```
// Andergen.cpp - GENERATE AN EXAMPLE ANDERBASE DBMS BACKUP FILE
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 03-10-05     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 DESTINATION BASE-CLASS

class Dest_c {
public:
        virtual void Write (const void *buf, size_t size) = 0;
                                        // Write data to the destination
};

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

// STANDARD OUTPUT SOURCE

class OutDest_c : public Dest_c {
public:
        void Write (const void *buf, size_t size)
                { cout.write (static_cast<const char*>(buf), size); }
                                        // Write data to the destination
};

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

// TAG-LENGTH-VALUE DESTINATION

class TagDest_c : public Dest_c {
        Dest_c          *tagDest;       // Lower level destination
        Tag_t           tagNo;          // Tag number
        char            tagBuf[MAX_VAL_LEN]; // Data buffer
        size_t          tagLen;         // Length in the buffer

        void WriteEnvelope ();
                                        // Write the current envelope
public:
        void Open (Dest_c *lowerDest, Tag_t tag);
                                        // Open the tag-length-value chain
        void Write (const void *buf, size_t size);
                                        // Write data to the destination
        void Close ();
                                        // Close the tag-length-value chain
};
```

```
//---------------------------------------------------------------------------

// WRITE THE CURRENT ENVELOPE

void
TagDest_c::WriteEnvelope ()
{
        char            lenBuf[2];      // Tag length buffer

        tagDest->Write (&tagNo, 1);
        lenBuf[0] = tagLen >> 8;
        lenBuf[1] = tagLen & 0xff;
        tagDest->Write (lenBuf, 2);
        tagDest->Write (tagBuf, tagLen);
        tagLen = 0;
}

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

// OPEN A TAG-LENGTH-VALUE CHAIN

void
TagDest_c::Open (
        Dest_c          *lowerDest,     // Lower level destination
        Tag_t           tag)            // Tag number
{
        tagDest = lowerDest;
        tagNo = tag;
        tagLen = 0;
}

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

// WRITE DATA TO A TAG-LENGTH-VALUE CHAIN

void
TagDest_c::Write (
        const void      *buf,           // Data buffer
        size_t          size)           // Data size
{
        size_t          ofs;            // Offset in the data buffer
        size_t          copyLen;        // Copy length

        ofs = 0;
        while (ofs < size) {
                if (tagLen == MAX_VAL_LEN) WriteEnvelope ();
                copyLen = size - ofs;
                if (copyLen > MAX_VAL_LEN-tagLen) copyLen = MAX_VAL_LEN-tagLen;
                memcpy (tagBuf+tagLen, static_cast<const char*>(buf)+ofs,
                        copyLen);
                tagLen += copyLen;
                ofs += copyLen;
        }
}

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

// CLOSE A TAG-LENGTH-VALUE CHAIN

void
TagDest_c::Close ()
{
        size_t          lastSegLen;     // Last segment length
```

```
        lastSegLen = tagLen;
        WriteEnvelope ();
        if (lastSegLen == MAX_VAL_LEN) WriteEnvelope ();
}

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

// WRITE A STRING VALUE

void
WriteString (
        Dest_c          *dest,          // Destination
        Tag_t           tagNo,          // Tag number
        const char      *value)         // String value
{
        TagDest_c       tagDest;        // Tag destination

        tagDest.Open (dest, tagNo);
        tagDest.Write (value, strlen(value));
        tagDest.Close ();
}

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

// WRITE AN UNSIGNED VALUE

void
WriteUnsigned (
        Dest_c          *dest,          // Destination
        Tag_t           tagNo,          // Tag number
        unsigned long   value,          // Value to write
        size_t          size)           // Value size
{
        TagDest_c       tagDest;        // Tag destination
        unsigned char   valBuf[4];      // Value buffer
        unsigned char   *p;             // General purpsoe pointer
        size_t          i;              // General purpose index

        p = valBuf + sizeof(valBuf);
        for (i = 0; i < size; i++) {
                *--p = static_cast<unsigned char>(value & 0xff);
                value >>= 8;
        }
        tagDest.Open (dest, tagNo);
        tagDest.Write (p, size);
        tagDest.Close ();
}

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

// WRITE A SIGNED VALUE

void
WriteSigned (
        Dest_c          *dest,          // Destination
        Tag_t           tagNo,          // Tag number
        long            value,          // Value to write
        size_t          size)           // Value size
{
        TagDest_c       tagDest;        // Tag destination
        unsigned char   valBuf[4];      // Value buffer
        unsigned char   *p;             // General purpsoe pointer
        size_t          i;              // General purpose index

        p = valBuf + sizeof(valBuf);
```

```
        for (i = 0; i < size; i++) {
                *--p = static_cast<unsigned char>(value & 0xff);
                value >>= 8;
        }
        tagDest.Open (dest, tagNo);
        tagDest.Write (p, size);
        tagDest.Close ();
}

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

// MAIN LINE

int
main ()
{
        OutDest_c       outDest;        // Output destination
        TagDest_c       tableDest;      // Table destination
        TagDest_c       secDest;        // Section destination
        TagDest_c       colDest;        // Column destination
        TagDest_c       rowDest;        // Row destination
        size_t          i, j;           // General purpose index
        ostringstream   oss;            // Output string stream
        string          val;            // Component value

        // Create the orders table

        tableDest.Open (&outDest, 0x01);
        secDest.Open (&tableDest, 0x02);
        WriteString (&secDest, 0x03, "orders");

        colDest.Open (&secDest, 0x04);
        WriteString (&colDest, 0x05, "orderNo");        // Column name
        WriteUnsigned (&colDest, 0x06, 0x10, 1);        // Column tag no
        WriteUnsigned (&colDest, 0x07, 0x04, 1);        // Integer type
        colDest.Close ();

        colDest.Open (&secDest, 0x04);
        WriteString (&colDest, 0x05, "orderName");      // Column name
        WriteUnsigned (&colDest, 0x06, 0x11, 1);        // Column tag no
        WriteUnsigned (&colDest, 0x08, 40, 2);          // Char(40) type
        colDest.Close ();

        secDest.Close ();

        // Create table rows

        secDest.Open (&tableDest, 0x09);
        for (i = 0; i < 1000; i++) {
                rowDest.Open (&secDest, 0x0a);

                // Write the order number

                WriteSigned (&rowDest, 0x10, static_cast<long>(i), 4);

                // Write the order nuame

                oss.str("");
                oss << "ORDER NAME " << i;
                val = oss.str();
                WriteString (&rowDest, 0x11, val.c_str());

                // Close the order row

                rowDest.Close ();
```

```
        }
        secDest.Close ();
        tableDest.Close ();

        // Create the items table

        tableDest.Open (&outDest, 0x01);
        secDest.Open (&tableDest, 0x02);
        WriteString (&secDest, 0x03, "items");

        colDest.Open (&secDest, 0x04);
        WriteString (&colDest, 0x05, "orderNo");        // Column name
        WriteUnsigned (&colDest, 0x06, 0x12, 1);        // Column tag no
        WriteUnsigned (&colDest, 0x07, 0x04, 1);        // Integer type
        colDest.Close ();

        colDest.Open (&secDest, 0x04);
        WriteString (&colDest, 0x05, "itemName");       // Column name
        WriteUnsigned (&colDest, 0x06, 0x13, 1);        // Column tag no
        WriteUnsigned (&colDest, 0x08, 40, 2);          // Char(40) type
        colDest.Close ();

        colDest.Open (&secDest, 0x04);
        WriteString (&colDest, 0x05, "quantity");       // Column name
        WriteUnsigned (&colDest, 0x06, 0x14, 1);        // Column tag no
        WriteUnsigned (&colDest, 0x07, 0x04, 1);        // Integer type
        colDest.Close ();

        secDest.Close ();

        // Create table rows

        secDest.Open (&tableDest, 0x09);
        for (i = 0; i < 1000; i++) {
                for (j = 0; j < 10; j++) {
                        rowDest.Open (&secDest, 0x0a);

                        // Write the order number

                        WriteSigned (&rowDest, 0x12, static_cast<long>(i), 4);

                        // Write the item nuame

                        oss.str("");
                        oss << "ITEM " << j << " IN ORDER '" << i << "'";
                        val = oss.str();
                        WriteString (&rowDest, 0x13, val.c_str());

                        // Write the quantity

                        WriteSigned (&rowDest, 0x14, rand() % 100, 4);

                        // Close the item row

                        rowDest.Close ();
                }
        }
        secDest.Close ();
        tableDest.Close ();

        return 0;
}
```