BAB VII

KESIMPULAN DAN SARAN

A. Kesimpulan

Penelitian ini mengusulkan modifikasi algoritma JBE yang dilakukan dengan cara mengeliminasi simbol nol dan simbol satu dari output pertama sehingga output pertama akan berisi data asli selain nol dan satu (dalam ukuran *byte*) dan output kedua akan berisi nilai dua bit yang menjelaskan posisi *byte* nol, *byte* satu, dan *byte* selain nol dan satu. Berdasarkan hasil penelitian dan pembahasan yang telah dilakukan, dapat dipetik beberapa kesimpulan, yaitu :

1. Rasio kompresi dari kedua algoritma ini dapat semakin ditingkatkan apabila disisipkan diantara tahapan GST dan EC dalam skema BWCA (menggantikan RLE).
2. Skema kombinasi algoritma yang menghasilkan rata-rata rasio kompresi terbaik sesuai urutannya yaitu:

a. BWT + MTF + algoritma hasil modifikasi + pengkodean aritmatika dengan 0,3357 untuk *dataset* calagary corpus dan 0,3002 untuk *dataset* canterbury corpus.
b. BWT + M1FF + algoritma hasil modifikasi + pengkodean aritmatika dengan 0,3363 untuk *dataset* calagary corpus dan 0,3005 untuk *dataset* canterbury corpus.
c. BWT + MTF + JBE + pengkodean aritmatika dengan 0,3426 untuk *dataset* calagary corpus dan 0,3128 untuk *dataset* canterbury corpus.

d. BWT + M1FF + JBE + pengkodean aritmatika dengan 0,3442 untuk *dataset* calagary corpus dan 0,3109 untuk *dataset* canterbury corpus.

Sehingga dapat dinyatakan bahwa algoritma hasil modifikasi lebih efektif dibandingkan dengan algoritma JBE.

3. Algoritma hasil modifikasi juga lebih efisien dibandingkan dengan algoritma JBE.

4. Algoritma JBE dan algoritma hasil modifikasi sama-sama memiliki kelebihan yaitu rasio kompresi yang dapat dicapai dapat diketahui sebelum data dikompresi.

B. Saran

Perlu dikaji perbandingan kinerja algoritma JBE dan algoritma hasil modifikasi dengan menggunakan *dataset* lainnya seperti *dataset* silesia corpus (Deorowicz, 2003) dengan menggunakan skema BWCA sebagaimana yang dilakukan dalam penelitian ini dan dengan ukuran blok BWT yang berbeda atau dengan menggunakan skema kombinasi algoritma lainnya.

DAFTAR PUSTAKA

Abel, J., 2005. *A Fast and Efficient Post BWT-Stage for The Burrows-Wheeler Compression Algorithm.* s.l., IEEE.

Abel, J., 2010. Post BWT Stages of The Burrows-Wheeler Compression Algorithm. *Software Practice and Experience,* XL(9), p. 751–777.

Abualkishik, A. M. & Omar, K., 2008 . Quranic Braille System. *International Journal of Computer, Electrical, Automation, Control and Information Engineering ,* II(10), pp. 3306-3312.

Adewumi, S. E., 2015. Character Analysis Scheme for Compressing Text Files. *International Journal ofComputer Theory and Engineering,* VII(5), pp. 362-365.

Adjeroh, D., Bell, T. & Mukherjee, A., 2008. *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching.* 1st penyunt. New York: Springer US.

Al-Bahadili, H. & Hussain, S. M., 2008. An adaptive character wordlength algorithm for data compression. *Computers and Mathematics with Applications,* LV(6), p. 1250–1256.

Ambadekar, S., Gandhi, K., Nagaria, J. & Shah, R., 2015. Advanced Data Compression Using J-bit Algorithm. *International Journal of Science and Research,* IV(3), pp. 1366-1368.

Ambadekar, S., Gandhi, K., Nagaria, J. & Shah, R., 2015. Advanced Data Compression Using J-bit Algorithm. *International Journal of Science and Research,* IV(3), pp. 1366-1368.

Balkenhol , B., Kurtz, S. & Shtarkov, Y. M., 1999. Data Compression Conference, 1999. Proceedings. DCC '99. *Modifications of the Burrows and Wheeler data compression algorithm,* March, pp. 188-197.

Bell, T., Powell, M., Horlor, J. & Arnold, R., 1997. *The Canterbury Corpus.* [Online] Available at: http://corpus.canterbury.ac.nz [Diakses 1 10 2015].

Burrows, M. & Wheeler, D. J., 1994. *A Block-sorting Lossless Data Compression Algorithm,* Palo Alto, California: Digital Systems Research Center Report.

Chang, W., Fang, B., Xiaochun, Y. & Wang, S., 2009. The Block Lossless Data Compression Algorithm. *International Journal of Computer Science and Network Security*, IX(10), pp. 116-123.

Chatterjee, S., 2013. *Block Based Data Compression Using Burrows- Wheeler Transform*, Kolkata, West Bengal: Jadavpur University.

Chatterjee, S., 2013. *Block Based Data Compression Using Burrows-Wheeler Transform*, Kolkata-India: Department Of Information Technology Jadavpur University.

David , S., 2006. *Data Compression The Complete Reference.* 4th penyunt. London: Springer–Verlag.

David, S., 2006. *Data Compression The Complete Reference.* 4th penyunt. London: Springer–Verlag.

David, S., 2007. *Variable-length Codes for Data Compression.* London: Springer–Verlag.

Deorowicz, S., 2003. *Silesia Compression Corpus.* [Online] Available at: http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia [Diakses 12 May 2016].

Fenwick, P., 2007. Burrows–Wheeler Compression: Principles and Reflections. *Theoretical Computer Science*, CCCLXXXIX(3), pp. 200-219.

Grewal, M. K. & Kaur, M. S., 2014. Encryption and Compression of Audio-Video Data Using Enhanced AES and J-Bit Algorithm. *International Journal for Scientific Research & Development*, II(8), pp. 159-165.

Hamid, M. T. & Ayyash, M. S., 2011. *Implementation of A Digital Communucation System*, Palestine: An-Najah National University.

Jain, A., Lakhtaria, K. I. & Srivastava, P., 2013. *A Comparative Study of Lossless Compression Algorithm on Text Data.* Elsevier India, International Conference on Advances in Engineering and Technology.

Karkkainen, J., 2007. Fast BWT in Small Space by Blockwise Suffix Sorting. *Theoretical Computer Science*, November, CCCLXXXIX(3), pp. 249-257.

Kodituwakku, S. R. & Amarasinghe, U. S., 2010. Comparison of Lossless Data Compression Algorithms for Text Data. *Indian Journal of Computer Science and Engineering*, I(4), pp. 416-425.

Mantaci, S., Restivo, A., Rosone, G. & Sciortino, M., 2007. An extension of the Burrows–Wheeler Transform. *Theoretical Computer Science,* November, CCCLXXXIX(3), p. 298–312.

Pradhan, A. et al., 2016. A Comparative Analysis of Compression Techniques – The Sparse Coding and BWT. *Procedia Computer Science,* Volume XCII, pp. 106 - 111.

Priyanto, D. & Nur, M., 2014. A Literacy Learning Smart System for the Blind Children. *International Journal of Computer and Information Technology ,* III(5), pp. 1100-1103.

Rathore, Y., Ahirwar, M. K. & Pandey, R., 2013. A Brief Study of Data Compression Algorithms. *International Journal of Computer Science and Information Security,* XI(10), pp. 86-94.

R, B. et al., 2014. A Lightweight Randomized Low Sampling Compression Technique Verified by GI with Merits over CR and IT Reconstruction Schemes. *International Journal of Innovative Research in Electronics and Communications (IJIREC),* I(5), pp. 22-33.

Sadiq, A. T., Duaimi, M. G. & Ali, R. S., 2013. Large Dataset Compression Approach Using Intelligent Technique. *Journal of Advanced Computer Science and Technology Research,* III(1), pp. 1-20.

Sasilal, L. & Govindan, V. K., 2013. Arithmetic Coding-A Reliable Implementation. *International Journal of Computer Applications,* LXXIII(7), pp. 1-5.

Sayood, K., 2006. *Introduction to Data Compression.* 3rd penyunt. United States of America: Morgan Kaufmann.

Seward, J., 2007. *A program and library for data compression.* [Online] Available at: http://www.bzip.org/ [Diakses 21 6 2016].

Shanmugasundaram, S. & Lourdusamy, R., 2011. A Comparative Study Of Text Compression Algorithms. *International Journal of Wisdom Based Computing,* I(3), pp. 68-76.

Sharma, N., Kaur, J. & Kaur, N., 2014. A Review on various Lossless Text Data Compression Techniques. *Research Cell: An International Journal of Engineering Sciences,* XII(2), pp. 58-63.

Singla, T. & Kumar, R., 2014. ECG Signal Monitored Under Various Compression Techniques and Transmission Environments - A Survey Approach. *International Journal of Applied Research in Computing*, II(4), pp. 69-74.

Steinruecken, C., 2014. *Lossless Data Compression*, Cambridge: University of Cambridge.

Suarjaya, I. M. A. D., 2012. A New Algorithm for Data Compression Optimization. *International Journal of Advanced Computer Science and Applications*, III(8), pp. 14-17.

Suchendra, D. R. & Wulandari, S., 2012. Implementasi Kompresi Data Text Menggunakan Huffman Coding. *JURNAL LPKIA*, I(1), pp. 22-27.

Syahrul, E., Dubois, J. & Vajnovszki, V., 2011. Combinatorial Transforms : Application in Lossless Image Compression. *Journal of Discrete Mathematical Sciences and Cryptography*, XIV(2), pp. 129-147.

Warkade, P. & Mishra, A., 2015. Lossless Speech Compression Techniques: A Literature Review. *International Journal of Innovative Research in Computer Science & Technology (IJIRCST)*, III(3), pp. 25-32.

Wiseman, Y., 2007. Burrows-Wheeler Based JPEG. *Data Science Journal*, Volume VI, pp. 19-27.

Witten, I. H., Bell, T. & Cleary, J. G., 1987. *Calgary corpus*. [Online] Available at: ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus [Diakses 1 10 2015].

LAMPIRAN

A. Source Code Program

1. bwt.h

```
/*****************************************************************/
                Header for Burrows-Wheeler Transform Library
File     :  bwt.h
Purpose  :  Provides that apply and reverse the Burrows-Wheeler transform (with
            or without move to front coding/decoding transformation).
Author   :  Michael Dipperstein
Date     :  August 20, 2004
/*****************************************************************/
#ifndef _BWT_H_
#define _BWT_H_

/*****************************************************************/
Fungsi      : BWTEncode
Deskripsi   : Fungsi ini membaca input berupa blok memory dengan ukuran 1 MB
              dan mengkodekannya dengan transformasi Burrows-Wheeler.
Parameter   : buffer1  - pointer ke blok memori berisi simbol yang akan dikodekan
              length   - panjang blok memori (buffer1)
Nilai balik : Nilai indeks berukuran 4 byte jika tidak ada kesalahan, nilai
              0xEF1E0AA5 (4 byte) jika terdapat kesalahan.
/*****************************************************************/
unsigned int BWTEncode(unsigned char *buffer1, int length);

/*****************************************************************/
Fungsi      :  BWTDecode
Deskripsi   :  Fungsi ini membaca input berupa blok memory dengan ukuran
               1 MB dan mengkodekannya dengan invers transformasi Burrows-
               Wheeler.
Parameter   :  buffer1  - pointer ke blok memori berisi simbol yang akan
               dikodekan
               length   - panjang blok memori (buffer1)
               s0Idx    - indeks posisi simbol pertama
Nilai balik :  Nilai 0x00000000 jika tidak ada kesalahan,
               nilai 0xEF1E0AA5 jika terdapat kesalahan.
/*****************************************************************/
unsigned int BWTDecode(unsigned char *buffer1, int length, int s0Idx);

#endif
```

2. bwt.c

```
/************************** INCLUDED FILES ************************/
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>
#include "bwt.h"

/*************************** CONSTANTS ****************************/
#define BLOKBWT  ((1<<20))      /* Ukuran Blok untuk 1 MB simbol*/
#if BLOKBWT > INT_MAX
#error BLOKBWT harus <= INT_MAX
#endif

/************************* TYPE DEFINITIONS ***********************/
unsigned char block[BLOKBWT];
size_t blockSize;
unsigned int counters[256], offsetTable[256];

/***************************** MACROS *****************************/
/* wraps array index within array bounds (assumes value < 2 * limit) */
#define Wrap(value, limit)      (((value) < (limit)) ? (value) : ((value) - (limit)))

/***************************** FUNCTIONS **************************/
/******************************************************************/
Function    :   ComparePresorted
Description:    This comparison function is designed for use with qsort and "block",
                a global array of "blockSize" unsigned chars. It compares two
                strings in "block" starting at indices s1 and s2 and ending at indices
                s1 - 1 and s2 - 1. The strings are assumed to be presorted so that first
                two characters are known to be matching.
Parameters:     s1 - The starting index of a string in block
                s2 - The starting index of a string in block
Returned    :   > 0 if string s1 > string s2
                0 if string s1 == string s2
                < 0 if string s1 < string s2
******************************************************************/
int ComparePresorted1(const void *s1, const void *s2)
{
    int offset1, offset2, i, result;
    offset1 = *((int *)s1);
    offset2 = *((int *)s2);
    for(i = 2; i < blockSize; i++)
    {       result =  (int)block[Wrap((offset1 + i), blockSize)] -
                      (int)block[Wrap((offset2 + i), blockSize)];
```

```
        if (result != 0)
        {   return result;   }
    }
    /* strings are identical */
    return 0;
}

/********************** FUNGSI : BWTENCODE ********************/
unsigned int BWTEncode(unsigned char *buffer1, int length)
{
    int i, j, k;
    FILE *fpIn, *fpOut;
    unsigned int *rotationIdx;      /* index of first char in rotation */
    unsigned int *v;                /* index of radix sorted charaters */
    int s0Idx;                      /* index of S0 in rotations (I) */
    unsigned char *last;            /* last characters from sorted rotations */

    rotationIdx = (unsigned int *)malloc(BLOKBWT * sizeof(unsigned int));
    if (NULL == rotationIdx)
    {   perror("Tidak dapat mengalokasi memory untuk rotasi!");
        return 0xEF1E0AA5;   }
    v = (unsigned int *)malloc(BLOKBWT * sizeof(unsigned int));
    if (v == rotationIdx)
    {   perror("Tidak dapat mengalokasi memory untuk pengurutan!");
        free(rotationIdx);     return 0xEF1E0AA5;   }
    last = (unsigned char *)malloc(BLOKBWT * sizeof(unsigned char));
    if (NULL == last)
    {   perror("Tidak dapat mengalokasi memory untuk kolom L!");
        free(rotationIdx);     free(v);  return 0xEF1E0AA5;   }
    blockSize =length;
    memcpy((void *)block, (void *)buffer1, sizeof(unsigned char) * length);
    memset(counters, 0, 256 * sizeof(int));
    for (i = 0; i < blockSize; i++)
    {   counters[block[i]]++;   }
    offsetTable[0] = 0;
    for(i = 1; i < 256; i++)
    {   offsetTable[i] = offsetTable[i - 1] + counters[i - 1];   }
    for (i = 0; i < blockSize - 1; i++)
    {   j=block[i + 1]; v[offsetTable[j]]=i;
        offsetTable[j]=offsetTable[j]+1; }
    j = block[0];
    v[offsetTable[j]] = i;
    offsetTable[0] = 0;
    for(i = 1; i < 256; i++)
    { offsetTable[i] = offsetTable[i - 1] + counters[i - 1]; }
    for (i = 0; i < blockSize; i++)
```

```
    {   j = v[i];  j = block[j];    rotationIdx[offsetTable[j]] = v[i];
        offsetTable[j] = offsetTable[j] + 1; }
    for (i = 0, k = 0; (i <= UCHAR_MAX) && (k < (blockSize - 1)); i++)
    {
        for (j = 0; (j <= UCHAR_MAX) && (k < (blockSize - 1)); j++)
        {   int first = k;
            while ((i == block[rotationIdx[k]]) &&
                    (j == block[Wrap(rotationIdx[k] + 1,  blockSize)]))
            {   k++;
                if (k == blockSize)
                {   break;    }
            }
            if (k - first > 1)
            {   qsort(&rotationIdx[first], k - first, sizeof(int), ComparePresorted1); }
        }
    }
    s0Idx = 0;
    for (i = 0; i < blockSize; i++)
    {   if (rotationIdx[i] != 0)   {   last[i] = block[rotationIdx[i] - 1];    }
        else   {   s0Idx = i;    last[i] = block[blockSize - 1];     }
    }
    memcpy((void *)buffer1, (void *)last, sizeof(unsigned char) * length);
    free(rotationIdx); free(v);   free(last);
    return s0Idx;
}

/********************** FUNGSI : BWTDECODE **********************/
unsigned int BWTDecode(unsigned char *buffer1, int length, int s0Idx)
{
    FILE *fpIn, *fpOut;
    int i, j, sum;
    int count[UCHAR_MAX + 1];
    int *pred;
    unsigned char *unrotated;

    pred = (int *)malloc(BLOKBWT * sizeof(int));
    if (NULL == pred)
    {   perror("Tidak dapat mengalokasi memory pengurutan!");
        return 0xEF1E0AA5;   }
    unrotated = (unsigned char *)malloc(BLOKBWT * sizeof(unsigned char));
    if (NULL == unrotated)
    {   perror("Tidak dapat mengalokasi memory untuk rotasi blok!");
        free(pred);       return 0xEF1E0AA5;   }
    blockSize =length;
    memcpy((void *)block, (void *)buffer1, sizeof(unsigned char) * length);
    for(i = 0; i <= UCHAR_MAX; i++)
```

```
    {   count[i] = 0;    }
    for (i = 0; i < blockSize; i++)
    { pred[i]=count[block[i]]; count[block[i]]++; }
    sum = 0;
    for(i = 0; i <= UCHAR_MAX; i++)
    { j=count[i]; count[i]=sum; sum+=j; }
    i = s0Idx;
    for(j = blockSize - 1; j >= 0; j--)
    {   unrotated[j] = block[i]; i = pred[i] + count[block[i]];  }
    memcpy((void *)buffer1, (void *)unrotated, sizeof(unsigned char) * length);
    free(pred);  free(unrotated);
    return 1;
}
```

3. bitfile.h

```
/*************************************************************************/
                        Bit Stream File Header
File        : bitfile.h
Purpose     : Provides definitions and prototypes for a simple library of I/O functions
              for files that contain data in sizes that aren't integral bytes. An attempt
              was made to make the functions in this library analogous to functions
              provided to manipulate byte streams. The functions contained in this
              library were created with compression algorithms in mind, but may be
              suited to other applications.
Author      : Michael Dipperstein
Date        : January 9, 2004
/*************************************************************************/
Bitfile: Bit stream File I/O Routines
Copyright (C) 2004-2007 by Michael Dipperstein (mdipper@cs.ucsb.edu)
This file is part of the bit file library. The bit file library is free software; you can
redistribute it and/or modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either version 3 of the
License, or (at your option) any later version.
The bit file library is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public
License for more details.
You should have received a copy of the GNU Lesser General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
/*************************************************************************/
#ifndef _BITFILE_H_
#define _BITFILE_H_

/************************ INCLUDED FILES ************************/
#include <stdio.h>
```

```
/********************** TYPE DEFINITIONS **********************/
typedef enum
{
    BF_READ = 0,
    BF_WRITE = 1,
    BF_APPEND= 2,
    BF_NO_MODE
} BF_MODES;

/* incomplete type to hide implementation */
struct bit_file_t;
typedef struct bit_file_t bit_file_t;

/************************** PROTOTYPES **************************/
/* open/close file */
bit_file_t *BitFileOpen(const char *fileName, const BF_MODES mode);
bit_file_t *MakeBitFile(FILE *stream, const BF_MODES mode);
int BitFileClose(bit_file_t *stream);
FILE *BitFileToFILE(bit_file_t *stream);

/* toss spare bits and byte align file */
int BitFileByteAlign(bit_file_t *stream);

/* fill byte with ones or zeros and write out results */
int BitFileFlushOutput(bit_file_t *stream, const unsigned char onesFill);

/* get/put character */
int BitFileGetChar(bit_file_t *stream);
int BitFilePutChar(const int c, bit_file_t *stream);

/* get/put single bit */
int BitFileGetBit(bit_file_t *stream);
int BitFilePutBit(const int c, bit_file_t *stream);

/* get/put number of bits (most significant bit to least significat bit) */
int BitFileGetBits(bit_file_t *stream, void *bits, const unsigned int count);
int BitFilePutBits(bit_file_t *stream, void *bits, const unsigned int count);

/*****************************************************************/
get/put number of bits from integer types (short, int, long, ...) machine endiness is
accounted for. size is the size of the data structure pointer to by bits.
/*****************************************************************/
int BitFileGetBitsInt(bit_file_t *stream, void *bits, const unsigned int count,
const size_t size);
int BitFilePutBitsInt(bit_file_t *stream, void *bits, const unsigned int count,
const size_t size);

#endif
```

4. bitfile.c

```
/*****************************************************************/
                    Bit Stream File Implementation
File        :  bitfile.c
Purpose     :  This file implements a simple library of I/O functions for files that
               contain data in sizes that aren't integral bytes. An attempt was made
               to make the functions in this library analogous to functions provided
               to manipulate byte streams. The functions contained in this library
               were created with compression algorithms in mind, but may be suited
               to other applications.
Author      :  Michael Dipperstein
Date        :  January 9, 2004
/*****************************************************************/
Bitfile: Bit stream File I/O Routines
Copyright (C) 2004-2007 by Michael Dipperstein (mdipper@cs.ucsb.edu)
This file is part of the bit file library.
The bit file library is free software; you can redistribute it and/or modify it under
the terms of the GNU Lesser General Public License as published by the Free
Software Foundation; either version 3 of the License, or (at your option) any later
version. The bit file library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Lesser General Public License for more details.
You should have received a copy of the GNU Lesser General Public License along
with this program.  If not, see <http://www.gnu.org/licenses/>.
/*****************************************************************/

/************************ INCLUDED FILES ************************/
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include "bitfile.h"

/************************ TYPE DEFINITIONS ************************/
typedef enum
{   BF_UNKNOWN_ENDIAN,
    BF_LITTLE_ENDIAN,
    BF_BIG_ENDIAN
} endian_t;

struct bit_file_t
{   FILE *fp;                   /* file pointer used by stdio functions */
    endian_t endian;            /* endianess of architecture            */
    unsigned char bitBuffer;    /* bits waiting to be read/written      */
    unsigned char bitCount;     /* number of bits in bitBuffer          */
    BF_MODES mode;              /* open for read, write, or append      */
};
```

```
/* union used to test for endianess */
typedef union
{
    unsigned long word;
    unsigned char bytes[sizeof(unsigned long)];
} endian_test_t;

/************************** PROTOTYPES ***************************/
endian_t DetermineEndianess(void);
int BitFilePutBitsLE(bit_file_t *stream, void *bits, const unsigned int count);
int BitFilePutBitsBE(bit_file_t *stream, void *bits, const unsigned int count, const
                     size_t size);
int BitFileGetBitsLE(bit_file_t *stream, void *bits, const unsigned int count);
int BitFileGetBitsBE(bit_file_t *stream, void *bits, const unsigned int count, const
                     size_t size);

/************************** FUNCTIONS ***************************/
/****************************************************************/
Function    :  BitFileOpen
Description:  This function opens a bit file for reading, writing, or appending.  If
              successful, a bit_file_t data structure will be allocated and a pointer
              to the structure will be returned.
Parameters:   fileName  -  NULL terminated string containing the name of the
                           file to be opened.
              mode      -  The mode of the file to be opened
Effects     :  The specified file will be opened and file structure will be allocated.
Returned    :  Pointer to the bit_file_t structure for the bit file opened, or NULL on
              failure.  errno will be set for all failure cases.
****************************************************************/
bit_file_t *BitFileOpen(const char *fileName, const BF_MODES mode)
{
    char modes[3][3] = {"rb", "wb", "ab"};  /* binary modes for fopen */
    bit_file_t *bf;
    bf = (bit_file_t *)malloc(sizeof(bit_file_t));
    if (bf == NULL)
    {   errno = ENOMEM;  }
    else
    {   bf->fp = fopen(fileName, modes[mode]);
        if (bf->fp == NULL)
        {   free(bf); bf = NULL;}
        else
        {   bf->bitBuffer = 0;    bf->bitCount = 0;
            bf->mode = mode;   bf->endian = DetermineEndianess();  }
    }
    return (bf);
}
```

```
/*************************************************************/
```
Function : MakeBitFile
Description: *This function naively wraps a standard file in a bit_file_t structure*. ANSI-C *doesn't support file status functions commonly found in other* C *variants, so the caller must be passed as a parameter*.
Parameters : stream - *pointer to the standard file being wrapped*.
mode - *The mode of the file being wrapped*.
Effects : *A bit_file_t structure will be created for the stream passed as a parameter.*
Returned : *Pointer to the bit_file_t structure for the bit file or NULL on failure. errno will be set for all failure cases.*
```
/*************************************************************/
bit_file_t *MakeBitFile(FILE *stream, const BF_MODES mode)
{
    bit_file_t *bf;
    if (stream == NULL)
    {
        errno = EBADF;
        bf = NULL;
    }
    else
    {
        bf = (bit_file_t *)malloc(sizeof(bit_file_t));
        if (bf == NULL)
        {
            errno = ENOMEM;
        }
        else
        {
            bf->fp = stream;  bf->bitBuffer = 0; bf->bitCount = 0;
            bf->mode = mode;    bf->endian = DetermineEndianess();
        }
    }
    return (bf);
}

/*************************************************************/
```
Function : DetermineEndianess
Description : *This function determines the endianess of the current hardware architecture. An unsigned long is set to 1. If the 1st byte of the unsigned long gets the 1, this is a little endian machine. If the last byte gets the 1, this is a big endian machine.*
Parameters : *None*
Effects : *None*
Returned : *endian_t for current machine architecture*
```
/*************************************************************/
```

```
endian_t DetermineEndianess(void)
{
   endian_t endian;
   endian_test_t endianTest;
   endianTest.word = 1;
   if (endianTest.bytes[0] == 1)
   {   endian = BF_LITTLE_ENDIAN;        }
   else if (endianTest.bytes[sizeof(unsigned long) - 1] == 1)
   {   endian = BF_BIG_ENDIAN;   }
   else
   {   endian = BF_UNKNOWN_ENDIAN; }
   return endian;
}

/*************************************************************************/
Function     :  BitFileClose
Description  :  This function closes a bit file and frees all associated data.
Parameters   :  stream - pointer to bit file stream being closed
Effects      :  The specified file will be closed and the file structure will be freed.
Returned     :  0 for success or EOF for failure.
/*************************************************************************/
int BitFileClose(bit_file_t *stream)
{
   int returnValue = 0;
   if (stream == NULL)
   {   return(EOF);   }
   if ((stream->mode == BF_WRITE) || (stream->mode == BF_APPEND))
   {
      if (stream->bitCount != 0)
      {
         (stream->bitBuffer) <<= 8 - (stream->bitCount);
         fputc(stream->bitBuffer, stream->fp);  }
   }
   returnValue = fclose(stream->fp);
   free(stream);
   return(returnValue);
}

/*************************************************************************/
Function     :  BitFileToFILE
Description  :  This function flushes and frees the bitfile structure, returning a
                pointer to a stdio file.
Parameters   :  stream - pointer to bit file stream being closed
Effects      :  The specified bitfile will be made usable as a stdio FILE.
Returned     :  Pointer to FILE.  NULL for failure.
/*************************************************************************/
```

```
FILE *BitFileToFILE(bit_file_t *stream)
{
    FILE *fp = NULL;
    if (stream == NULL)
    {   return(NULL);    }
    if ((stream->mode == BF_WRITE) || (stream->mode == BF_APPEND))
    {
        if (stream->bitCount != 0)
        {
            (stream->bitBuffer) <<= 8 - (stream->bitCount);
            fputc(stream->bitBuffer, stream->fp);
        }
    }
    fp = stream->fp;
    free(stream);
    return(fp);
}

/*****************************************************************/
```

Function : BitFileByteAlign

Description : *This function aligns the bitfile to the nearest byte. For output files, this means writing out the bit buffer with extra bits set to 0. For input files, this means flushing the bit buffer.*

Parameters : stream - *pointer to bit file stream to align*

Effects : *Flushes out the bit buffer.*

Returned : *EOF if stream is NULL or write fails. Writes return the byte aligned contents of the bit buffer. Reads returns the unaligned contents of the bit buffer.*

```
/*****************************************************************/
int BitFileByteAlign(bit_file_t *stream)
{
    int returnValue;
    if (stream == NULL)
    {   return(EOF);   }
    returnValue = stream->bitBuffer;
    if ((stream->mode == BF_WRITE) || (stream->mode == BF_APPEND))
    {   if (stream->bitCount != 0)
        {
            (stream->bitBuffer) <<= 8 - (stream->bitCount);
            fputc(stream->bitBuffer, stream->fp);
        }
    }
    stream->bitBuffer = 0;
    stream->bitCount = 0;
    return (returnValue);
}
```

```
/*****************************************************************/
Function    :  BitFileFlushOutput
Description :  This function flushes the output bit buffer.  This means left
               justifying any pending bits, and filling spare bits with the fill
               value.
Parameters  :  stream  -  pointer to bit file stream to align onesFill - non-zero
                          if spare bits are filled with ones
Effects     :  Flushes out the bit buffer, filling spare bits with ones or zeros.
Returned    :  EOF if stream is NULL or not writeable.  Otherwise, the bit buffer
               value written. -1 if no data was written.
/*****************************************************************/
int BitFileFlushOutput(bit_file_t *stream, const unsigned char onesFill)
{
    int returnValue;
    if (stream == NULL)
    {   return(EOF);   }
    returnValue = -1;
    if (stream->bitCount != 0)
    {
        stream->bitBuffer <<= (8 - stream->bitCount);
        if (onesFill)
        {   stream->bitBuffer |= (0xFF >> stream->bitCount); }
        returnValue = fputc(stream->bitBuffer, stream->fp);
    }
    stream->bitBuffer = 0;
    stream->bitCount = 0;
    return (returnValue);
}

/*****************************************************************/
Function    :  BitFileGetChar
Description :  This function returns the next byte from the file passed as a
               parameter.
Parameters  :  stream  -  pointer to bit file stream to read from
Effects     :  Reads next byte from file and updates buffer accordingly.
Returned    :  EOF if a whole byte cannot be obtained.  Otherwise, the
               character read.
/*****************************************************************/
int BitFileGetChar(bit_file_t *stream)
{
    int returnValue;
    unsigned char tmp;
    if (stream == NULL)
    {   return(EOF);   }
    returnValue = fgetc(stream->fp);
    if (stream->bitCount == 0)
```

```
    {   return returnValue;  }
    if (returnValue != EOF)
    {
        tmp = ((unsigned char)returnValue) >> (stream->bitCount);
        tmp |= ((stream->bitBuffer) << (8 - (stream->bitCount)));
        stream->bitBuffer = returnValue;
        returnValue = tmp;
    }
    return returnValue;
}

/*****************************************************************/
Function    :  BitFilePutChar
Description :  This function writes the byte passed as a parameter to the file
               passed a parameter.
Parameters  :  c       -  the character to be written
               stream  -  pointer to bit file stream to write to
Effects     :  Writes a byte to the file and updates buffer accordingly.
Returned    :  On success, the character written, otherwise EOF.
/*****************************************************************/
int BitFilePutChar(const int c, bit_file_t *stream)
{
    unsigned char tmp;
    if (stream == NULL)
    {   return(EOF);   }
    if (stream->bitCount == 0)
    {   return fputc(c, stream->fp);      }
    tmp = ((unsigned char)c) >> (stream->bitCount);
    tmp = tmp | ((stream->bitBuffer) << (8 - stream->bitCount));
    if (fputc(tmp, stream->fp) != EOF)
    {   stream->bitBuffer = c;   }
    else
    {   return EOF;    }
    return tmp;
}


/*****************************************************************/
Function    :  BitFileGetBit
Description :  This function returns the next bit from the file passed as a
               parameter.  The bit value returned is the msb in the bit buffer.
Parameters  :  stream - pointer to bit file stream to read from
Effects     :  Reads next bit from bit buffer.  If the buffer is empty, a new byte
               will be read from the file.
Returned    :  0 if bit == 0, 1 if bit == 1, and EOF if operation fails.
/*****************************************************************/
```

```
int BitFileGetBit(bit_file_t *stream)
{
    int returnValue;
    if (stream == NULL)
    {   return(EOF);   }
    if (stream->bitCount == 0)
    {
        if ((returnValue = fgetc(stream->fp)) == EOF)
        {   return EOF;    }
        else
        {   stream->bitCount = 8;
            stream->bitBuffer = returnValue;
        }
    }
    stream->bitCount--;
    returnValue = (stream->bitBuffer) >> (stream->bitCount);
    return (returnValue & 0x01);
}

/*****************************************************************/
Function    :  BitFilePutBit
Description :  This function writes the bit passed as a parameter to the file
               passed a parameter.
Parameters  :  c       -  the bit value to be written
               stream  -  pointer to bit file stream to write to
Effects     :  Writes a bit to the bit buffer.  If the buffer has a byte, the buffer is
               written to the file and cleared.
Returned    :  On success, the bit value written, otherwise EOF.
/*****************************************************************/
int BitFilePutBit(const int c, bit_file_t *stream)
{
    int returnValue = c;
    if (stream == NULL)
    {   return(EOF);   }
    stream->bitCount++;
    stream->bitBuffer <<= 1;
    if (c != 0)
    {   stream->bitBuffer |= 1;  }
    if (stream->bitCount == 8)
    {   if (fputc(stream->bitBuffer, stream->fp) == EOF)
        {   returnValue = EOF;  }
        stream->bitCount = 0;
        stream->bitBuffer = 0;
    }
    return returnValue;
}
```

```
/*****************************************************************/
```
Function : BitFileGetBits
Description : *This function reads the specified number of bits from the file passed as a parameter and writes them to the requested memory location (msb to lsb).*
Parameters : stream - *pointer to bit file stream to read from*
bits - *address to store bits read*
count - *number of bits to read*
Effects : *Reads bits from the bit buffer and file stream. The bit buffer will be modified as necessary.*
Returned : *EOF for failure, otherwise the number of bits read. If an EOF is reached before all the bits are read, bits will contain every bit through the last complete byte.*

```
/*****************************************************************/
int BitFileGetBits(bit_file_t *stream, void *bits, const unsigned int count)
{
   unsigned char *bytes, shifts;
   int offset, remaining, returnValue;
   bytes = (unsigned char *)bits;
   if ((stream == NULL) || (bits == NULL))
   {   return(EOF);   }
   offset = 0;  remaining = count;
   while (remaining >= 8)
   {
      returnValue = BitFileGetChar(stream);
      if (returnValue == EOF)
      {   return EOF;    }
      bytes[offset] = (unsigned char)returnValue;
      remaining -= 8;    offset++;
   }
   if (remaining != 0)
   {
      shifts = 8 - remaining;   bytes[offset] = 0;
      while (remaining > 0)
      {
         returnValue = BitFileGetBit(stream);
         if (returnValue == EOF)
         {   return EOF;    }
         bytes[offset] <<= 1;
         bytes[offset] |= (returnValue & 0x01);
         remaining--;
      }
      bytes[offset] <<= shifts;
   }
   return count;
}
```

```
/*************************************************************************/
```
Function : BitFilePutBits
Description : *This function writes the specified number of bits from the memory location passed as a parameter to the file passed as a parameter. Bits are written msb to lsb.*
Parameters : stream - *pointer to bit file stream to write to*
bits - *pointer to bits to write*
count - *number of bits to write*
Effects : *Writes bits to the bit buffer and file stream. The bit buffer will be modified as necessary.*
Returned : *EOF for failure, otherwise the number of bits written. If an error occurs after a partial write, the partially written bits will not be unwritten.*

```
/*************************************************************************/
int BitFilePutBits(bit_file_t *stream, void *bits, const unsigned int count)
{
   unsigned char *bytes, tmp;
   int offset, remaining, returnValue;
   bytes = (unsigned char *)bits;
   if ((stream == NULL) || (bits == NULL))
   {   return(EOF);   }
   offset = 0;
   remaining = count;
   while (remaining >= 8)
   {
      returnValue = BitFilePutChar(bytes[offset], stream);
      if (returnValue == EOF)
      {   return EOF;    }
      remaining -= 8;
      offset++;
   }
   if (remaining != 0)
   {
      tmp = bytes[offset];
      while (remaining > 0)
      {
         returnValue = BitFilePutBit((tmp & 0x80), stream);
         if (returnValue == EOF)
         {   return EOF;    }
         tmp <<= 1;
         remaining--;
      }
   }
   return count;
}
```

```
/*****************************************************************/
Function    :  BitFileGetBitsInt
Description :  This function provides a machine independent layer that allows a
               single function call to stuff an arbitrary number of bits into an
               integer type variable.
Parameters  :  stream  -  pointer to bit file stream to read from
               bits    -  address to store bits read
               count   -  number of bits to read
               size    -  sizeof type containing "bits"
Effects     :  Calls a function that reads bits from the bit buffer and file stream.
               The bit buffer will be modified as necessary. the bits will be
               written to "bits" from least significant byte to most significant
               byte.
Returned    :  EOF for failure, otherwise the number of bits read by the called
               function.
/*****************************************************************/
int BitFileGetBitsInt(bit_file_t *stream, void *bits, const unsigned int count,
const size_t size)
{
   int returnValue;
   if ((stream == NULL) || (bits == NULL))
   {  return(EOF);  }
   if (stream->endian == BF_LITTLE_ENDIAN)
   {  returnValue = BitFileGetBitsLE(stream, bits, count);  }
   else if (stream->endian == BF_BIG_ENDIAN)
   {  returnValue = BitFileGetBitsBE(stream, bits, count, size);    }
   else
   {  returnValue = EOF; }
   return returnValue;
}

/*****************************************************************/
Function    :  BitFileGetBitsLE   (Little Endian)
Description :  This function reads the specified number of bits from the file
               passed as a parameter and writes them to the requested memory
               location (LSB to MSB).
Parameters  :  stream  -  pointer to bit file stream to read from
               bits    -  address to store bits read
               count   -  number of bits to read
Effects     :  Reads bits from the bit buffer and file stream.  The bit buffer will
               be modified as necessary.  bits is treated as a little endian integer
               of length >= (count/8) + 1.
Returned    :  EOF for failure, otherwise the number of bits read.  If an EOF is
               reached before all the bits are read, bits will contain every bit
               through the last successful read.
/*****************************************************************/
```

```
int BitFileGetBitsLE(bit_file_t *stream, void *bits, const unsigned int count)
{
    unsigned char *bytes;
    int offset, remaining, returnValue;
    bytes = (unsigned char *)bits;
    offset = 0;  remaining = count;
    while (remaining >= 8)
    {   returnValue = BitFileGetChar(stream);
        if (returnValue == EOF)
        {   return EOF;    }
        bytes[offset] = (unsigned char)returnValue;
        remaining -= 8;       offset++;
    }
    if (remaining != 0)
    {
        while (remaining > 0)
        {   returnValue = BitFileGetBit(stream);
            if (returnValue == EOF)
            { return EOF; }
            bytes[offset] <<= 1;
            bytes[offset] |= (returnValue & 0x01);
            remaining--;
        }
    }
    return count;
}

/*****************************************************************/
```

Function : BitFileGetBitsBE *(Big Endian)*

Description : *This function reads the specified number of bits from the file passed as a parameter and writes them to the requested memory location (LSB to MSB).*

Parameters : stream - *pointer to bit file stream to read from*
bits - *address to store bits read*
count - *number of bits to read*
size - *sizeof type containing "bits"*

Effects : *Reads bits from the bit buffer and file stream. The bit buffer will be modified as necessary. bits is treated as a big endian integer of length size.*

Returned : *EOF for failure, otherwise the number of bits read. If an EOF is reached before all the bits are read, bits will contain every bit through the last successful read.*

```
/*****************************************************************/
int BitFileGetBitsBE(bit_file_t *stream, void *bits, const unsigned int count,
const size_t size)
{
```

```
    unsigned char *bytes;
    int offset, remaining, returnValue;
    if (count > (size * 8))
    {   return EOF;    }
    bytes = (unsigned char *)bits;
    offset = size - 1;
    remaining = count;
    while (remaining >= 8)
    {   returnValue = BitFileGetChar(stream);
        if (returnValue == EOF)
        {   return EOF;    }
        bytes[offset] = (unsigned char)returnValue;
        remaining -= 8;
        offset--;
    }
    if (remaining != 0)
    {
        while (remaining > 0)
        {   returnValue = BitFileGetBit(stream);
            if (returnValue == EOF)
            {   return EOF;    }
            bytes[offset] <<= 1;
            bytes[offset] |= (returnValue & 0x01);
            remaining--;
        }
    }
    return count;
}

/*********************************************************************/
```

Function : BitFilePutBitsInt

Description : *This function provides a machine independent layer that allows a single function call to write an arbitrary number of bits from an integer type variable into a file.*

Parameters : stream - *pointer to bit file stream to write to*
bits - *pointer to bits to write*
count - *number of bits to write*
size - *sizeof type containing "bits"*

Effects : *Calls a function that writes bits to the bit buffer and file stream. The bit buffer will be modified as necessary. the bits will be written to the file stream from least significant byte to most significant byte.*

Returned : *EOF for failure, otherwise the number of bits written. If an error occurs after a partial write, the partially written bits will not be unwritten.*

```
/*********************************************************************/
```

```
int BitFilePutBitsInt(bit_file_t *stream, void *bits, const unsigned int count,
const size_t size)
{
    int returnValue;
    if ((stream == NULL) || (bits == NULL))
    {   return(EOF);   }
    if (stream->endian == BF_LITTLE_ENDIAN)
    {   returnValue = BitFilePutBitsLE(stream, bits, count);   }
    else if (stream->endian == BF_BIG_ENDIAN)
    {   returnValue = BitFilePutBitsBE(stream, bits, count, size);      }
    else
    {   returnValue = EOF; }
    return returnValue;
}

/*********************************************************************/
Function      :  BitFilePutBitsLE   (Little Endian)
Description   :  This function writes the specified number of bits from the memory
                 location passed as a parameter to the file passed as a parameter.
                 Bits are written LSB to MSB.
Parameters    :  stream - pointer to bit file stream to write to
                 bits - pointer to bits to write
                 count - number of bits to write
Effects       :  Writes bits to the bit buffer and file stream.  The bit buffer will be
                 modified as necessary.  bits is treated as a little endian integer of
                 length >= (count/8) + 1.
Returned      :  EOF for failure, otherwise the number of bits written.  If an error
                 occurs after a partial write, the partially written bits will not be
                 unwritten.
/*********************************************************************/
int BitFilePutBitsLE(bit_file_t *stream, void *bits, const unsigned int count)
{
    unsigned char *bytes, tmp;
    int offset, remaining, returnValue;
    bytes = (unsigned char *)bits;
    offset = 0;
    remaining = count;
    while (remaining >= 8)
    {
        returnValue = BitFilePutChar(bytes[offset], stream);
        if (returnValue == EOF)
        {   return EOF;    }
        remaining -= 8;
        offset++;
    }
```

```
    if (remaining != 0)
    {
        tmp = bytes[offset];
        tmp <<= (8 - remaining);
        while (remaining > 0)
        {
            returnValue = BitFilePutBit((tmp & 0x80), stream);
            if (returnValue == EOF)
            {   return EOF;    }
            tmp <<= 1;
            remaining--;
        }
    }
    return count;
}

/*************************************************************/
Function    :  BitFilePutBitsBE   (Big Endian)
Description :  This function writes the specified number of bits from the memory
               location passed as a parameter to the file passed as a parameter.
               Bits are written LSB to MSB.
Parameters  :  stream - pointer to bit file stream to write to
               bits - pointer to bits to write
               count - number of bits to write
Effects     :  Writes bits to the bit buffer and file stream.  The bit buffer will be
               modified as necessary.  bits is treated as a big endian integer of
               length size.
Returned    :  EOF for failure, otherwise the number of bits written.  If an error
               occurs after a partial write, the partially written bits will not be
               unwritten.
/*************************************************************/
int BitFilePutBitsBE(bit_file_t *stream, void *bits, const unsigned int count,
const size_t size)
{
    unsigned char *bytes, tmp;
    int offset, remaining, returnValue;
    if (count > (size * 8))
    {   return EOF;    }
    bytes = (unsigned char *)bits;
    offset = size - 1;
    remaining = count;
    while (remaining >= 8)
    {
        returnValue = BitFilePutChar(bytes[offset], stream);
        if (returnValue == EOF)
        {   return EOF;    }
```

```
        remaining -= 8;
        offset--;
    }

    if (remaining != 0)
    {
        tmp = bytes[offset];
        tmp <<= (8 - remaining);
        while (remaining > 0)
        {
            returnValue = BitFilePutBit((tmp & 0x80), stream);
            if (returnValue == EOF)
            {   return EOF;    }
            tmp <<= 1;
            remaining--;
        }
    }
    return count;
}
```

5. arcode.h

```
/*************************************************************************/
                Header for Arithmetic Encoding and Decoding Library
File        :   arcode.h
Purpose     :   Provides prototypes for functions that use arithmetic coding
                to encode/decode files.
Author      :   Michael Dipperstein
Date        :   April 2, 2004
/*************************************************************************/
Arcode: An ANSI C Arithmetic Encoding/Decoding Routines
Copyright (C) 2004, 2006-2007 by Michael Dipperstein (mdipper@cs.ucsb.edu)
This file is part of the arcode library.
The arcode library is free software; you can redistribute it and/or modify it under
the terms of the GNU Lesser General Public License as published by the Free
Software Foundation; either version 3 of the License, or (at your option) any later
version. The arcode library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Lesser General Public License for more details. You should have received a
copy of the GNU Lesser General Public License along with this program.  If not,
see <http://www.gnu.org/licenses/>.
/*************************************************************************/
#ifndef _ARCODE_H_
#define _ARCODE_H_
```

```
/************************** CONSTANTS ***************************/
#ifndef FALSE
#define FALSE      0
#endif

#ifndef TRUE
#define TRUE       1
#endif

/************************** PROTOTYPES ***************************/
/* encode inFile */
int ArEncodeFile(FILE *inFile, FILE *outFile, char staticModel);

/* decode inFile*/
int ArDecodeFile(FILE *inFile, FILE *outFile, char staticModel);

#
```

6. arcode.c

```
/*****************************************************************/
                    Arithmetic Encoding and Decoding Library
File        :  arcode.c
Purpose     :  Use arithmetic coding to compress/decompress file streams
Author      :  Michael Dipperstein
Date        :  April 2, 2004
/*****************************************************************/
Arcode: An ANSI C Arithmetic Encoding/Decoding Routines
Copyright (C) 2004, 2006-2007 by Michael Dipperstein (mdipper@cs.ucsb.edu)
This file is part of the arcode library. The arcode library is free software; you can
redistribute it and/or modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either version 3 of the
License, or (at your option) any later version.
The arcode library is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public
License for more details. You should have received a copy of the GNU Lesser
General Public License along with this program.
If not, see <http://www.gnu.org/licenses/>.
/*****************************************************************/

/************************** INCLUDED FILES ***************************/
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <assert.h>
```

```
#include "arcode.h"
#include "bitfile.c"

#ifdef NDEBUG
#define PrintDebug(ARGS) do {} while (0)
#else
#define PrintDebug(ARGS) printf ARGS
#endif

#if !(USHRT_MAX < ULONG_MAX)
#error "Implementation requires USHRT_MAX < ULONG_MAX"
#endif

/********************** TYPE DEFINITIONS **********************/
#define EOF_CHAR    (UCHAR_MAX + 1)
typedef unsigned short probability_t;       / probability count type */
typedef struct
{
    probability_t ranges[EOF_CHAR + 2];
    probability_t cumulativeProb;
    probability_t lower;
    probability_t upper;
    probability_t code;
    unsigned char underflowBits;
} stats_t;


/************************** CONSTANTS ***************************/
/* number of bits used to compute running code values */
#define PRECISION           (8 * sizeof(probability_t))

/* 2 bits less than precision. keeps lower and upper bounds from crossing. */
#define MAX_PROBABILITY     (1 << (PRECISION - 2))

/***************************** MACROS ***************************/
/* set bit x to 1 in probability_t.  Bit 0 is MSB */
#define MASK_BIT(x) (probability_t)(1 << (PRECISION - (1 + (x))))

/* indices for a symbol's lower and upper cumulative probability ranges */
#define LOWER(c)    (c)
#define UPPER(c)    ((c) + 1)

/************************** PROTOTYPES **************************/
/* read write file headers */
void WriteHeader(bit_file_t *bfpOut, stats_t *stats);
int ReadHeader(bit_file_t *bfpIn, stats_t *stats);
```

```
/* applies symbol's ranges to current upper and lower range bounds */
void ApplySymbolRange(int symbol, stats_t *stats, char staticModel);

/* routines for encoding*/
void WriteEncodedBits(bit_file_t *bfpOut, stats_t *stats);
void WriteRemaining(bit_file_t *bfpOut, stats_t *stats);
int BuildProbabilityRangeList(FILE *fpIn, stats_t *stats);
void InitializeAdaptiveProbabilityRangeList(stats_t *stats);

/* routines for decoding */
void InitializeDecoder(bit_file_t *bfpOut, stats_t *stats);
probability_t GetUnscaledCode(stats_t *stats);
int GetSymbolFromProbability(probability_t probability, stats_t *stats);
void ReadEncodedBits(bit_file_t *bfpIn, stats_t *stats);

/************************** FUNCTIONS **************************/
/***************************************************************/
Function    :  ArEncodeFile
Description :  This routine generates a list of arithmetic code ranges for a file
               and then uses them to write out an encoded version of that file.
Parameters  :  inFile      -  FILE stream to encode
               outFile     -  FILE stream to write encoded output to
               staticModel -  TRUE if encoding with a static model
Effects     :  File is arithmetically encoded
Returned    :  TRUE for success, otherwise FALSE.
***************************************************************/
int ArEncodeFile(FILE *inFile, FILE *outFile, char staticModel)
{
    int c;
    bit_file_t *bOutFile;
    stats_t stats;

    if (NULL == inFile)
    {   inFile = stdin; }

    if (outFile == NULL)
    {   bOutFile = MakeBitFile(stdout, BF_WRITE); }
    else
    {   bOutFile = MakeBitFile(outFile, BF_WRITE); }

    if (NULL == bOutFile)
    {
        fprintf(stderr, "Error: Creating binary output file\n");
        return FALSE;
    }
```

```
    if (staticModel)
    {   if (!BuildProbabilityRangeList(inFile, &stats))
        {   fclose(inFile);
            BitFileClose(bOutFile);
            fprintf(stderr, "Error determining frequency ranges.\n");
            return FALSE;
        }
        rewind(inFile);  WriteHeader(bOutFile, &stats);
    }
    else
    {   InitializeAdaptiveProbabilityRangeList(&stats); }
    stats.lower = 0;
    stats.upper = ~0;
    stats.underflowBits = 0;
    while ((c = fgetc(inFile)) != EOF)
    {   ApplySymbolRange(c, &stats, staticModel);
        WriteEncodedBits(bOutFile, &stats);
    }
    ApplySymbolRange(EOF_CHAR, &stats, staticModel);
    WriteEncodedBits(bOutFile, &stats);
    WriteRemaining(bOutFile, &stats);
    outFile = BitFileToFILE(bOutFile);
    return TRUE;
}

/*****************************************************************/
Function    :  SymbolCountToProbabilityRanges
Description :  This routine converts the ranges array containing only symbol
               counts to an array containing the upper and lower probability
               ranges for each symbol.
Parameters  :  stats - structure containing data used to encode symbols
Effects     :  ranges struct containing symbol counts in the upper field for
               each symbol is converted to a list of upper and lower probability
               bounds for each symbol.
Returned    :  None
*****************************************************************/
void SymbolCountToProbabilityRanges(stats_t *stats)
{
    int c;
    stats->ranges[0] = 0;
    stats->ranges[UPPER(EOF_CHAR)] = 1;
    stats->cumulativeProb++;
    for (c = 1; c <= UPPER(EOF_CHAR); c++)
    {   stats->ranges[c] += stats->ranges[c - 1]; }
    return;
}
```

```
/*****************************************************************/
Function    :  BuildProbabilityRangeList
Description :  This routine reads the input file and builds the global list of
               upper and lower probability ranges for each symbol.
Parameters  :  fpIn  -  file to build range list for stats - structure containing
                        data used to encode symbols
Effects     :  stats struct is made to contain probability ranges for each
               symbol.
Returned    :  TRUE for success, otherwise FALSE.
/*****************************************************************/
int BuildProbabilityRangeList(FILE *fpIn, stats_t *stats)
{
    int c;
    unsigned long countArray[EOF_CHAR];
    unsigned long totalCount = 0;
    unsigned long rescaleValue;
    if (fpIn == NULL)
    {   return FALSE; }
    for (c = 0; c < EOF_CHAR; c++)
    { countArray[c] = 0; }
    while ((c = fgetc(fpIn)) != EOF)
    {   if (totalCount == ULONG_MAX)
        {   fprintf(stderr, "Error: file too large\n");
            return FALSE;
        }
        countArray[c]++;        totalCount++;
    }

    if (totalCount >= MAX_PROBABILITY)
    {   rescaleValue = (totalCount / MAX_PROBABILITY) + 1;
        for (c = 0; c < EOF_CHAR; c++)
        {   if (countArray[c] > rescaleValue)
            {   countArray[c] /= rescaleValue; }
            else if (countArray[c] != 0)
            {  countArray[c] = 1; }
        }
    }
    stats->ranges[0] = 0;
    stats->cumulativeProb = 0;
    for (c = 0; c < EOF_CHAR; c++)
    {   stats->ranges[UPPER(c)] = countArray[c];
        stats->cumulativeProb += countArray[c];
    }
    SymbolCountToProbabilityRanges(stats);
    return TRUE;
}
```

```
/*****************************************************************/
Function    :  WriteHeader
Description :  This function writes each symbol contained in the encoded file
               as well as its rescaled number of occurrences.  A decoding
               algorithm may use these numbers to reconstruct the probability
               range list used to encode the file.
Parameters  :  bfpOut  -  pointer to open binary file to write to.
               stats   -  structure containing data used to encode symbols
Effects     :  Symbol values and symbol counts are written to a file.
Returned    :  None
/*****************************************************************/
void WriteHeader(bit_file_t *bfpOut, stats_t *stats)
{
    int c;      probability_t previous = 0;
    for(c = 0; c <= (EOF_CHAR - 1); c++)
    {   if (stats->ranges[UPPER(c)] > previous)
        {   BitFilePutChar((char)c, bfpOut);
            previous = (stats->ranges[UPPER(c)] - previous);
            BitFilePutBitsInt(bfpOut, &previous, (PRECISION - 2),
            sizeof(probability_t));
            previous = stats->ranges[UPPER(c)];         }
    }
    BitFilePutChar(0x00, bfpOut);      previous = 0;
    BitFilePutBits(bfpOut, (void *)&previous, PRECISION - 2);
}

/*****************************************************************/
Function    :  InitializeAdaptiveProbabilityRangeList
Description :  This routine builds the initial global list of upper and lower
               probability ranges for each symbol.  This routine is used by both
               adaptive encoding and decoding. Currently it provides a
               uniform symbol distribution. Other distributions might be better
               suited for known data types (such as English text).
Parameters  :  stats - structure containing data used to encode symbols
Effects     :  ranges array is made to contain initial probability ranges for
               each symbol.
Returned    :  NONE
/*****************************************************************/
void InitializeAdaptiveProbabilityRangeList(stats_t *stats)
{
    int c;      stats->ranges[0] = 0;
    for (c = 1; c <= UPPER(EOF_CHAR); c++)
    {   stats->ranges[c] = stats->ranges[c - 1] + 1;      }
    stats->cumulativeProb = UPPER(EOF_CHAR);
    return;
}
```

```
/*****************************************************************/
Function    :  ApplySymbolRange
Description :  This function is used for both encoding and decoding.  It applies
               the range restrictions of a new symbol to the current upper and
               lower range bounds of an encoded stream. If an adaptive model
               is being used, the probability range list will be updated after the
               effect of the symbol is applied.
Parameters  :  symbol      - The symbol to be added to the current code range
               stats       - structure containing data used to encode symbols
               staticModel - TRUE if encoding/decoding with a static model.
Effects     :  The current upper and lower range bounds are adjusted to
               include the range effects of adding another symbol to the
               encoded stream.  If an adaptive model is being used, the
               probability range list will be updated.
Returned    :  None
/*****************************************************************/
void ApplySymbolRange(int symbol, stats_t *stats, char staticModel)
{
   unsigned long range, rescaled;
   int i;  probability_t original;  probability_t delta;
   range = (unsigned long)(stats->upper - stats->lower) + 1;
   rescaled = (unsigned long)(stats->ranges[UPPER(symbol)]) * range;
   rescaled /= (unsigned long)(stats->cumulativeProb);
   stats->upper = stats->lower + (probability_t)rescaled - 1;
   rescaled = (unsigned long)(stats->ranges[LOWER(symbol)]) * range;
   rescaled /= (unsigned long)(stats->cumulativeProb);
   stats->lower = stats->lower + (probability_t)rescaled;
   if (!staticModel)
   {  stats->cumulativeProb++;
      for (i = UPPER(symbol); i <= UPPER(EOF_CHAR); i++)
      {  stats->ranges[i] += 1;         }
      if (stats->cumulativeProb >= MAX_PROBABILITY)
      {  stats->cumulativeProb = 0;  original = 0;
         for (i = 1; i <= UPPER(EOF_CHAR); i++)
         {  delta = stats->ranges[i] - original;
            original = stats->ranges[i];
            if (delta <= 2)
            {  stats->ranges[i] = stats->ranges[i - 1] + 1;     }
            else
            {  stats->ranges[i] = stats->ranges[i - 1] + (delta / 2);    }
            stats->cumulativeProb +=(stats->ranges[i] - stats->ranges[i - 1]);
         }
      }
   }
   assert(stats->lower <= stats->upper);
}
```

```
/*****************************************************************/
```
Function : WriteEncodedBits
Description : *This function attempts to shift out as many code bits as possible, writing the shifted bits to the encoded output file. Only bits that will be unchanged when additional symbols are encoded may be written out.*
If the n most significant bits of the lower and upper range bounds match, they will not be changed when additional symbols are encoded, so they may be shifted out. Adjustments are also made to prevent possible underflows that occur when the upper and lower ranges are so close that encoding another symbol won't change their values.
Parameters : bfpOut - *pointer to open binary file to write to.*
stats - *structure containing data used to encode symbols*
Effects : *The upper and lower code bounds are adjusted so that they only contain only bits that may be affected by the addition of a new symbol to the encoded stream.*
Returned : *None*

```
/*****************************************************************/
void WriteEncodedBits(bit_file_t *bfpOut, stats_t *stats)
{
    for (;;)
    {   if ((stats->upper & MASK_BIT(0)) == (stats->lower & MASK_BIT(0)))
        {   BitFilePutBit((stats->upper & MASK_BIT(0)) != 0, bfpOut);
            while (stats->underflowBits > 0)
            {
                BitFilePutBit((stats->upper & MASK_BIT(0)) == 0, bfpOut);
                stats->underflowBits--;
            }
        }
        else if ((stats->lower & MASK_BIT(1)) && !(stats->upper &
                MASK_BIT(1)))
        {
            stats->underflowBits += 1;
            stats->lower &= ~(MASK_BIT(0) | MASK_BIT(1));
            stats->upper |= MASK_BIT(1);
        }
        else
        {
            return ;
        }
        stats->lower <<= 1;
        stats->upper <<= 1;
        stats->upper |= 1;
    }
}
```

```
/*****************************************************************/
Function    :  WriteRemaining
Description :  This function writes out all remaining significant bits in the
               upper and lower ranges and the underflow bits once the last
               symbol has been encoded.
Parameters  :  bfpOut   -  pointer to open binary file to write to.
               stats    -  structure containing data used to encode symbols
Effects     :  Remaining significant range bits are written to the output file.
Returned    :  None
/*****************************************************************/
void WriteRemaining(bit_file_t *bfpOut, stats_t *stats)
{
    BitFilePutBit((stats->lower & MASK_BIT(1)) != 0, bfpOut);
    for (stats->underflowBits++; stats->underflowBits > 0; stats->underflowBits--)
    {   BitFilePutBit((stats->lower & MASK_BIT(1)) == 0, bfpOut); }
}

/*****************************************************************/
Function    :  ArDecodeFile
Description :  This routine opens an arithmetically encoded file, reads it's
               header, and builds a list of probability ranges which it then uses
               to decode the rest of the file.
Parameters  :  inFile       -  FILE stream to decode
               outFile      -  FILE stream to write decoded output to
               staticModel  -  TRUE if decoding with a static model
Effects     :  Encoded file is decoded
Returned    :  TRUE for success, otherwise FALSE.
/*****************************************************************/
int ArDecodeFile(FILE *inFile, FILE *outFile, char staticModel)
{
    int c;
    probability_t unscaled;
    bit_file_t *bInFile;
    stats_t stats;
    if (NULL == outFile)
    {   outFile = stdout; }
    if (NULL == inFile)
    {   fprintf(stderr, "Error: Invalid input file\n");
        return FALSE;
    }
    bInFile = MakeBitFile(inFile, BF_READ);

    if (NULL == bInFile)
    {   fprintf(stderr, "Error: Unable to create binary input file\n");
        return FALSE;
    }
```

```
    if (staticModel)
    {   if (ReadHeader(bInFile, &stats) == FALSE)
        {   BitFileClose(bInFile);
            fclose(outFile);
            return FALSE;
        }
    }
    else
    {   InitializeAdaptiveProbabilityRangeList(&stats);        }

    InitializeDecoder(bInFile, &stats);
    for (;;)
    {   unscaled = GetUnscaledCode(&stats);
        if((c = GetSymbolFromProbability(unscaled, &stats)) == -1)
        {   break;    }
        if (c == EOF_CHAR)
        {   break;    }
        fputc((char)c, outFile);
        ApplySymbolRange(c, &stats, staticModel);
        ReadEncodedBits(bInFile, &stats);
    }
    inFile = BitFileToFILE(bInFile);
    return TRUE;
}

/*****************************************************************/
Function    :  ReadHeader
Description :  This function reads the header information stored by
               WriteHeader.  The header can then be used to build a
               probability range list matching the list that was used to encode
               the file.
Parameters  :  bfpIn    -  file to read from
               Stats    -  structure containing data used to encode symbols
Effects     :  Probability range list is built.
Returned    :  TRUE for success, otherwise FALSE
*****************************************************************/
int ReadHeader(bit_file_t *bfpIn, stats_t *stats)
{
    int c;
    probability_t count;
    stats->cumulativeProb = 0;
    for (c = 0; c <= UPPER(EOF_CHAR); c++)
    {   stats->ranges[UPPER(c)] = 0; }
    for (;;)
    {   c = BitFileGetChar(bfpIn);
        count = 0;
```

```
        if (BitFileGetBitsInt(bfpIn, &count, (PRECISION - 2),
        sizeof(probability_t)) == EOF)
        {   fprintf(stderr, "Error: unexpected EOF\n");
            return FALSE;
        }
        if (count == 0)
        {       break;    }
        stats->ranges[UPPER(c)] = count;
        stats->cumulativeProb += count;
    }
    SymbolCountToProbabilityRanges(stats);
    return TRUE;
}

/*****************************************************************/
Function    :  InitializeDecoder
Description :  This function starts the upper and lower ranges at their max/min
               values and reads in the most significant encoded bits.
Parameters  :  bfpIn    -  file to read from
               stats    -  structure containing data used to encode symbols
Effects     :  upper, lower, and code are initialized.  The probability range
               list will also be initialized if an adaptive model will be used.
Returned    :  TRUE for success, otherwise FALSE
*****************************************************************/
void InitializeDecoder(bit_file_t *bfpIn, stats_t *stats)
{
    int i;
    stats->code = 0;
    for (i = 0; i < (int)PRECISION; i++)
    {   stats->code <<= 1;
        if(BitFileGetBit(bfpIn) == 1)
        {   stats->code |= 1;       }
    }
    stats->lower = 0;
    stats->upper = ~0;
}

/*****************************************************************/
Function    :  GetUnscaledCode
Description :  This function undoes the scaling that ApplySymbolRange
               performed before bits were shifted out.  The value returned is
               the probability of the encoded symbol.
Parameters  :  stats - structure containing data used to encode symbols
Effects     :  None
Returned    :  The probability of the current symbol
*****************************************************************/
```

```
probability_t GetUnscaledCode(stats_t *stats)
{
    unsigned long range;
    unsigned long unscaled;
    range = (unsigned long)(stats->upper - stats->lower) + 1;
    unscaled = (unsigned long)(stats->code - stats->lower) + 1;
    unscaled = unscaled * (unsigned long)(stats->cumulativeProb) - 1;
    unscaled /= range;
    return ((probability_t)unscaled);
}

/*****************************************************************/
Function    :  GetSymbolFromProbability
Description :  Given a probability, this function will return the symbol whose
               range includes that probability.  Symbol is found binary search
               on probability ranges.
Parameters  :  probability - probability of symbol.
               stats - structure containing data used to encode symbols
Effects     :  None
Returned    :  -1 for failure, otherwise encoded symbol
/*****************************************************************/
int GetSymbolFromProbability(probability_t probability, stats_t *stats)
{
    int first, last, middle;
    first = 0;
    last = UPPER(EOF_CHAR);
    middle = last / 2;

    while (last >= first)
    {   if (probability < stats->ranges[LOWER(middle)])
        {   last = middle - 1;
            middle = first + ((last - first) / 2);
            continue;
        }
        if (probability >= stats->ranges[UPPER(middle)])
        {   first = middle + 1;
            middle = first + ((last - first) / 2);
            continue;
        }
        return middle;
    }
    fprintf(stderr, "Unknown Symbol: %d (max: %d)\n", probability,
    stats->ranges[UPPER(EOF_CHAR)]);
    return -1;
}
```

```
/*****************************************************************/
Function    :  ReadEncodedBits
Description :  This function attempts to shift out as many code bits as possible,
               as bits are shifted out the coded input is populated with bits
               from the encoded file.  Only bits that will be unchanged when
               additional symbols are decoded may be shifted out.
               If the n most significant bits of the lower and upper range
               bounds match, they will not be changed when additional
               symbols are decoded, so they may be shifted out.
               Adjustments are also made to prevent possible underflows
               that occur when the upper and lower ranges are so close
               that decoding another symbol won't change their values.
Parameters  :  bfpOut - pointer to open binary file to read from.
               stats - structure containing data used to encode symbols
Effects     :  The upper and lower code bounds are adjusted so that they only
               contain only bits that will be affected by the addition of a new
               symbol.  Replacements are read from the encoded stream.
Returned    :  None
/*****************************************************************/
void ReadEncodedBits(bit_file_t *bfpIn, stats_t *stats)
{
    int nextBit;
    for (;;)
    {   if ((stats->upper & MASK_BIT(0)) == (stats->lower & MASK_BIT(0)))
        { }
        else if ((stats->lower & MASK_BIT(1)) && !(stats->upper &
                 MASK_BIT(1)))
        {   stats->lower   &= ~(MASK_BIT(0) | MASK_BIT(1));
            stats->upper  |= MASK_BIT(1);
            stats->code ^= MASK_BIT(1);
        }
        else
        {   return;   }
        stats->lower <<= 1;
        stats->upper <<= 1;
        stats->upper |= 1;
        stats->code <<= 1;
        if ((nextBit = BitFileGetBit(bfpIn)) == EOF)
        { }
        else
        {   stats->code |= nextBit;         }
    }
    return;
}
```

7. jbit1.h

```
/*****************************************************************/
                    Header untuk Fungsi jbit1
/*****************************************************************/
#ifndef _JBIT1_H_
#define _JBIT1_H_

/*****************************************************************/
Fungsi      :  EncodeJbit1
Deskripsi   :  Fungsi ini membaca input berupa file, menjalankan algoritma
               BWT+MTF/M1FF+algoritma hasil modifikasi kemudian
               menuliskan kodenya ke dua file output.
Parameter   :  inFile1     - Nama file input
               outFile     - Nama file output
               outFile2    - Nama file output
               MTF         - Mode MTF (MTF0 atau M1FF)
Nilai balik :  Nilai 1 jika tidak ada kesalahan, nilai 0 jika terdapat kesalahan.
/*****************************************************************/
unsigned int EncodeJbit1(char *inFile, char *outFile, char *outFile2, int MTF);

/*****************************************************************/
Fungsi      :  DecodeJbit1
Deskripsi   :  Fungsi ini membaca input berupa dua file, menjalankan
               algoritma hasil modifikasi+MTF/M1FF+BWT kemudian
               menuliskan kodenya ke file output.
Parameter   :  inFile1     - Nama file input1
               inFile2     - Nama file input2
               outFile     - Nama file output
               MTF         - Mode MTF (MTF atau M1FF)
Nilai balik :  Nilai 1 jika tidak ada kesalahan, nilai 0 jika terdapat kesalahan.
/*****************************************************************/
unsigned int DecodeJbit1(char *inFile1, char *inFile2, char *outFile, int MTF);

/*****************************************************************/
Fungsi      :  EncodeJbit0
Deskripsi   :  Fungsi ini membaca input berupa file, menjalankan algoritma
               BWT+MTF/M1FF+J-Bit Encoding,  kemudian menuliskan
               kodenya ke dua file output.
Parameter   :  inFile          - Nama file input
               outFile     - Nama file output
               outFile2    - Nama file output
               MTF         - Mode MTF (MTF atau M1FF)
Nilai balik :  Nilai 1 jika tidak ada kesalahan, nilai 0 jika terdapat kesalahan.
/*****************************************************************/
unsigned int EncodeJbit0(char *inFile, char *outFile, char *outFile2, int MTF);
```

```
/*****************************************************************/
Fungsi        :  DecodeJbit0
Deskripsi     :  Fungsi ini membaca input berupa dua file, menjalankan
                 algoritma J-Bit Encoding+MTF+BWT, kemudian menuliskan
                 kodenya ke file output.
Parameter     :  inFile1     - Nama file input1
                 inFile2     - Nama file input2
                 outFile     - Nama file output
                 MTF         - Mode MTF (MTF0 atau M1FF)
Nilai balik   :  Nilai 1 jika tidak ada kesalahan, nilai 0 jika terdapat kesalahan.
/*****************************************************************/
unsigned int DecodeJbit0(char *inFile, char *inFile2, char *outFile, int MTF);

/*****************************************************************/
Fungsi        :  Gabung
Deskripsi     :  Fungsi ini membaca input berupa dua file dan
                 menggabungkannya menjadi sebuah file output.
Parameter     :  inFile1     - Nama file input1
                 inFile2     - Nama file input2
                 outFile     - Nama file output
Nilai balik   :  Nilai 1 jika tidak ada kesalahan, nilai 0 jika terdapat kesalahan.
/*****************************************************************/
unsigned int Gabung(char *inFile1, char *inFile2, char *outFile);

/*****************************************************************/
Fungsi        :  Pisah
Deskripsi     :  Fungsi ini membaca input berupa dua file dan
                 menggabungkannya ke sebuah file output.
Parameter     :  inFile      - Nama file input
                 outFile1    - Nama file output1
                 outFile2    - Nama file output2
Nilai balik   :  Nilai 1 jika tidak ada kesalahan, nilai 0 jika terdapat kesalahan.
/*****************************************************************/
unsigned int Pisah(char *inFile, char *outFile1, char *outFile2);

#endif
```

8. jbit1.c

```
/************************ INCLUDED FILES ************************/
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>
#include "jbit1.h"
#include "bwt.c"
```

```
/************************* CONSTANTS ****************************/
#define BLOKFREK        ((1<<20))
#define BLOK1           ((1<<20))

/************************  Fungsi : EncodeJbit1************************/
unsigned int EncodeJbit1(char *inFile, char *outFile, char *outFile2, int MTF)
{
    FILE *fpIn;
    FILE *fpOut;
    FILE *fpOut2;
    unsigned char list[256];
    unsigned char *buffer1, *buffer2, *buffer3;
    unsigned int panjangfile, i, j, k, m, l;

    /* Alokasi memori */
    buffer1 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
    if ((NULL == buffer1))
    {  perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!"); return 0; }
    buffer2 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
    if ((NULL == buffer2)||(buffer1 == buffer2))
    {   free(buffer1);
        perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!"); return 0; }
    buffer3 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
    if ((NULL == buffer3)||(buffer1 == buffer3)||(buffer2 == buffer3))
    {   free(buffer1); free(buffer2);
        perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!"); return 0; }
    /* Buka file input dan output */
    if ((fpIn = fopen(inFile, "rb")) == NULL)
    {   free(buffer1); free(buffer2); free(buffer3);
        perror("Terjadi kesalahan! Tidak dapat membuka file input!"); return 0; }
    if ((fpOut = fopen(outFile, "wb")) == NULL)
    {   free(buffer1); free(buffer2); free(buffer3); fclose(fpIn);
        perror("Terjadi kesalahan! Tidak dapat membuka file output!"); return 0; }
    if ((fpOut2 = fopen(outFile2, "wb")) == NULL)
    {   free(buffer1); free(buffer2); free(buffer3);fclose(fpIn); fclose(fpOut);
        perror("Terjadi kesalahan! Tidak dapat membuka file output!"); return 0; }
        /*                              Baca dari file input                    */
    while((panjangfile = fread(buffer1, sizeof(unsigned char), BLOK1, fpIn))!= 0)
    {   printf ("\nBWT"); // Jalankan BWT
        m=BWTEncode(buffer1, panjangfile);
        //Periksa error
        if(m==0xEF1E0AA5)
        {   perror("Terjadi kesalahan! Tidak dapat menjalankan BWT!");
            free(buffer1); free(buffer2); free(buffer3);
            fclose(fpOut); fclose(fpOut2); fclose(fpIn); return 0;
        }
```

```
//Tulis bwt indeks
fwrite(&m, sizeof(int), 1, fpOut);
// Jalankan MTF/M1FF
for(i = 0; i <= UCHAR_MAX; i++)
{   list[i] = (unsigned char)i;      }
if(MTF==0)
{   printf ("-MTF");
    for (i = 0; i < panjangfile; i++)
    {   for (j = 0; j <= UCHAR_MAX; j++)
        {   if (list[j] == buffer1[i])
            {   l=(unsigned int)buffer1[i];
                buffer1[i] =(unsigned char) j; break;         }
        }
        for (; j > 0; j--)
        {   list[j] = list[j - 1]; }
        list[0] = (unsigned char) l;
    }
}
else
{   printf ("-M1FF");
    for (i = 0; i < panjangfile; i++)
    {   for (j = 0; j <= UCHAR_MAX; j++)
        {   if (list[j] == buffer1[i])
            {   l=(unsigned int)buffer1[i];
                buffer1[i] =(unsigned char) j; break;         }
        }
        if(j>1)
        {   for (; j > 1; j--) { list[j] = list[j - 1]; }
            list[1] = (unsigned char) l;
        }
        else
        {   for (; j > 0; j--) {          list[j] = list[j - 1]; }
            list[0] = (unsigned char) l;
        }
    }
}
printf ("-JBE Modifikasi"); //Jalankan JBE Modifikasi
j=0;
for (i = 0; i < panjangfile; i++)
{   if((buffer1[i]>0x01))
    {   buffer2[j]=buffer1[i]; j++; buffer1[i]=(0x02);     }
    else if((buffer1[i]==0x01))
    {   buffer1[i]=(0x01);          }
    else if((buffer1[i]==0x00))
    {   buffer1[i]=(0x00);          }
}
```

```
        fwrite(&panjangfile, sizeof(int), 1, fpOut); fwrite(&j, sizeof(int), 1, fpOut);
        fwrite(buffer2, sizeof(unsigned char), j, fpOut); k=0;
        for (i = 0; i < panjangfile; i+=4)
        {   buffer3[k]=(buffer1[i+3]<<6)|(buffer1[i+2]<<4)|(buffer1[i+1]<<2)|
            (buffer1[i]); k++;
        }
        fwrite(&k, sizeof(int), 1, fpOut2);
        fwrite(buffer3, sizeof(unsigned char), k, fpOut2);
    }
    free(buffer1); free(buffer2); free(buffer3);
    fclose(fpOut); fclose(fpOut2); fclose(fpIn);
    return 1;
}

/********************** Fungsi : DecodeJbit1 **********************/
unsigned int DecodeJbit1(char *inFile, char *inFile2, char *outFile, int MTF)
{
    FILE *fpIn;
    FILE *fpOut;
    FILE *fpIn2;
    unsigned char list[256];
    unsigned char *buffer1, *buffer2, *buffer3;
    unsigned int panjangfile, i, j, k, m, l;

    /* Alokasi memori */
    buffer1 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
    if ((NULL == buffer1))
    {   perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!"); return 0; }
    buffer2 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
    if ((NULL == buffer2)||(buffer1 == buffer2))
    {   free(buffer1);
        perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!");
        return 0;
    }
    buffer3 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
    if ((NULL == buffer3)||(buffer1 == buffer3)||(buffer2 == buffer3))
    {   free(buffer1);    free(buffer2);
        perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!");
        return 0;
    }
    /* Buka file input dan output */
    if ((fpIn = fopen(inFile, "rb")) == NULL)
    {   free(buffer1);    free(buffer2);  free(buffer3);
        perror("Terjadi kesalahan! Tidak dapat membuka file input!");
        return 0;
    }
```

```
if ((fpIn2 = fopen(inFile2, "rb")) == NULL)
{   free(buffer1);    free(buffer2);  free(buffer3);
    fclose(fpIn);  perror("Terjadi kesalahan! Tidak dapat membuka file input!");
    return 0;
}
if ((fpOut = fopen(outFile, "wb")) == NULL)
{   free(buffer1); free(buffer2); free(buffer3);
    fclose(fpIn); fclose(fpIn2);
    perror("Terjadi kesalahan! Tidak dapat membuka file output!");
    return 0;
}

/* Baca dari file input */
while((fread(&panjangfile, sizeof(int), 1, fpIn2))!= 0)
{   fread(buffer3, sizeof(unsigned char), panjangfile, fpIn2);
    printf ("\nJBE Modifikasi");
    m=0;
    for (i = 0; i < panjangfile; i++)
    {   k=buffer3[i];
        for(j=0;j<4;j++)
        {   buffer1[m]=k&0x03; m++;
            k=k>>2;
        }
    }
    fread(&m, sizeof(int), 1, fpIn);
    fread(&panjangfile, sizeof(int), 1, fpIn);
    fread(&j, sizeof(int), 1, fpIn);
    fread(buffer2, sizeof(unsigned char), j, fpIn);
    k=0;
    for (i = 0; i < panjangfile; i++)
    {   if(buffer1[i]>0x01)
        {   buffer1[i]=buffer2[k]; k++;         }
    }
    //Jalankan MTF/M1FF
    for(i = 0; i <= UCHAR_MAX; i++)
    {   list[i] = (unsigned char)i; }
    if(MTF==0)
    {   printf ("-MTF");
        for (i = 0; i < panjangfile; i++)
        {   l=(unsigned int)buffer1[i];
            buffer1[i] = list[l];
            for (j = l; j > 0; j--)
            {       list[j] = list[j - 1]; }
            list[0] = buffer1[i];
        }
    }
```

```
        else // M1FF
        {   printf ("-M1FF");
            for (i = 0; i < panjangfile; i++)
            {   l=(unsigned int)buffer1[i];
                buffer1[i] = list[l];
                if(l>1)
                {   for (j = l; j > 1; j--)
                    {   list[j] = list[j - 1]; }
                    list[1] = buffer1[i];
                }
                else
                {   for (j = l; j > 0; j--)
                    {   list[j] = list[j - 1]; }
                    list[0] = buffer1[i];
                }
            }
        }
        printf ("-BWT");//Jalankan BWT
        m=BWTDecode(buffer1, panjangfile, m);
        //Periksa error
        if(m==0xEF1E0AA5)
        {   perror("Terjadi kesalahan! Tidak dapat menjalankan BWT!");
            free(buffer1); free(buffer2); free(buffer3);
            fclose(fpOut); fclose(fpIn); fclose(fpIn2);
            return 0;
        }
        //Tulis hasil
        fwrite(buffer1, sizeof(unsigned char), panjangfile, fpOut);
    }
    free(buffer1); free(buffer2); free(buffer3);
    fclose(fpOut); fclose(fpIn); fclose(fpIn2);
    return 1;
}

/************************ Fungsi : EncodeJbit0 ************************/
unsigned int EncodeJbit0(char *inFile, char *outFile, char *outFile2, int MTF)
{
    FILE *fpIn;
    FILE *fpOut;
    FILE *fpOut2;
    unsigned char list[256];
    unsigned char *buffer1;
    unsigned char *buffer2;
    unsigned char *buffer3;
    unsigned int panjangfile, i, j, k, m, l;
```

```
/* Alokasi memori */
buffer1 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
if ((NULL == buffer1))
{   perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!"); return 0; }
buffer2 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
if ((NULL == buffer2)||(buffer1 == buffer2))
{   free(buffer1);
    perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!"); return 0;
}
buffer3 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
if ((NULL == buffer3)||(buffer1 == buffer3)||(buffer2 == buffer3))
{   free(buffer1);    free(buffer2);
    perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!"); return 0;
}
/* Buka file input dan output */
if ((fpIn = fopen(inFile, "rb")) == NULL)
{   free(buffer1);    free(buffer2);  free(buffer3);
    perror("Terjadi kesalahan! Tidak dapat membuka file input!"); return 0;
}
if ((fpOut = fopen(outFile, "wb")) == NULL)
{   free(buffer1);    free(buffer2);  free(buffer3);  fclose(fpIn);
    perror("Terjadi kesalahan! Tidak dapat membuka file output!"); return 0;
}
if ((fpOut2 = fopen(outFile2, "wb")) == NULL)
{   free(buffer1); free(buffer2); free(buffer3);
    fclose(fpIn); fclose(fpOut);
    perror("Terjadi kesalahan! Tidak dapat membuka file output!"); return 0;
}
/* Baca dari file input */
while((panjangfile = fread(buffer1, sizeof(unsigned char), BLOK1, fpIn))!= 0)
{   printf ("\nBWT");         //Jalankan BWT
    m=BWTEncode(buffer1, panjangfile);
    //Periksa error
    if(m==0xEF1E0AA5)
    {   perror("Terjadi kesalahan! Tidak dapat menjalankan BWT!");
        free(buffer1); free(buffer2); free(buffer3);
        fclose(fpOut); fclose(fpOut2); fclose(fpIn); return 0;
    }
    //Tulis bwt indeks
    fwrite(&m, sizeof(int), 1, fpOut);
    //Jalankan MTF / M1FF
    for(i = 0; i <= UCHAR_MAX; i++)
    {   list[i] = (unsigned char)i; }
    // MTF
    if(MTF==0)
    {   printf ("-MTF");
```

```
    for (i = 0; i < panjangfile; i++)
    {   for (j = 0; j <= UCHAR_MAX; j++)
        {   if (list[j] == buffer1[i])
            {   l=(unsigned int)buffer1[i];
                buffer1[i] =(unsigned char) j;
                break;
            }
        }
        for (; j > 0; j--)
        {   list[j] = list[j - 1]; }
        list[0] = (unsigned char) l;
    }
}
else // M1FF
{   printf ("-M1FF");
    for (i = 0; i < panjangfile; i++)
    {   for (j = 0; j <= UCHAR_MAX; j++)
        {   if (list[j] == buffer1[i])
            {   l=(unsigned int)buffer1[i];
                buffer1[i] =(unsigned char) j; break;
            }
        }
        if(j>1)
        {   for (; j > 1; j--)
            {   list[j] = list[j - 1]; }
            list[1] = (unsigned char) l;
        }
        else
        {   for (; j > 0; j--)
            {   list[j] = list[j - 1];    }
            list[0] = (unsigned char) l;
        }
    }
}
printf ("-JBE"); //Jalankan JBE
j=0;
for (i = 0; i < panjangfile; i++)
{   if(buffer1[i]!=0x00)
    {   buffer2[j]=buffer1[i]; j++; buffer1[i]=(0x01);    }
    else
    {   buffer1[i]=(0x00); }
}
fwrite(&panjangfile, sizeof(int), 1, fpOut);
fwrite(&j, sizeof(int), 1, fpOut);
fwrite(buffer2, sizeof(unsigned char), j, fpOut);
k=0;
```

```
        for (i = 0; i < panjangfile; i+=8)
        {   buffer3[k]=(buffer1[i+7]<<7)|(buffer1[i+6]<<6)|(buffer1[i+5]<<5)|
            (buffer1[i+4]<<4)|(buffer1[i+3]<<3)|(buffer1[i+2]<<2)|
            (buffer1[i+1]<<1)|(buffer1[i]); k++;
        }
        fwrite(&k, sizeof(int), 1, fpOut2);
        fwrite(buffer3, sizeof(unsigned char), k, fpOut2);
    }
    free(buffer1); free(buffer2); free(buffer3);
    fclose(fpOut); fclose(fpIn); fclose(fpOut2);
    return 1;
}

/************************ Fungsi : DecodeJbit0 ************************/
unsigned int DecodeJbit0(char *inFile, char *inFile2, char *outFile, int MTF)
{
    FILE *fpIn;
    FILE *fpOut;
    FILE *fpIn2;
    unsigned char list[256];
    unsigned char *buffer1;
    unsigned char *buffer2;
    unsigned char *buffer3;
    unsigned int panjangfile, i, j, k, m, l;

    /* Alokasi memori */
    buffer1 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
    if ((NULL == buffer1))
    {   perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!"); return 0; }
    buffer2 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
    if ((NULL == buffer2)||(buffer1 == buffer2))
    {   free(buffer1);
        perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!"); return 0;
    }
    buffer3 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
    if ((NULL == buffer3)||(buffer1 == buffer3)||(buffer2 == buffer3))
    {   free(buffer1); free(buffer2);
        perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!");
        return 0;
    }
    /* Buka file input dan output */
    if ((fpIn = fopen(inFile, "rb")) == NULL)
    {   free(buffer1); free(buffer2); free(buffer3);
        perror("Terjadi kesalahan! Tidak dapat membuka file input!");
        return 0;
    }
```

```
if ((fpIn2 = fopen(inFile2, "rb")) == NULL)
{   free(buffer1); free(buffer2); free(buffer3); fclose(fpIn);
    perror("Terjadi kesalahan! Tidak dapat membuka file input!"); return 0;
}
if ((fpOut = fopen(outFile, "wb")) == NULL)
{   free(buffer1);    free(buffer2);  free(buffer3); fclose(fpIn); fclose(fpIn2);
    perror("Terjadi kesalahan! Tidak dapat membuka file output!"); return 0;
}
/* Baca dari file input */
while((fread(&panjangfile, sizeof(int), 1, fpIn2))!= 0)
{   fread(buffer3, sizeof(unsigned char), panjangfile, fpIn2);
    printf ("\nJBE"); //Jalankan JBIT
    m=0;
    for (i = 0; i < panjangfile; i++)
    {   k=buffer3[i];
        for(j=0;j<8;j++)
        {   buffer1[m]=k&0x01; m++;
            k=k>>1;   }
    }
    fread(&m, sizeof(int), 1, fpIn);
    fread(&panjangfile, sizeof(int), 1, fpIn);
    fread(&j, sizeof(int), 1, fpIn);
    fread(buffer2, sizeof(unsigned char), j, fpIn);
    k=0;
    for (i = 0; i < panjangfile; i++)
    {   if(buffer1[i]!=0x00)
        {   buffer1[i]=buffer2[k]; k++; }
    }
    //Jalankan MTF / M1FF
    for(i = 0; i <= UCHAR_MAX; i++)
    {   list[i] = (unsigned char)i; }
    if(MTF==0)
    {   printf ("-MTF");
        for (i = 0; i < panjangfile; i++)
        {   l=(unsigned int)buffer1[i];
            buffer1[i] = list[l];
            for (j = l; j > 0; j--)
            {   list[j] = list[j - 1];       }
            list[0] = buffer1[i];
        }
    }
    else // MTF1
    {   printf ("-M1FF");
        for (i = 0; i < panjangfile; i++)
        {   l=(unsigned int)buffer1[i];
```

```
                buffer1[i] = list[l];
                if(l>1)
                {   for (j = l; j > 1; j--)
                    {   list[j] = list[j - 1]; }
                    list[1] = buffer1[i];
                }
                else
                {   for (j = l; j > 0; j--)
                    {   list[j] = list[j - 1]; }
                    list[0] = buffer1[i];
                }
            }
        }
        printf ("-BWT"); //Jalankan BWT
        m=BWTDecode(buffer1, panjangfile, m);
        //Periksa error
        if(m==0xEF1E0AA5)
        {   perror("Terjadi kesalahan! Tidak dapat menjalankan BWT!");
            free(buffer1); free(buffer2); free(buffer3);
            fclose(fpOut); fclose(fpIn); fclose(fpIn2); return 0;
        }
        //Tulis hasil
        fwrite(buffer1, sizeof(unsigned char), panjangfile, fpOut);
    }
    free(buffer1); free(buffer2); free(buffer3);
    fclose(fpOut); fclose(fpIn); fclose(fpIn2);
    return 1;
}

/************************* Fungsi : Gabung ***************************/
unsigned int Gabung(char *inFile1, char *inFile2, char *outFile)
{
    FILE *fpIn1;
    FILE *fpOut;
    FILE *fpIn2;
    unsigned char *buffer1;
    unsigned int panjangfile,i,j;
    /* Alokasi memori */
    buffer1 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
    if ((NULL == buffer1))
    {   perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!"); return 0; }
    /* Buka file input dan output */
    if ((fpIn1 = fopen(inFile1, "rb")) == NULL)
    {   free(buffer1);
        perror("Terjadi kesalahan! Tidak dapat membuka file input!"); return 0;
    }
```

```
    if ((fpIn2 = fopen(inFile2, "rb")) == NULL)
    {   free(buffer1);     fclose(fpIn1);
        perror("Terjadi kesalahan! Tidak dapat membuka file input!"); return 0;
    }
    if ((fpOut = fopen(outFile, "wb")) == NULL)
    {   free(buffer1); fclose(fpIn1); fclose(fpIn2);
        perror("Terjadi kesalahan! Tidak dapat membuka file output!"); return 0;
    }
    /* Baca dari file input1 */
    fseek( fpIn1, 0, SEEK_END );
    panjangfile = ftell( fpIn1 );
    fwrite(&panjangfile, sizeof(int), 1, fpOut); rewind(fpIn1);
    for(i=0;i<panjangfile;i++)
    {   j = fgetc(fpIn1); fputc(j,fpOut);  }
    /* Baca dari file input2 */
    fseek( fpIn2, 0, SEEK_END );
    panjangfile = ftell( fpIn2 );
    fwrite(&panjangfile, sizeof(int), 1, fpOut);
    rewind(fpIn2);
    for(i=0;i<panjangfile;i++)
    {   j = fgetc(fpIn2); fputc(j,fpOut); }
    free(buffer1); fclose(fpOut); fclose(fpIn1); fclose(fpIn2);
    return 1;
}

/************************** Fungsi : Pisah **************************/
unsigned int Pisah(char *inFile, char *outFile1, char *outFile2)
{
    FILE *fpIn;
    FILE *fpOut1;
    FILE *fpOut2;
    unsigned char *buffer1;
    unsigned int panjangfile,i,j;
    /* Alokasi memori */
    buffer1 = (unsigned char *)malloc((BLOKFREK) * sizeof(unsigned char));
    if ((NULL == buffer1))
    { perror("Terjadi kesalahan! Tidak dapat mengalokasi memori!"); return 0; }
    /* Buka file input dan output */
    if ((fpIn = fopen(inFile, "rb")) == NULL)
    {  free(buffer1);
        perror("Terjadi kesalahan! Tidak dapat membuka file input!"); return 0;
    }
    if ((fpOut1 = fopen(outFile1, "wb")) == NULL)
    {   free(buffer1);     fclose(fpIn);
        perror("Terjadi kesalahan! Tidak dapat membuka file output!"); return 0;
    }
```

```
    if ((fpOut2 = fopen(outFile2, "wb")) == NULL)
    {   free(buffer1);    fclose(fpIn);    fclose(fpOut1);
        perror("Terjadi kesalahan! Tidak dapat membuka file output!"); return 0;
    }
    /* Baca dari file input */
    fread(&panjangfile, sizeof(int), 1, fpIn);
    for(i=0;i<panjangfile;i++)
    {   j = fgetc(fpIn); fputc(j,fpOut1); }
    fread(&panjangfile, sizeof(int), 1, fpIn);
    for(i=0;i<panjangfile;i++)
    { j = fgetc(fpIn); fputc(j,fpOut2); }
    free(buffer1); fclose(fpIn); fclose(fpOut1); fclose(fpOut2); return 1;
}
```

9. mainprogram.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "arcode.c"
#include "jbit1.c"

int main(int argc, char *argv[])
{
    FILE *fpIn;
    FILE *fpOut;
    char nama_fileinput[81];
    char nama_fileoutput[81];
    int pil,c,i,j;
    unsigned long in_file_len = 0, out_file_len = 0;
    float ratio;
    do
    {   system ("cls");
        printf("/*************************************************/\n");
        printf ("                    Aplikasi Kompresi Data\n");
        printf("/*************************************************/\n");
        printf ("Pilih jenis kompresi :\n");
printf ("1. Kompresi  dengan skema BWT + MTF + JBE + Aritmatika\n");
printf ("2. Kompresi  dengan skema BWT + MTF + JBE(mod) + Aritmatika\n");
printf ("3. Kompresi  dengan skema BWT + M1FF + JBE + Aritmatika\n");
printf ("4. Kompresi  dengan skema BWT + M1FF + JBE(mod) + Aritmatika\n");
printf ("5. Dekompresi dengan skema BWT + MTF + JBE     + Aritmatika\n");
printf ("6. Dekompresi dengan skema BWT + MTF + JBE(mod) + Aritmatika\n");
printf ("7. Dekompresi dengan skema BWT + M1FF + JBE     + Aritmatika\n");
printf ("8. Dekompresi dengan skema BWT + M1FF + JBE(mod)+Aritmatika\n");
printf ("0. Keluar\n");
```

```
printf ("\nMasukkan pilihan: ");
scanf("%d",&pil);fflush(stdin);
switch (pil)
{
/************************* 0. Keluar *************************/
case 0:  return EXIT_SUCCESS; break;

/*************************************************************/
                1. Kompresi BWT+MTF+JBE+Aritmatika
/*************************************************************/
case 1:
    /* Dapatkan nama file input dan output dari user */
    printf ("\nMasukan nama file input:");
    scanf("%[^\n]s",&nama_fileinput);
    strcpy(nama_fileoutput,nama_fileinput);
    strcat(nama_fileoutput,".zip");
    /* Buka file input dan output */
    if ((fpIn = fopen(nama_fileinput, "rb")) == NULL)
    { printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");
        return 0;   }
    fseek( fpIn, 0, SEEK_END );
    in_file_len = ftell( fpIn );
    fclose(fpIn);

/*************** Kompresi dengan BWT+MTF+JBE ***************/
if (EncodeJbit0(nama_fileinput, "out1.tmp", "out2.tmp", 0)==1)
{   printf("\n\nKompresi dengan algoritma BWT+MTF+JBE berhasil
    dilakukan.\n");
}
else
{   printf("\nTerjadi kesalahan! Kompresi dengan algoritma
    BWT+MTF+JBE tidak berhasil dilakukan.\n"); return EXIT_FAILURE;
}

/*************************************************************/
                Kompresi dengan Pengkodean Aritmatika
/*************************************************************/
/* Buka file input dan output */
if ((fpIn = fopen("out1.tmp", "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return 0; }
if ((fpOut = fopen("out11.tmp", "wb")) == NULL)
{   fclose(fpIn);
    printf("\nTerjadi kesalahan! Tidak dapat membuka file output!");
    return 0;
}
```

```
if (ArEncodeFile (fpIn,fpOut,'1')==1)
{   printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n"); return EXIT_FAILURE;
}
fclose(fpIn); fclose(fpOut);
/* Buka file input dan output */
if ((fpIn = fopen("out2.tmp", "rb")) == NULL)
{printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return 0; }
if ((fpOut = fopen("out12.tmp", "wb")) == NULL)
{   fclose(fpIn);
    printf("\nTerjadi kesalahan! Tidak dapat membuka file output!");return 0;
}
if (ArEncodeFile (fpIn,fpOut,'1')==1)
{   printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n"); return EXIT_FAILURE;
}
fclose(fpIn); fclose(fpOut);

/********************** Penggabungan File **********************/
if (Gabung("out11.tmp", "out12.tmp", nama_fileoutput)==1)
{   printf("\nPenggabungan berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Penggabungan tidak berhasil dilakukan.\n");
    return EXIT_FAILURE; }
/* Buka file input dan output */
if ((fpIn = fopen(nama_fileoutput, "rb")) == NULL)
{   printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");
    return 0; }
fseek( fpIn, 0, SEEK_END ); out_file_len = ftell( fpIn ); fclose(fpIn);
j=remove("out1.tmp"); j=remove("out11.tmp");
j=remove("out2.tmp"); j=remove("out12.tmp");
fprintf(stderr, "\n\nNama file input      : %s", nama_fileinput );
fprintf(stderr, "\nNama file output     : %s", nama_fileoutput );
fprintf(stderr, "\nPanjang file input   = %15lu byte", in_file_len );
fprintf(stderr, "\nPanjang file output  = %15lu byte", out_file_len );
ratio = (((float) in_file_len - (float) out_file_len)/(float) in_file_len ) *
          (float) 100;
fprintf(stderr, "\nCompression ratio    = %15.2f %%\n", ratio );
break;
```

```
/*************************************************************/
            2. Kompresi BWT+MTF+JBE (modifikasi)+Aritmatika
/*************************************************************/
case 2:
/* Dapatkan nama file input dan output dari user */
printf ("\nMasukan nama file input:");
scanf("%[^\n]s",&nama_fileinput);
strcpy(nama_fileoutput,nama_fileinput);
strcat(nama_fileoutput,".zip");
/* Buka file input dan output */
if ((fpIn = fopen(nama_fileinput, "rb")) == NULL)
{printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");return 0;}
fseek( fpIn, 0, SEEK_END );in_file_len = ftell( fpIn );fclose(fpIn);

/********** Kompresi dengan BWT+MTF+JBE(modifikasi) **********/
if (EncodeJbit1(nama_fileinput, "out1.tmp", "out2.tmp", 0)==1)
{   printf("\n\nKompresi dengan algoritma BWT+MTF+JBE(modifikasi)
    berhasil dilakukan.\n");
}
else
{   printf("\nTerjadi kesalahan! Kompresi dengan algoritma
    BWT+MTF+JBE(modifikasi) tidak berhasil dilakukan.\n");
    return EXIT_FAILURE;
}

/*************************************************************/
                Kompresi dengan Pengkodean Aritmatika
/*************************************************************/
/* Buka file input dan output */
if ((fpIn = fopen("out1.tmp", "rb")) == NULL)
{   printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");
    return 0; }
if ((fpOut = fopen("out11.tmp", "wb")) == NULL)
{   fclose(fpIn);
    printf("\nTerjadi kesalahan! Tidak dapat membuka file output!");
    return 0; }
if (ArEncodeFile (fpIn,fpOut,'1')==1)
{   printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n");         return 0; }
fclose(fpIn); fclose(fpOut);
/* Buka file input dan output */
if ((fpIn = fopen("out2.tmp", "rb")) == NULL)
{   printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");
    return 0; }
```

```
if ((fpOut = fopen("out12.tmp", "wb")) == NULL)
{   fclose(fpIn); printf("\nTerjadi kesalahan! Tidak dapat membuka file
    output!"); return 0; }
if (ArEncodeFile (fpIn,fpOut,'1')==1)
{   printf("\nPengkodean Aritmatika berhasil dilakukan.\n");   }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n");         return 0; }
fclose(fpIn);       fclose(fpOut);

/********************** Penggabungan File **********************/
if (Gabung("out11.tmp", "out12.tmp", nama_fileoutput)==1)
{   printf("\nPenggabungan berhasil dilakukan.\n");     }
else
{   printf("\nTerjadi kesalahan! Penggabungan tidak berhasil dilakukan.\n");
    return 0; }
/* Buka file input dan output */
if ((fpIn = fopen(nama_fileoutput, "rb")) == NULL)
{   printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");
    return 0; }
fseek( fpIn, 0, SEEK_END ); out_file_len = ftell( fpIn ); fclose(fpIn);
j=remove("out1.tmp");  j=remove("out11.tmp");
j=remove("out2.tmp");  j=remove("out12.tmp");
fprintf(stderr, "\n\nNama file input      : %s", nama_fileinput );
fprintf(stderr, "\nNama file output     : %s", nama_fileoutput );
fprintf(stderr, "\nPanjang file input   = %15lu byte", in_file_len );
fprintf(stderr, "\nPanjang file output  = %15lu byte", out_file_len );
ratio = (((float) in_file_len - (float) out_file_len) / (float) in_file_len ) *
          (float) 100;
fprintf(stderr, "\nCompression ratio    = %15.2f %%\n", ratio );
break;

/*************************************************************/
                  3. Kompresi BWT+M1FF+JBE+Aritmatika
/*************************************************************/
case 3:
/* Dapatkan nama file input dan output dari user */
printf ("\nMasukan nama file input:");
scanf("%[^\n]s",&nama_fileinput);
strcpy(nama_fileoutput,nama_fileinput);
strcat(nama_fileoutput,".zip");
/* Buka file input dan output */
if ((fpIn = fopen(nama_fileinput, "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return 0;}
fseek( fpIn, 0, SEEK_END ); in_file_len = ftell( fpIn ); fclose(fpIn);
```

```
/*************** Kompresi dengan BWT+M1FF+JBE **************/
if (EncodeJbit0(nama_fileinput, "out1.tmp", "out2.tmp", 1)==1)
{   printf("\n\nKompresi dengan algoritma BWT+M1FF+JBE berhasil
    dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Kompresi dengan algoritma
    BWT+M1FF+JBE tidak berhasil dilakukan.\n");return EXIT_FAILURE;
}

/************* Kompresi dengan Pengkodean Aritmatika *************/
if ((fpIn = fopen("out1.tmp", "rb")) == NULL)
{printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");return 0;}
if ((fpOut = fopen("out11.tmp", "wb")) == NULL)
{   fclose(fpIn); printf("\nTerjadi kesalahan! Tidak dapat membuka file
    output!"); return 0;
}
if (ArEncodeFile (fpIn,fpOut,'1')==1)
{printf("\nPengkodean Aritmatika berhasil dilakukan.\n");}
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n"); return EXIT_FAILURE;
}
fclose(fpIn); fclose(fpOut);

/* Buka file input dan output */
if ((fpIn = fopen("out2.tmp", "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return 0;}
if ((fpOut = fopen("out12.tmp", "wb")) == NULL)
{   fclose(fpIn); printf("\nTerjadi kesalahan! Tidak dapat membuka file
    output!"); return 0;
}
if (ArEncodeFile (fpIn,fpOut,'1')==1)
{   printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n");         return EXIT_FAILURE; }
fclose(fpIn); fclose(fpOut);

/********************** Penggabungan File **********************/
if (Gabung("out11.tmp", "out12.tmp", nama_fileoutput)==1)
{ printf("\nPenggabungan berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Penggabungan tidak berhasil dilakukan.\n");
    return EXIT_FAILURE;
}
```

```
/* Buka file input dan output */
if ((fpIn = fopen(nama_fileoutput, "rb")) == NULL)
{   printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");
    return 0;
}
fseek( fpIn, 0, SEEK_END ); out_file_len = ftell( fpIn ); fclose(fpIn);
j=remove("out1.tmp"); j=remove("out11.tmp");
j=remove("out2.tmp"); j=remove("out12.tmp");
fprintf(stderr, "\n\nNama file input      : %s", nama_fileinput );
fprintf(stderr, "\nNama file output     : %s", nama_fileoutput );
fprintf(stderr, "\nPanjang file input   = %15lu byte", in_file_len );
fprintf(stderr, "\nPanjang file output  = %15lu byte", out_file_len );
ratio = (((float) in_file_len - (float) out_file_len) / (float) in_file_len ) *
         (float) 100;
fprintf(stderr, "\nCompression ratio    = %15.2f %%\n", ratio );
break;

/*************************************************************/
            4. Kompresi BWT+M1FF+JBE (modifikasi)+Aritmatika
/*************************************************************/
case 4:
/* Dapatkan nama file input dan output dari user */
printf ("\nMasukan nama file input:");
scanf("%[^\n]s",&nama_fileinput);
strcpy(nama_fileoutput,nama_fileinput);
strcat(nama_fileoutput,".zip");
/* Buka file input dan output */
if ((fpIn = fopen(nama_fileinput, "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return 0;}
fseek( fpIn, 0, SEEK_END ); in_file_len = ftell( fpIn ); fclose(fpIn);

/********* Kompresi dengan BWT+M1FF+JBE(modifikasi) **********/
if (EncodeJbit1(nama_fileinput, "out1.tmp", "out2.tmp", 1)==1)
{   printf("\n\nKompresi dengan algoritma BWT+M1FF+JBE(modifikasi)
    berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Kompresi dengan algoritma
    BWT+M1FF+JBE(modifikasi) tidak berhasil dilakukan.\n");
    return EXIT_FAILURE;}

/************* Kompresi dengan Pengkodean Aritmatika ************/
if ((fpIn = fopen("out1.tmp", "rb")) == NULL)
{printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return  0;}
if ((fpOut = fopen("out11.tmp", "wb")) == NULL)
{   fclose(fpIn);  printf("\nTerjadi kesalahan! Tidak dapat membuka file
    output!");return 0;}
```

```
if (ArEncodeFile (fpIn,fpOut,'1')==1)
{ printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n"); return EXIT_FAILURE; }
fclose(fpIn); fclose(fpOut);
/* Buka file input dan output */
if ((fpIn = fopen("out2.tmp", "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return 0;}
if ((fpOut = fopen("out12.tmp", "wb")) == NULL)
{   fclose(fpIn); printf("\nTerjadi kesalahan! Tidak dapat membuka file
    output!"); return 0; }
if (ArEncodeFile (fpIn,fpOut,'1')==1)
{ printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n"); return EXIT_FAILURE; }
fclose(fpIn); fclose(fpOut);

/********************** Penggabungan File *********************/
if (Gabung("out11.tmp", "out12.tmp", nama_fileoutput)==1)
{ printf("\nPenggabungan berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Penggabungan tidak berhasil dilakukan.\n");
    return EXIT_FAILURE; }
/* Buka file input dan output */
if ((fpIn = fopen(nama_fileoutput, "rb")) == NULL)
{   printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");return 0;}
fseek( fpIn, 0, SEEK_END );out_file_len = ftell( fpIn );fclose(fpIn);
j=remove("out1.tmp");j=remove("out11.tmp");
j=remove("out2.tmp");j=remove("out12.tmp");
fprintf(stderr, "\n\nNama file input      : %s", nama_fileinput );
fprintf(stderr, "\nNama file output     : %s", nama_fileoutput );
fprintf(stderr, "\nPanjang file input   = %15lu byte", in_file_len );
fprintf(stderr, "\nPanjang file output  = %15lu byte", out_file_len );
ratio = (((float) in_file_len - (float) out_file_len)/ (float) in_file_len ) *
          (float) 100;
fprintf(stderr, "\nCompression ratio    = %15.2f %%\n", ratio );
break;

/*************************************************************/
                5. Dekompresi BWT+MTF+JBE+Aritmatika
/*************************************************************/
case 5:
/* Dapatkan nama file input dan output dari user */
printf ("\nMasukan nama file input:");
```

```
scanf("%[^\n]s",&nama_fileinput);
in_file_len=strlen(nama_fileinput);
in_file_len-=4;
strncpy(nama_fileoutput,nama_fileinput,in_file_len);

/************************ Pemisahan File ************************/
if (Pisah(nama_fileinput,"out11.tmp", "out12.tmp")==1)
{   printf("\nPemisahan berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pemisahan tidak berhasil dilakukan.\n");
    return EXIT_FAILURE; }

/************ Dekompresi dengan Pengkodean Aritmatika ************/
if ((fpIn = fopen("out11.tmp", "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");return 0;}
if ((fpOut = fopen("out1.tmp", "wb")) == NULL)
{   fclose(fpIn);  printf("\nTerjadi kesalahan! Tidak dapat membuka file
    output!");return 0; }
if (ArDecodeFile(fpIn,fpOut,'1')==1)
{   printf("\nPengkodean Aritmatika berhasil dilakukan.\n");   }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n"); return EXIT_FAILURE;
}
fclose(fpIn); fclose(fpOut);
/* Buka file input dan output */
if ((fpIn = fopen("out12.tmp", "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return 0;}
if ((fpOut = fopen("out2.tmp", "wb")) == NULL)
{   fclose(fpIn);  printf("\nTerjadi kesalahan! Tidak dapat membuka file
    output!"); return 0;
}
if (ArDecodeFile(fpIn,fpOut,'1')==1)
{ printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n"); return EXIT_FAILURE;
}
fclose(fpIn);fclose(fpOut);

/*************** Dekompresi dengan BWT+MTF+JBE **************/
if (DecodeJbit0("out1.tmp", "out2.tmp", nama_fileoutput,0)==1)
{   printf("\n\nDekompresi dengan algoritma BWT+MTF+JBE berhasil
    dilakukan.\n");
    printf("\nNama file hasil dekompresi adalah %s\n",nama_fileoutput);
}
```

```
else
{   printf("\nTerjadi kesalahan! Dekompresi dengan algoritma
    BWT+MTF+JBE tidak berhasil dilakukan.\n"); return EXIT_FAILURE;
}
j=remove("out1.tmp"); j=remove("out11.tmp");
j=remove("out2.tmp"); j=remove("out12.tmp");
break;

/*************************************************************/
          6. Dekompresi BWT+MTF+JBE (modifikasi)+Aritmatika
/*************************************************************/
case 6:
/* Dapatkan nama file input dan output dari user */
printf ("\nMasukan nama file input:");
scanf("%[^\n]s",&nama_fileinput);
in_file_len=strlen(nama_fileinput);
in_file_len-=4;
strncpy(nama_fileoutput,nama_fileinput,in_file_len);

/************************ Pemisahan File ************************/
if (Pisah(nama_fileinput,"out11.tmp", "out12.tmp")==1)
{ printf("\nPemisahan berhasil dilakukan.\n"); }
else
{ printf("\nTerjadi kesalahan! Pemisahan tidak berhasil dilakukan.\n");
  return EXIT_FAILURE; }

/************* Dekompresi dengan Pengkodean Aritmatika ************/
if ((fpIn = fopen("out11.tmp", "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return 0;}

if ((fpOut = fopen("out1.tmp", "wb")) == NULL)
{   fclose(fpIn); printf("\nTerjadi kesalahan! Tidak dapat membuka file
    output!"); return 0; }

if (ArDecodeFile(fpIn,fpOut,'1')==1)
{ printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n"); return EXIT_FAILURE; }
fclose(fpIn);
fclose(fpOut);

/* Buka file input dan output */
if ((fpIn = fopen("out12.tmp", "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return 0;}
```

```
if ((fpOut = fopen("out2.tmp", "wb")) == NULL)
{   fclose(fpIn); printf("\nTerjadi kesalahan! Tidak dapat membuka file
    output!"); return 0; }

if (ArDecodeFile(fpIn,fpOut,'1')==1)
{ printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n");
    return EXIT_FAILURE;
}
fclose(fpIn);
fclose(fpOut);

/********* Dekompresi dengan BWT+MTF+JBE (modifikasi) *********/
if (DecodeJbit1("out1.tmp", "out2.tmp", nama_fileoutput,0)==1)
{   printf("\n\nDekompresi dengan algoritma BWT+MTF+JBE (modifikasi)
    berhasil dilakukan.\n");
    printf("\nNama file hasil dekompresi adalah %s\n",nama_fileoutput);
}
else
{   printf("\nTerjadi kesalahan! Dekompresi dengan algoritma
    BWT+MTF+JBE (modifikasi) tidak berhasil dilakukan.\n");
    return EXIT_FAILURE;
}
j=remove("out1.tmp"); j=remove("out11.tmp");
j=remove("out2.tmp"); j=remove("out12.tmp");
break;

/*****************************************************************/
                7. Dekompresi BWT+MTF+JBE+Aritmatika
/*****************************************************************/
case 7:
/* Dapatkan nama file input dan output dari user */
printf ("\nMasukan nama file input:");
scanf("%[^\n]s",&nama_fileinput);
in_file_len=strlen(nama_fileinput);
in_file_len-=4;
strncpy(nama_fileoutput,nama_fileinput,in_file_len);

/********************** Pemisahan File **********************/
if (Pisah(nama_fileinput,"out11.tmp", "out12.tmp")==1)
{ printf("\nPemisahan berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pemisahan tidak berhasil dilakukan.\n");
    return EXIT_FAILURE; }
```

```
/************ Dekompresi dengan Pengkodean Aritmatika ***********/
if ((fpIn = fopen("out11.tmp", "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");return 0;}

if ((fpOut = fopen("out1.tmp", "wb")) == NULL)
{
    fclose(fpIn);
    printf("\nTerjadi kesalahan! Tidak dapat membuka file output!");
    return 0; }

if (ArDecodeFile(fpIn,fpOut,'1')==1)
{   printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n"); return EXIT_FAILURE; }
fclose(fpIn); fclose(fpOut);

/* Buka file input dan output */
if ((fpIn = fopen("out12.tmp", "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return 0;}

if ((fpOut = fopen("out2.tmp", "wb")) == NULL)
{   fclose(fpIn); printf("\nTerjadi kesalahan! Tidak dapat membuka file
    output!"); return 0; }

if (ArDecodeFile(fpIn,fpOut,'1')==1)
{   printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n"); return EXIT_FAILURE; }
fclose(fpIn); fclose(fpOut);

/************** Dekompresi dengan BWT+M1FF+JBE *************/
if (DecodeJbit0("out1.tmp", "out2.tmp", nama_fileoutput,1)==1)
{   printf("\n\nDekompresi dengan algoritma BWT+M1FF+JBE berhasil
    dilakukan.\n");
    printf("\nNama file hasil dekompresi adalah %s\n",nama_fileoutput);
}
else
{   printf("\nTerjadi kesalahan! Dekompresi dengan algoritma
    BWT+M1FF+JBE tidak berhasil dilakukan.\n");
    return EXIT_FAILURE;
}
j=remove("out1.tmp"); j=remove("out11.tmp");
j=remove("out2.tmp"); j=remove("out12.tmp");
break;
```

```
/*************************************************************/
        8. Dekompresi BWT+M1FF+JBE (modifikasi)+Aritmatika
/*************************************************************/
case 8:
/* Dapatkan nama file input dan output dari user */
printf ("\nMasukan nama file input:");
scanf("%[^\n]s",&nama_fileinput);
in_file_len=strlen(nama_fileinput);
in_file_len-=4;
strncpy(nama_fileoutput,nama_fileinput,in_file_len);

/********************** Pemisahan File **********************/
if (Pisah(nama_fileinput,"out11.tmp", "out12.tmp")==1)
{ printf("\nPemisahan berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pemisahan tidak berhasil dilakukan.\n");
    return EXIT_FAILURE; }

/************* Dekompresi dengan Pengkodean Aritmatika ***********/
if ((fpIn = fopen("out11.tmp", "rb")) == NULL)
{ printf("\nTerjadi kesalahan! Tidak dapat membuka file input!"); return 0;}

if ((fpOut = fopen("out1.tmp", "wb")) == NULL)
{   fclose(fpIn);
    printf("\nTerjadi kesalahan! Tidak dapat membuka file output!");
    return 0; }

if (ArDecodeFile(fpIn,fpOut,'1')==1)
{ printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
else
{   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
    dilakukan.\n"); return EXIT_FAILURE;
}
fclose(fpIn); fclose(fpOut);

if ((fpIn = fopen("out12.tmp", "rb")) == NULL)
{   printf("\nTerjadi kesalahan! Tidak dapat membuka file input!");
    return 0;
}

if ((fpOut = fopen("out2.tmp", "wb")) == NULL)
{   fclose(fpIn);
    printf("\nTerjadi kesalahan! Tidak dapat membuka file output!");
    return 0;
}
```

```
        if (ArDecodeFile(fpIn,fpOut,'1')==1)
        { printf("\nPengkodean Aritmatika berhasil dilakukan.\n"); }
        else
        {   printf("\nTerjadi kesalahan! Pengkodean Aritmatika  tidak berhasil
            dilakukan.\n");
            return EXIT_FAILURE;
        }
        fclose(fpIn);
        fclose(fpOut);

        /********* Dekompresi dengan BWT+M1FF+JBE (modifikasi) ********/
        if (DecodeJbit1("out1.tmp", "out2.tmp", nama_fileoutput,1)==1)
        {   printf("\n\nDekompresi dengan algoritma BWT+M1FF+JBE
            (modifikasi) berhasil dilakukan.\n"); printf("\nNama file hasil
            dekompresi adalah %s\n",nama_fileoutput);
        }
        else
        {   printf("\nTerjadi kesalahan! Dekompresi dengan algoritma
            BWT+M1FF+JBE (modifikasi) tidak berhasil dilakukan.\n");
            return EXIT_FAILURE;
        }
        j=remove("out1.tmp");
         j=remove("out11.tmp");
        j=remove("out2.tmp");
        j=remove("out12.tmp");
        break;

        /*Pilihan selain 1-8*/
        default: printf("\nTerjadi kesalahan! Pilihan anda tidak dikenali!\n");
        }

        printf ("\nInput '0' jika ingin keluar! ");
        scanf("%d",&pil);fflush(stdin);
    }while(pil != 0);

    return EXIT_SUCCESS;
}
```