate explicit error messages in the printout or in the created CIF if requested to do so (*e.g.* during dictionary checks). If data processing cannot continue (*i.e.* fatal errors), an appropriate error message is placed in the printout and execution terminates. However, the default approach is to remain mute and for error detection to be monitored by the application program *via* the *CIFtbx* functions returning `.true.` or `.false.` values that tell the application program whether the command was performed correctly. This places the primary responsibility for error checking on the application software. The importance of this approach is that it enables the local application to respond to run-time problems in a controlled way and to take corrective action if it is possible. However, some types of processing errors, such as exceeding the dimensions of critical *CIFtbx* arrays, do require appropriate messages to be issued and for execution to cease.