```
// seek.cpp - RECOVER A FILE FROM THE UNUSED STORAGE ON A HARD DISK
//
// MAINTENANCE HISTORY
// DATE              PROGRAMMER AND DETAILS
// 18-09-14   JS     Original
// 23-09-14   MPF    Removed unused variables
// 13-10-14   MPF    Corrected usage description
// 13-10-14   MPF    Check if all pages are missing
//
//--------------------------------------------------------------------------

#include <cstring>          // C-style string manipulation functions
#include <cstdlib>          // C-style standard library
#include <map>                      // C++ map declarations
#include <string>           // C++ string declarations
using namespace std;        // Expand the standard namespace
#include <stdint.h>         // Standard integer type declarations
#include "hideAndSeek.h"    // Hide and seek declarations

//--------------------------------------------------------------------------

// MAIN LINE

int
main (
       int           argc,         // Argument count
       char          *argv[])      // Argument value pointers
{
       const char    *key;         // Encryption and signature key
       FILE          *diskFp;      // Whole disk file pointer
       PlainPage_t   plainPage;    // Plaintext page
       CipherPage_t  cipherPage;   // Encrypted page
       unsigned char theoSig[20];  // Theoretical signature of the page
       map<unsigned long,string> plainMap; // Plaintext map
       map<unsigned long,string>::iterator plainIter; // Plaintext iterator
       size_t        writeCnt;     // Number of pages written
       unsigned long expPageNo;    // Expected page number

       // Decode arguments

       if (argc != 2) {
              fprintf (stderr, "Error: usage: seek password > output\n");
              exit (1);
       }
       key = argv[1];

       // Read each page on the disk

       diskFp = fopen ("wholedisk", "r");
       while (fread (&cipherPage, sizeof(cipherPage), 1, diskFp) != 0) {

              // If the actual signature is the same as the theoretical
              // signature of the data, process the page

              GetSignature (key, cipherPage.cipherText, theoSig);
              if (memcmp(cipherPage.signature,theoSig,sizeof(theoSig)) == 0) {

                     // Decrypt the page
```

```
                Decrypt (key, cipherPage.cipherText, &plainPage);

                // Save the page in the plaintext map

                plainMap[plainPage.pageNumber].assign (
                        plainPage.plainData, plainPage.plainSize);
        }
    }
    fclose (diskFp);

    // Emit the pages

    expPageNo = 0;
    for (
        plainIter = plainMap.begin();
        plainIter != plainMap.end();
        plainIter ++
    ) {
        while (expPageNo < plainIter->first) {
            fprintf (stderr, "Warning: missing page %lu\n",
                    expPageNo);
            expPageNo ++ ;
        }
        writeCnt = fwrite (plainIter->second.data(),
            plainIter->second.length(), 1, stdout);
        if (writeCnt != 1) {
            fprintf (stderr, "Error: cannot write page\n");
            exit (1);
        }
        expPageNo ++ ;
    }
    if (expPageNo == 0) {
        fprintf (stderr, "Error: all pages missing\n");
        exit (1);
    }

    return 0;
}
```