```
//*****************************************************************************
*******//
// FILE:                lz77.c
// PROJECT:             zarLib  - ver 1.00 -
// COPYRIGHTS:  The famous group 4 of "Roskilde Universites Center"
//*****************************************************************************
*******//
// Headers:
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>
#include <search.h>
#include <string.h>
#include "zarLib.h"
#include "main.h"


//*****************************************************************************
****
// Constants:

#define MIN_POINTER_BIT_SIZE            9               // in bits
#define MAX_POINTER_BIT_SIZE            16              // in bits

#define LOOK_AHEAD_BIT_SIZE                     4               // in bits
#define LOOK_AHEAD_SIZE                         (2 << (LOOK_AHEAD_BIT_SIZE -1))

#define LOOK_BACK_BIT_SIZE                      12
#define LOOK_BACK_SIZE                          (2 << (LOOK_BACK_BIT_SIZE -1))

#define MAX_BUFFER_SIZE                         (2 << 20)

#define WINDOW_SIZE                                     (LOOK_AHEAD_SIZE +LOOK_B
ACK_SIZE)

#define READ_CACHE                                      480


//*****************************************************************************
****
// Function prototypes:

static void             runCompression(_byte* pBuffer, _dword dwBufSize, int nUn
readData, ZAR_FILE* pzDst, const ZAR_FILE* pzSrc);
static void             runExtraction(_byte* pBuffer, _dword dwBufSize,  ZAR_FIL
E* pzDst, const ZAR_FILE* pzSrc);
static _dword   recacheBuffer(_byte* pBuffer, _byte** ppWndReadPos, _byte** ppWn
dBackPos, _dword dwAheadSize, const ZAR_FILE* pzSrc);
static void             resetBuffer(_byte* pBuffer, _byte** ppReadPos, int nSize
);

//*****************************************************************************
****
// lz77Compress()

int lz77Compress(ZAR_FILE* pzSrc, ZAR_FILE* pzDst)
{
        _dword  dwBufSize;
        _byte*  pBuffer;
        int             nUnreadData;

        // Print info:
        printf("--- Starting LZ77 compression ---\n");
        printf("Progress: ");

        // Get the file size of the file to compress:
        nUnreadData = zarFileSize(pzSrc);
        dwBufSize   = min(nUnreadData, MAX_BUFFER_SIZE);        // The number of
 bytes in the buffer:
        nUnreadData -= dwBufSize;
```



```
        // Allocate memory for the buffer:
        if (NULL == (pBuffer = malloc(dwBufSize)))
                return zarError(ZAR_ERR_FAILED_MEMORY_ALLOC, NULL);

        // Read the data into the buffer:
        if (dwBufSize != zarGetBytes(pzSrc, pBuffer, dwBufSize))
        {
                free(pBuffer);
                return zarError(ERR_INPUT_FILE_NOT_READABLE, NULL);
        }

        // Compress the shit:
        runCompression(pBuffer, dwBufSize, nUnreadData, pzDst, pzSrc);  // stupi
d name, but VC++ has some bugs, which fuck functions with the same name!

        // Clean-up:
        free(pBuffer);

        return 1;
}

//*****************************************************************************
****
// lz77Extract()

int lz77Extract(ZAR_FILE* pzSrc, ZAR_FILE* pzDst)
{
        _dword  dwBufSize;
        _byte*  pBuffer;

        // Print info:
        printf("--- Starting LZ77 extraction ---\n");
        printf("Progress: ");

        // Init:
        dwBufSize = MAX_BUFFER_SIZE;

        // Allocate memory for buffer:
        if (NULL == (pBuffer = malloc(dwBufSize)))
                return zarError(ZAR_ERR_FAILED_MEMORY_ALLOC, NULL);

        runExtraction(pBuffer, dwBufSize, pzDst, pzSrc);

        free(pBuffer);

        return 1;
}



//*****************************************************************************
****
// runCompression()

static void runCompression(_byte* pBuffer, _dword dwBufSize, int nUnreadData, ZA
R_FILE* pzDst, const ZAR_FILE* pzSrc)
{
        _byte* pBufEnd = pBuffer +dwBufSize;    // The end of the buffer (NEVER
go pass this point!)

        _byte*  pReadPos;               // The current position
(static)
        _byte*  pReadBack;              // The last position of the window
(static)
        _byte*  pReadAhead;             // The last position of the window  (sta
tic)

        _byte*  pFwdTmp;                // Forward pointer to search for match
```

```
        _byte*  pRwdTmp;                // Back pointer to search for match
        _byte*  pMatch;                 // The largest match found
        int     cxMatch;                // The count of the largest match
        int             cxCurMatch;             // The count of the current matc
h
        _dword  dwOut;

        // Init pointers:
        pReadBack       = pBuffer;
        pReadPos        = pBuffer;

        while (1)
        {
                // Init data:
                cxCurMatch = 0;
                pReadAhead = min(pReadPos +LOOK_AHEAD_SIZE -1, pBufEnd -1);

                do // Search for matches:
                {
                        // Init pointers:
                        pFwdTmp = pReadPos;
                        pRwdTmp = pReadBack;

                        cxMatch = 0;
                        pMatch  = pReadPos;     // smart, because (pReadPos -pMa
tch, gives index)

                        // Search up to the current position:
                        while (pRwdTmp < pReadPos)
                        {
                                // Search for match:
                                if (*pFwdTmp == *pRwdTmp)
                                {
                                        cxCurMatch = 0;

                                        do
                                        { // Go to next position:


                                                pRwdTmp++;
                                                pFwdTmp++;
                                                cxCurMatch++;

                                                if (pFwdTmp == pReadAhead || pRw
dTmp == pReadPos)
                                                        break;

                                        } while (*pFwdTmp == *pRwdTmp);

                                        // Did we find a larger match?
                                        if (cxCurMatch > cxMatch)
                                        {
                                                // Save the largest match:
                                                cxMatch = cxCurMatch;
                                                pMatch  = pRwdTmp -cxMatch;

                                                if (cxMatch == 15)
                                                        break;
                                        }

                                        pFwdTmp = pReadPos;
                                }

                                // Get the next letter:
                                pRwdTmp++;
                        }

                        // Output the current symbol:
                        zarSetByte(pzDst, pReadPos[cxMatch]);
```



```
                        // Output pointer to data (if any)
                        dwOut = ((pReadPos -pMatch) << 4) +cxMatch;
                        zarSetBytesF(pzDst, (_byte*) &dwOut, 2);        // Use f
unction version so this function will fit into the cache

                        // Update:
                        pReadPos   += cxMatch +1;
                        pReadAhead  = pReadPos +LOOK_AHEAD_SIZE -1;
                        pReadBack   = max(pBuffer, pReadPos -LOOK_BACK_SIZE +1);

                } while(pReadAhead < pBufEnd);

                // Should we read up more data?
                if (nUnreadData != 0)
                {
                        int nReadData;
                        int nTmp = pBufEnd -pReadPos;

                        nReadData = recacheBuffer(pBuffer, &pReadPos, &pReadBack
, nTmp, pzSrc);
                        pReadBack = pBuffer;
                        pBufEnd   = min(pBuffer +dwBufSize, pReadPos +nTmp +nRea
dData);
                        nUnreadData -= nReadData;
                }
                else
                {
                        // Have we raeched the end yet?
                        if (pReadPos < (pBufEnd -1))
                                pReadAhead = pBufEnd -1;
                        else
                                break;
                }
        }
}


//***************************************************************************
****
// recacheBuffer()

static _dword recacheBuffer(_byte* pBuffer, _byte** ppWndReadPos, _byte** ppWndB
ackPos, _dword dwAheadSize, const ZAR_FILE* pzSrc)
{
        _byte* pBack      = *ppWndBackPos;
        _byte* pRead      = *ppWndReadPos;
        _dword dwMoveSize = pRead -pBack +dwAheadSize;
        _byte* pGetPos    = pBuffer +dwMoveSize;

        // Move the window:
        memcpy(pBuffer, pBack, dwMoveSize);
        memset(pBuffer +dwMoveSize, 0, MAX_BUFFER_SIZE -dwMoveSize);

        // Reallocate pointers:
        *ppWndBackPos = pBuffer;
        *ppWndReadPos = pBuffer +LOOK_BACK_SIZE -1;

        // Read the last data:
        return zarGetBytesF(pzSrc, pGetPos, MAX_BUFFER_SIZE -dwMoveSize);
}

//***************************************************************************
****
// runExtraction()

static void runExtraction(_byte* pBuffer, _dword dwBufSize,  ZAR_FILE* pzDst, co
nst ZAR_FILE* pzSrc)
{
        _byte* pEndofCache, *pEndofBuffer;
        _byte  readCache[READ_CACHE], bChar;
```

```
        _byte* pReadPos, *pWritePos, *pFlush;
        _dword dwValue, dwBack, dwLen;
        _dword dwCacheSize = READ_CACHE;

        pWritePos        = pBuffer;
        pEndofBuffer = pBuffer +dwBufSize;
        pFlush      = pBuffer;

        while(dwCacheSize == READ_CACHE)
        {
                // Read up data:
                dwCacheSize = zarGetBytes(pzSrc, readCache, READ_CACHE);
                pReadPos    = readCache;
                pEndofCache = readCache +dwCacheSize;

                while (pReadPos < pEndofCache)
                {
                        // Extract the fucking byte:
                        dwValue = *((_dword*) pReadPos);
                        bChar   = (_byte) dwValue;
                        dwLen   = (dwValue >> 8)   & 0x0000000f;
                        dwBack  = (dwValue >> 12)  & 0x00000fff;


                        // Test for buffer recache:
                        if (pWritePos +dwLen +1 >= pEndofBuffer)
                        {
                                zarSetBytesF(pzDst, pBuffer, pWritePos -pBuffer)
;
                                resetBuffer(pBuffer, &pWritePos, LOOK_BACK_SIZE)
;
                                pFlush = pBuffer + (pWritePos -pBuffer);
                        }

                        // Write the output:
                        if (dwBack != 0)
                        {
                                memcpy(pWritePos, pWritePos -dwBack, dwLen);
                                pWritePos += dwLen;
                        }

                        // Write the byte:
                        *pWritePos = bChar;

                        // Update pointers:
                        pReadPos += 3;
                        pWritePos++;
                }
        }

        // Flush last data:
        if (pFlush)
                zarSetBytes(pzDst, pFlush, (pWritePos -pFlush));
}


//*************************************************************************
****
// resetBuffer()

static void resetBuffer(_byte* pBuffer, _byte** ppWritePos, int nSize)
{
        _byte* pos = *ppWritePos -nSize +1;

        // Move old memory:
        memcpy(pBuffer, pos, nSize);
        //memset(pBuffer +nSize, 0, MAX_BUFFER_SIZE -nSize);

        // Reallocate pointer:
        *ppWritePos = pBuffer +nSize -1;
```



```
}
```