```
// Anderbase.cpp - EXTRACT THE DATABASE FROM AN ANDERBASE BACKUP FILE
//
// MODULE INDEX
// NAME                         CONTENTS
// SQL                          Placeholder sql function
// Error                        Process an error
// Source_c::ReadContents       Read source contents
// InSource_c::Read             Read standard input
// TagSource_c::LoadEnvelope    Load an envelope
// TagSource_c::Open            Open a tag-length-value chain
// TagSource_c::Read            Read data from a tag-length-value chain
// ToDbForm                     Convert a string to the format required by the
//                              database
// main                         Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 03-10-15     JS      Original
//
//-----------------------------------------------------------------------------

#include <cstring>              // C-style string manipulation functions
#include <cstdlib>              // C-style standard library
#include <string>               // C++ string declarations
#include <iostream>             // C++ I/O stream declarations
#include <sstream>              // C++ string stream declarations
#include <vector>               // C++ vector declarations
using namespace std;            // Expand the standard namespace

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

// DEFINITIONS

typedef unsigned char Tag_t;            // Tag type
const size_t    MAX_VAL_LEN = 1024;     // Maximum value length

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

// DATA SOURCE BASE-CLASS

class Source_c {
public:
        virtual size_t Read (void *buf, size_t cap) = 0;
                                        // Read data from the source
        string ReadContents ();
                                        // Read source contents
};

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

// STANDARD INPUT SOURCE

class InSource_c : public Source_c {
public:
        size_t Read (void *buf, size_t cap);
                                        // Read data from the source
};

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

// TAG-LENGTH-VALUE SOURCE

class TagSource_c : public Source_c {
        Source_c        *tagSource;     // Lower level source
```

```
        Tag_t           tagNo;          // Tag number
        char            tagBuf[MAX_VAL_LEN]; // Data buffer
        size_t          tagLen;         // Length in the buffer
        size_t          tagOfs;         // Current offset in the buffer

        bool LoadEnvelope (Tag_t *tag);
                                        // Load an envelope from the source
public:
        bool Open (Source_c *lowerSource, Tag_t *tag);
                                        // Open the tag-length-value chain
        size_t Read (void *buf, size_t cap);
                                        // Read data from the source
};

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

// COLUMN TYPE ENUMERATOR

enum ColType_t {
        COL_TYPE_INTEGER,               // Integer
        COL_TYPE_CHAR                   // Character
};

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

// COLUMN VECTOR STRUCTURE

struct Col_t {
        string          colName;        // Column name
        Tag_t           colTag;         // Column tag
        ColType_t       colType;        // Column type
        size_t          colSize;        // Column size
};
typedef vector<Col_t> ColVec_t;

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

// COLUMN DATA VECTOR STRUCTURE

struct Data_t {
        size_t          dataColNo;      // Column number
        long            dataIntVal;     // Integer value
        string          dataCharVal;    // Character value
};
typedef vector<Data_t> DataVec_t;

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

// PLACEHOLDER SQL FUNCTION

void
SQL (
        const char      *stmt)          // SQL statement
{
#       if 0
                string          db2Stmt;        // DB2 statement
                const char      *p;             // General purpose pointer

                db2Stmt = "db2 connect to sultinex;";
                db2Stmt += "db2 \"";
                for (p = stmt; *p != '\0'; p++) {
                        if (*p == '"')
                                db2Stmt += "\\\"";
                        else
                                db2Stmt += *p;
```

```
                }
                db2Stmt += "\";";
                system (db2Stmt.c_str());
#       else
                cout << stmt << '\n';
#       endif
}

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

// PROCESS AN ERROR

void
Error (
        const char      *message)       // Error message
{
        cerr << "Error: " << message << '\n';
        exit (1);
}

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

// READ SOURCE CONTENTS

string
Source_c::ReadContents ()
{
        string          str;            // String variable
        char            rdBuf[MAX_VAL_LEN]; // Read buffer
        size_t          rdLen;          // Read length

        while ((rdLen = Read (rdBuf, MAX_VAL_LEN)) != 0)
                str.append (rdBuf, rdLen);
        return str;
}

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

// READ STANDARD INPUT

size_t
InSource_c::Read (
        void            *buf,           // Input buffer
        size_t          cap)            // Buffer capacity
{
        cin.read (static_cast<char*>(buf), cap);
        return cin.gcount();
}

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

// LOAD AN ENVELOPE

bool
TagSource_c::LoadEnvelope (
        Tag_t           *tag)           // Returned tag number
{
        size_t          rdLen;          // Read length
        size_t          i;              // General purpose index
        unsigned char   lenBuf[2];      // Length buffer

        if (tagSource->Read (tag, sizeof(*tag)) != sizeof(*tag)) return 0;
        if (tagSource->Read (lenBuf, 2) != 2) Error ("missing length");
        tagLen = static_cast<size_t>(lenBuf[0]) << 8 | lenBuf[1];
        if (tagLen > MAX_VAL_LEN) Error ("invalid length");
```

```
        rdLen = tagSource->Read (tagBuf, tagLen);
        if (rdLen != tagLen) Error ("truncated value");
        tagOfs = 0;
        return 1;
}

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

// OPEN A TAG-LENGTH-VALUE CHAIN

bool
TagSource_c::Open (
        Source_c        *lowerSource,   // Lower-level source
        Tag_t           *tag)           // Returned tag number
{
        // Load the first envelope

        tagSource = lowerSource;
        if ( ! LoadEnvelope (&tagNo)) return 0;
        *tag = tagNo;
        return 1;
}

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

// READ DATA FROM A TAG-LENGTH-VALUE CHAIN

size_t
TagSource_c::Read (
        void            *buf,           // Destination buffer
        size_t          cap)            // Buffer capacity
{
        size_t          len;            // Length read
        size_t          copyLen;        // Copy length
        Tag_t           rdTag;          // Read tag

        len = 0;
        while (len < cap && (tagOfs < tagLen || tagLen == MAX_VAL_LEN)) {
                if (tagOfs == tagLen) {
                        if ( ! LoadEnvelope (&rdTag) || rdTag != tagNo)
                                Error ("tag mismatch");
                }
                copyLen = cap - len;
                if (copyLen > tagLen-tagOfs) copyLen = tagLen - tagOfs;
                memcpy (static_cast<char*>(buf)+len, tagBuf+tagOfs, copyLen);
                len += copyLen;
                tagOfs += copyLen;
        }
        return len;
}

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

// CONVERT A STRING TO THE FORMAT REQUIRED BY THE DATABASE

string
ToDbForm (
        const string    &s)             // String to convert
{
        size_t          i;              // General purpose index
        string          dbForm;         // String in database form

        dbForm = '\'';
        for (i = 0; i < s.length(); i++) {
                if (s[i] < 32 || s[i] > 126)
```

```
                        Error ("illegal character in character data");
                dbForm += s[i];
                if (s[i] == '\'') dbForm += '\'';
        }
        dbForm += '\'';
        return dbForm;
}

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

// MAIN LINE

int
main ()
{
        InSource_c      inSource;       // Standard input source
        TagSource_c     tableSource;    // Table data source
        TagSource_c     secSource;      // Section data source
        TagSource_c     colSource;      // Column parameters source
        TagSource_c     fldSource;      // Field data source
        Tag_t           tag;            // Tag number
        string          tableName;      // Table name
        Col_t           col;            // Column data
        ColVec_t        colVec;         // Column vector
        string          content;        // Source contents
        ostringstream   sqlOss;         // SQL output string stream
        string          sqlStmt;        // SQL statement string
        size_t          i, j;           // General purpose indices
        TagSource_c     rowSource;      // Table row data source
        Data_t          data;           // Column data structure
        DataVec_t       dataVec;        // Column data vector
        TagSource_c     dataSource;     // Column data source
        long            intVal;         // Integer value

        // Process each table in the source

        while (tableSource.Open (&inSource, &tag)) {
                if (tag != 0x01) Error ("wrong table tag");

                // Read the table schema entry

                if ( ! secSource.Open (&tableSource, &tag) || tag != 0x02)
                        Error ("wrong schema tag");

                // Read the table name

                if ( ! fldSource.Open (&secSource, &tag) || tag != 0x03)
                        Error ("no table name");
                tableName = fldSource.ReadContents ();

                // Read each column declaration

                colVec.clear ();
                while (colSource.Open (&secSource, &tag)) {
                        if (tag != 0x04) Error ("wrong column tag");

                        // Read the column name

                        if ( ! fldSource.Open (&colSource, &tag) || tag != 0x05)
                                Error ("no column name");
                        col.colName = fldSource.ReadContents ();

                        // Read the column tag number

                        if ( ! fldSource.Open (&colSource, &tag) || tag != 0x06)
```

```
                Error ("no column tag");
        content = fldSource.ReadContents ();
        if (content.length() != 1)
                Error ("invalid tag size");
        col.colTag = content[0];

        // Read the column type tag

        if ( ! fldSource.Open (&colSource, &tag))
                Error ("no type tag");

        // Read integer column parameters

        if (tag == 0x07) {
                col.colType = COL_TYPE_INTEGER;
                content = fldSource.ReadContents ();
                if (content.length() != 1)
                        Error ("invalid integer size");
                col.colSize = content[0];
                if (col.colSize != 2 && col.colSize != 4)
                        Error ("invalid integer size");
        }

        // Read character column parameters

        else if (tag == 0x08) {
                col.colType = COL_TYPE_CHAR;
                content = fldSource.ReadContents ();
                if (content.length() != 2)
                        Error ("invalid integer size");
                col.colSize = content[0] & 0xff;
                col.colSize <<= 8;
                col.colSize |= content[1] & 0xff;
                if (col.colSize < 1)
                        Error ("invalid char column size");
        }

        // Other column types are unsupported

        else
                Error ("unsupported column type");

        // Append the column to the table

        colVec.push_back (col);
}

// Create the table on the destination database

sqlOss.str ("");
sqlOss << "create table " << tableName << " (";
for (i = 0; i < colVec.size(); i++) {
        if (i != 0) sqlOss << ", ";
        sqlOss << colVec[i].colName;
        switch (colVec[i].colType) {
        case COL_TYPE_INTEGER:
                if (colVec[i].colSize == 2)
                        sqlOss << " smallint";
                else if (colVec[i].colSize == 4)
                        sqlOss << " integer";
                else
                        Error ("unsupported integer size");
                break;
        case COL_TYPE_CHAR:
                sqlOss << " char(" << colVec[i].colSize << ')';
```

```
                break;
        default:
                Error ("unsupported column type");
        }
}
sqlOss << ')';
sqlStmt = sqlOss.str();
SQL (sqlStmt.c_str());

// Read the contents of the table

if ( ! secSource.Open (&tableSource, &tag) || tag != 0x09)
        Error ("wrong contents tag");

// Read each row in the table

while (rowSource.Open (&secSource, &tag)) {
        if (tag != 0x0a) Error ("wrong row tag");

        // Read each column value in the table

        dataVec.clear ();
        while (dataSource.Open (&rowSource, &tag)) {

                // Look up the column

                i = 0;
                while (
                        i < colVec.size() &&
                        colVec[i].colTag != tag
                ) i ++ ;
                if (i >= colVec.size())
                        Error ("unrecognised column tag");
                data.dataColNo = i;

                // Check for duplicate column tags

                j = 0;
                while (
                        j < dataVec.size() &&
                        dataVec[j].dataColNo != i
                ) j ++ ;
                if (j < dataVec.size())
                        Error ("duplicate column data");

                // Load the column data

                switch (colVec[i].colType) {
                case COL_TYPE_INTEGER:
                        content = dataSource.ReadContents();
                        if ((content[0] & 0x80) == 0)
                                intVal = 0;
                        else
                                intVal = -1;
                        for (j = 0; j < content.length(); j++)
                                intVal = (intVal << 8 & ~0xffL)
                                        | (content[j] & 0xff);
                        data.dataIntVal = intVal;
                        break;
                case COL_TYPE_CHAR:
                        data.dataCharVal =
                                dataSource.ReadContents();
                        break;
                default:
                        Error ("unsupported column type");
```

```
                        }

                        // Append the column data to the data vector

                        dataVec.push_back (data);
                }

                // Create the SQL insert statement

                sqlOss.str("");
                sqlOss << "insert into " << tableName << " (";
                for (j = 0; j < dataVec.size(); j++) {
                        if (j != 0) sqlOss << ", ";
                        sqlOss << colVec[dataVec[j].dataColNo].colName;
                }
                sqlOss << ") values (";
                for (j = 0; j < dataVec.size(); j++) {
                        if (j != 0) sqlOss << ", ";
                        i = dataVec[j].dataColNo;
                        switch (colVec[i].colType) {
                        case COL_TYPE_INTEGER:
                                sqlOss << dataVec[j].dataIntVal;
                                break;
                        case COL_TYPE_CHAR:
                                sqlOss << ToDbForm (
                                        dataVec[j].dataCharVal);
                                break;
                        default:
                                Error ("unsupported column type");
                        }
                }
                sqlOss << ')';
                sqlStmt = sqlOss.str();
                SQL (sqlStmt.c_str());
        }

        // Verify that there is no superfluous data in the table chain

        if (tableSource.Read (&tag, 1) != 0)
                Error ("superfluous table data");
    }
    return 0;
}
```