```
//----------------------------------------------------------------------------
//
//  ise.h
//
//  Authors: Joe Jarchow, Geoffrey Griffith, Joseph Kadhim, Shinya Daigaku
//           Andrew Pouzeshi
//
//  Sponsor: Tom Lookabaugh, Assistant Professor of Computer Science
//           University of Colorado
//
//  Senior Project: Team ISE (Image Selective Encryption)
//                  December 2003
//
//  For more information go to: http://128.138.75.184
//----------------------------------------------------------------------------
//
//  This code is open source and may be used with no cost.
//  The authors are in no way responsible for any effects
//  from the usage of this code.  It is provided as is with
//  no warranties, protections, promises or any form of
//  support.  The authors would hope it would only be used
//  for good purposes.  Thank you.
//
//----------------------------------------------------------------------------
//
//  The purpose of this file is to define what functions and members
//  are to be exported for a programmer using the ISE class.  ISE
//  is a class defined to implement image selective encryption for
//  jpeg images.  class ise is intended to be the super class and
//  must be inherited by sub classes.  We have only implemented the
//  jpeg_ise class at this time but other classes could be implemented
//  following the outline used.  Along with constructors there are
//  are various functions for setting and getting the class members
//  each is defined in detail preceeding the appropriate function
//  in the ise.cpp file.
//
//----------------------------------------------------------------------------
#include <stdlib.h>
#include <iostream>
#include <fstream>
using std::ifstream;
using std::ofstream;

#ifndef ISE_H
#define ISE_H
    class ise
    {
        public:
                        ise(char*, char* = NULL, char* = NULL);
                        virtual ~ise();
                        virtual int encrypt_file() { return 0; }
                        virtual int decrypt_file() { return 0; }
                        int set_key(char*);
                        int set_input_file_name(char*);
                        int set_output_file_name(char*);
                        char* get_input_file_name();
                        char* get_output_file_name();
                protected:
                        ise();
                        int get_ise_file_type(char*);
                        int make_ise_file_name();
                        int make_output_file_name();
                        char* get_key();
            private:
                        char* input_file_name;
                        char* output_file_name;
                        char* key;
    };
#endif //ISE_H
```

```
#ifndef JPEG_ISE_H
#define JPEG_ISE_H
    class jpeg_ise : public ise
    {
        public:
                        jpeg_ise(char*, char* = NULL, char* = NULL);
                        ~jpeg_ise();
                        int encrypt_file();
                        int decrypt_file();
        protected:
                        jpeg_ise();
    };
#endif //JPEG_ISE_H
```

```
//----------------------------------------------------------------------------
//
//  ise.cpp
//
//  Authors: Joe Jarchow, Geoffrey Griffith, Joseph Kadhim, Shinya Daigaku
//           Andrew Pouzeshi
//
//  Sponsor: Tom Lookabaugh, Assistant Professor of Computer Science
//           University of Colorado
//
//  Senior Project: Team ISE (Image Selective Encryption)
//                  December 2003
//
//  For more information go to: http://128.138.75.184
//----------------------------------------------------------------------------
//
//  This code is open source and may be used with no cost.
//  The authors are in no way responsible for any effects
//  from the usage of this code.  It is provided as is with
//  no warranties, protections, promises or any form of
//  support.  The authors would hope it would only be used
//  for good purposes.  Thank you.
//
//----------------------------------------------------------------------------
//
//  The purpose of this file is to define what functions and members
//  are to be exported for a programmer using the ISE class.  ISE
//  is a class defined to implement image selective encryption for
//  jpeg images.  class ise is intended to be the super class and
//  must be inherited by sub classes.  We have only implemented the
//  jpeg_ise class at this time but other classes could be implemented
//  following the outline used.  Along with constructors there are
//  are various functions for setting and getting the class members
//  each is defined in detail preceeding the appropriate function
//  in the ise.cpp file.
//
//----------------------------------------------------------------------------

#include <stdlib.h>
#include <string>
#include <iostream>
#include <stack>
#include <cstdlib>
#include "rijndael-api-fst.h" // use for block cipher encryption/decryption

using namespace std;
using std::cerr;
using std::endl;
using std::nothrow;

const int  JPEG_TYPE      = 1;     // specify jpeg ise
const char JPEG_FILE_TYPE = ’1’;   // specify jpeg file type
const unsigned int  MIN_KEY_LENGTH = 32;    // minimum length of the key
const int  BUFFER_LENGTH  = 16;    // size of Rijndael encryption block

typedef unsigned char byte;

#include "ise.h"

//----------------------------------------------------------------------------
//
// Default Constructor
// Pre-conditions:      None.
// Post-conditions: None.
// Parameters:          None.
// Return values:       Constructor, no return type.
// Description:     Default constructor is not used by users.
//
//----------------------------------------------------------------------------
```

```
ise::ise()
{
}

//----------------------------------------------------------------------------
//
// Overloaded Constructor
// Pre-conditions:      The key must be a pointer to a character string.
// Post-conditions: An ISE object is created containing the specified
//                  data members.
// Parameters:      The first argument is a pointer to the key.
//                  The second argument is the name and path of the input file
//                  to be encrypted or decrypted.  The third argument is
//                  the file name and path for the output file generated by
//                  encryption or decryption.
// Return values:       Constructor, no return type.
// Description:     An ISE object is constructed with the data necessary to
//                  encrypt or decrypt a file.  This overloaded
//                  constructor only requires that the first argument
//                  be provided.  The second and third arguments are optional
//                  and will be set to a default value of NULL.
//
//----------------------------------------------------------------------------
ise::ise(char* key, char* input_file_name, char* output_file_name)
{
        size_t length;
        char * key_copy;
        char * temp;

        // check that the key in not NULL
        if (key == NULL)
        {
                exit(1);
        }

        // check that the input and output files are of type jpeg or ise
        char * index;
        if (input_file_name != NULL)
        {
                index = strstr(input_file_name, ".jp");
                if (index == NULL)
                {
                        index = strstr(input_file_name, ".JP");
                        if (index == NULL)
                        {
                                index = strstr(input_file_name, ".ise");
                                if (index == NULL)
                                {
                                        index = strstr(input_file_name, ".ISE");
                                        if (index == NULL)
                                        {
                                                exit(1);
                                        }
                                }
                        }
                }
        }

        if (output_file_name != NULL)
        {
                index = strstr(output_file_name, ".jp");
                if (index == NULL)
                {
                        index = strstr(output_file_name, ".JP");
                        if (index == NULL)
                        {
                                index = strstr(output_file_name, ".ise");
                                if (index == NULL)
                                {
                                        index = strstr(output_file_name, ".ISE");
                                        if (index == NULL)
                                        {
                                                exit(1);
                                        }
                                }
                        }
                }
        }

        // set the key
        length = strlen(key);
        key_copy = new (nothrow) char [length + 1];
        if (key_copy == NULL)
        {
                exit(1);
        }
        temp = new (nothrow) char [length * 2 + 1];
        if (temp == NULL)
        {
                exit(1);
        }
        strcpy(key_copy, key);

        // split each character into four bit values
        for (size_t i = 0; i < length; i++)
        {
            temp[i * 2] = key_copy[i] >> 4;
            key_copy[i] = key_copy[i] << 4;
            temp[i * 2 + 1] = key_copy[i] >> 4;
        }

        // convert four bit values to hexadecimal characters
        length = length * 2;
        temp[length] = ’\0’;
        for (size_t i = 0; i < length; i++)
        {
                switch((int)temp[i])
                {
                case 0:
                        temp[i] = ’0’;
                        break;
                case 1:
                        temp[i] = ’1’;
                        break;
                case 2:
                        temp[i] = ’2’;
                        break;
                case 3:
                        temp[i] = ’3’;
                        break;
                case 4:
                        temp[i] = ’4’;
                        break;
                case 5:
                        temp[i] = ’5’;
                        break;
                case 6:
                        temp[i] = ’6’;
                        break;
                case 7:
                        temp[i] = ’7’;
                        break;
                case -8:
                        temp[i] = ’8’;
                        break;
                case -7:
                        temp[i] = ’9’;
                        break;
```

```
                case -6:
                        temp[i] = ’a’;
                        break;
                case -5:
                        temp[i] = ’b’;
                        break;
                case -4:
                        temp[i] = ’c’;
                        break;
                case -3:
                        temp[i] = ’d’;
                        break;
                case -2:
                        temp[i] = ’e’;
                        break;
                case -1:
                        temp[i] = ’f’;
                        break;
                }
        }

        // extend the key length to 32 bytes
        if (length < MIN_KEY_LENGTH)
        {
                this->key = new (nothrow) char[MIN_KEY_LENGTH + 1];
                if (this->key == NULL)
                {
                        exit(1);
                }
                strcpy(this->key, temp);
                for (size_t i = length; i < MIN_KEY_LENGTH; i++)
                {
                        this->key[i] = ’0’;
                }
                this->key[MIN_KEY_LENGTH] = ’\0’;
        }
        else
        {
                this->key = new (nothrow) char[length + 1];
                if (this->key == NULL)
                {
                        exit(1);
                }
                strcpy(this->key, temp);
        }
        delete [] key_copy;
        delete [] temp;

        // set the input file name
        if (input_file_name != NULL)
        {
                length = strlen(input_file_name);
                this->input_file_name = new (nothrow) char[length + 1];
                if (this->input_file_name == NULL)
                {
                        exit(1);
                }
                strcpy(this->input_file_name, input_file_name);
        }
        else
        {
                this->input_file_name = NULL;
        }

        // set the output file name
        if (output_file_name != NULL)
        {
                length = strlen(output_file_name);
                this->output_file_name = new (nothrow) char[length + 1];
```

```cpp
                if (this->output_file_name == NULL)
                {
                        exit(1);
                }
                strcpy(this->output_file_name, output_file_name);
        }
        else
        {
                this->output_file_name = NULL;
        }
}

ise::~ise()
{
        if (key != NULL)
        {
                delete [] key;
        }
        if (input_file_name != NULL)
        {
                delete [] input_file_name;
        }
        if (output_file_name != NULL)
        {
                delete [] output_file_name;
        }
}

//----------------------------------------------------------------------------
//
// Pre-conditions:  The key must point to a character string.
// Post-conditions:     The key will be set using the new string specified.
//                  Any previous information in key will be lost.
// Parameters:      The only argument to this method is a pointer to
//                  a character string containing the key information
//                  for either encryption or decryption.
// Return values:       An integer is returned indicating a success or failure.
//                  A zero will indicate a success.
//                  A one will indicate an invalid key.
//                                      A two will indicate a memory allocation
error.
// Description:     The method will use the specified character string to
//                  create a valid key to be used by the encryption or
//                  decryption methods.
//
//----------------------------------------------------------------------------
int ise::set_key(char* name)
{
        size_t length;
        char * name_copy;
        char * temp;

        // check that the key is not NULL
        if (name == NULL)
        {
                return 1;
        }

        length = strlen(name);
        name_copy = new (nothrow) char[length + 1];
        if (name_copy == NULL)
        {
                return 2;
        }
        temp = new (nothrow) char[length * 2 + 1];
        if (temp == NULL)
        {
                return 2;
        }
```

```cpp
        strcpy(name_copy, name);

        // split each character into four bit values.
        for (size_t i = 0; i < length; i++)
        {
            temp[i * 2] = name_copy[i] >> 4;
            name_copy[i] = name_copy[i] << 4;
            temp[i * 2 + 1] = name_copy[i] >> 4;
        }

        length = length * 2;
        temp[length] = '\0';

        // convert four bit values to hexadecimal characters
        for (size_t i = 0; i < length; i++)
          {
            switch((int)temp[i])
              {
              case 0:
                temp[i] = '0';
                break;
              case 1:
                temp[i] = '1';
                break;
              case 2:
                temp[i] = '2';
                break;
              case 3:
                temp[i] = '3';
                break;
              case 4:
                temp[i] = '4';
                break;
              case 5:
                temp[i] = '5';
                break;
              case 6:
                temp[i] = '6';
                break;
              case 7:
                temp[i] = '7';
                break;
              case -8:
                temp[i] = '8';
                break;
              case -7:
                temp[i] = '9';
                break;
              case -6:
                temp[i] = 'a';
                break;
              case -5:
                temp[i] = 'b';
                break;
              case -4:
                temp[i] = 'c';
                break;
              case -3:
                temp[i] = 'd';
                break;
              case -2:
                temp[i] = 'e';
                break;
              case -1:
                temp[i] = 'f';
                break;
              }
          }
```

```
    // delete the previous key information
        delete [] key;

        // extend the key length to 32 bytes
        if (length < MIN_KEY_LENGTH)
        {
                key = new (nothrow) char[MIN_KEY_LENGTH + 1];
                if (key == NULL)
                {
                        return 2;
                }
                strcpy(key, temp);
                for (size_t i = length; i < MIN_KEY_LENGTH; i++)
                {
                        key[i] = '0';
                }
                key[MIN_KEY_LENGTH] = '\0';
        }
        else
        {
                key = new (nothrow) char[length + 1];
                if (key == NULL)
                {
                        return 2;
                }
                strcpy(key, temp);
        }

        delete [] name_copy;
        delete [] temp;

        return 0;
}

//-----------------------------------------------------------------------------
//
// Pre-conditions:      The name must be a pointer to a valid jpeg or ise file
//                  type.
// Post-conditions:     The input_file_name will be set using the new string
//                  specified.  Any previous data in input_file_name will
//                  be lost.
// Parameters:      The only argument to this method is a pointer to a
//                  character string containing the input_file_name,
//                  specifying the input file to encryption or decryption.
// Return values:   An integer is returned indicating a success or failure.
//                  A zero will indicate a success.
//                  A one will indicate an invalid input file name.
//                                      A two will indicate a memory allocation
error.
// Description:     This method is used to set the input_file_name.
//                  The method must be called prior to the encryption
//                  or decryption methods if they were not specified
//                  in the constructor.
//
//-----------------------------------------------------------------------------
int ise::set_input_file_name(char* name)
{
        size_t length;

        // check that the name is not NULL
        if (name == NULL)
        {
                return 1;
        }

        // check that the name is a jpeg or ise file type
        char * index;
        index = strstr(name, ".jp");
        if (index == NULL)
```

```
        {
                index = strstr(name, ".JP");
                if (index == NULL)
                {
                        index = strstr(name, ".ise");
                        if (index == NULL)
                        {
                                index = strstr(name, ".ISE");
                                if (index == NULL)
                                {
                                        return 1;
                                }
                        }
                }
        }

        // delete any previous input file information
        if (input_file_name != NULL)
        {
                delete [] input_file_name;
        }

        // set the input file name
        length = strlen(name);
        input_file_name = new (nothrow) char[length + 1];
        if (input_file_name == NULL)
        {
                return 2;
        }
        strcpy(input_file_name, name);

        return 0;
}

//-----------------------------------------------------------------------------
//
// Pre-conditions:      The name must be a pointer to a valid jpeg or ise file
//                  type.
// Post-conditions:     The output_file_name will be set using the new string
//                  specified.  Any previous data in output_file_name will
//                  be lost.
// Parameters:      The only argument to this method is a pointer to a
//                  character string containing the output_file_name,
//                  specifying the output file to encryption or decryption.
// Return values:   An integer is returned indicating a success or failure.
//                  A zero will indicate a success.
//                  A one will indicate an invalid output file name.
//                                      A two will indicate a memory allocation
error.
// Description:     This method is used to set the output_file_name.
//
//-----------------------------------------------------------------------------
int ise::set_output_file_name(char* name)
{
        size_t length;

        // check that the name is not NULL
        if (name == NULL)
        {
                return 1;
        }

        // check that the name is a jpeg or ise file type
        char * index;
        index = strstr(name, ".jp");
        if (index == NULL)
        {
                index = strstr(name, ".JP");
                if (index == NULL)
                {
                        index = strstr(name, ".ise");
                        if (index == NULL)
                        {
                                index = strstr(name, ".ISE");
                                if (index == NULL)
                                {
                                        return 1;
                                }
                        }
                }
        }

        // delete any previous output file information
        if (output_file_name != NULL)
        {
                delete [] output_file_name;
        }

        // set the output file name
        length = strlen(name);
        output_file_name = new (nothrow) char[length + 1];
        if (output_file_name == NULL)
        {
                return 2;
        }
        strcpy(output_file_name, name);

        return 0;
}

//----------------------------------------------------------------------------
//
// Pre-conditions:      None.
// Post-conditions:     None.
// Parameters:      None.
// Return values:   The method will return the input_file_name character string.
//                  If the input_file_name is not set, the method will return
//                  NULL.
// Description:     This is the accessor method for the input file name.
//
//----------------------------------------------------------------------------
char* ise::get_input_file_name()
{
        // check that the input file is not NULL
        if (input_file_name == NULL)
        {
                return NULL;
        }
        return input_file_name;
}

//----------------------------------------------------------------------------
//
// Pre-conditions:      None.
// Post-conditions:     None.
// Parameters:      None.
// Return values:   The method will return the output_file_name character string
.
//                  If the output_file_name is not set, the method will return
//                  NULL.
// Description:     This is the accessor method for the output file name.
//
//----------------------------------------------------------------------------
char* ise::get_output_file_name()
{
        // check that the output file is not NULL
        if (output_file_name == NULL)
        {
```

```
                return NULL;
        }
        return output_file_name;
}

//----------------------------------------------------------------------------
//
// Pre-conditions:      The name must be a pointer to a valid ISE file.
// Post-conditions:     None
// Parameters:      The only argument for this method is a pointer
//                  to a character string indicating the name of a
//                  valid ISE file.
// Return values:   The function will return an integer indicating
//                  the type of the original file from which the specified
//                  ISE file was created.
//                  0 will indicate an unknown or unimplemented file type.
//                  1 will indicate a jpeg file.
//                  2 will indicate a mp3 file.
//                  3 will indicate a zip file.
//                  The return values may be extended to accommodate other file
types.
// Description:     This method will return an integer corresponding to
//                  the original file type of an encrypted ISE file.
//
//----------------------------------------------------------------------------
int ise::get_ise_file_type(char* name)
{
        char the_type;

        ifstream ise_infs(name, ios::binary);

        // check that the file can be opened
        if (ise_infs.good() == false)
        {
                return 0;
        }
        // read the first byte from the ise file
        ise_infs.read(&the_type, sizeof(the_type));

        // check if the file is a jpeg ise
        if (the_type == '1')
        {
                return 1;
        }
        // check if the file is a mp3 ise
        if (the_type == '2')
        {
                return 2;
        }
        // check if the file is a zip ise
        if (the_type == '3')
        {
                return 3;
        }

        ise_infs.close();

        // otherwise the file is unknown
        return 0;
}

//----------------------------------------------------------------------------
//
// Pre-conditions:      The user of the class has previously set the input_file_
name.
// Post-conditions: The output_file_name data member points to a string with
//                  a file name and file path, based upon the string pointed
//                  to by the input_file_name.
// Parameters:      None.
// Return values:   An integer is returned indicating a success or failure.
//                  A zero will indicate a success.
//                  A one will indicate a failure.
// Description:     The file name and path created will be the same as the
//                  string pointed to by the input_file_name data member,
//                  except that the extension of the file will be changed
//                  to .ise.  If this file already exists, then a 0 will be
//                  added on to the end of the file name, just before the
//                  extension.  If this file already exists, we will keep
//                  incrementing this number and checking, until the new file
//                  name does not previously exist.
//
//----------------------------------------------------------------------------
int ise::make_ise_file_name()
{
        char* index;
        // size equals length of extention number
        size_t length, size;
        // used to find the name extention number
        int number, temp, remainder, count, digit;
        char letter = ’0’;
        // stores name extention number
        stack<int> file_index;
        ifstream InFile;

        // set an ise file name from the input file name
        number = 0;
        length = strlen(input_file_name);
        output_file_name = new (nothrow) char[length + 1];
        if (output_file_name == NULL)
        {
                return 1;
        }
        strcpy(output_file_name, input_file_name);
        // check the jpeg file extention
        index = strstr(output_file_name, ".jp");
        // if file extention is ".JPG"
        if (index == NULL)
        {
                index = strstr(output_file_name, ".JP");
        }
        // check if not a jpeg file
        if (index == NULL)
        {
                return 1;
        }

        // add ise extention
        *(index+1) = ’i’;
        *(index+2) = ’s’;
        *(index+3) = ’e’;
        *(index+4) = ’\0’;

        InFile.open(output_file_name);

        // if file name already exists, make a new file name
        while (InFile.good())
        {
                InFile.close();
                number++;
                temp = number;
                // calculate name extention number
                while (temp != 0)
                {
                        remainder = temp % 10;
                        file_index.push(remainder);
                        temp = temp / 10;
                }
```

```
                // create output file name
                if (output_file_name != NULL)
                {
                        delete [] output_file_name;
                }
                size = file_index.size();
                output_file_name = new (nothrow) char[length + size + 1];
                if (output_file_name == NULL)
                {
                        return 1;
                }
                strcpy(output_file_name, input_file_name);
                index = strstr(output_file_name, ".jp");
                // if file extention is ".JPG"
                if (index == NULL)
                {
                        index = strstr(output_file_name, ".JP");
                }
                count = 0;

                // convert top of stack to a character
                while (!file_index.empty())
                {
                        digit = file_index.top();
                        file_index.pop();
                        switch (digit)
                        {
                        case 0:
                                letter = ’0’;
                                break;
                        case 1:
                                letter = ’1’;
                                break;
                        case 2:
                                letter = ’2’;
                                break;
                        case 3:
                                letter = ’3’;
                                break;
                        case 4:
                                letter = ’4’;
                                break;
                        case 5:
                                letter = ’5’;
                                break;
                        case 6:
                                letter = ’6’;
                                break;
                        case 7:
                                letter = ’7’;
                                break;
                        case 8:
                                letter = ’8’;
                                break;
                        case 9:
                                letter = ’9’;
                                break;
                        }
                        // add extention number
                        *(index + count) = letter;
                        count++;
                }
                // add ise file extention
                *(index + size) = ’.’;
                *(index + size + 1) = ’i’;
                *(index + size + 2) = ’s’;
                *(index + size + 3) = ’e’;
                *(index + size + 4) = ’\0’;
                InFile.open(output_file_name);
        }
        return 0;
}

//-----------------------------------------------------------------------------
//
// Pre-conditions:      The user of the class has previously set the input_file_
name.
// Post-conditions: The output_file_name data member points to a string with
//                  a file name and file path, based upon the string pointed
//                  to by the input_file_name.
// Parameters:      None.
// Return values:   An integer is returned indicating a success or failure.
//                  A zero will indicate a success.
//                  A one will indicate a failure.
// Description:     The file name and path created will be the same as the
//                  string pointed to by the input_file_name data member,
//                  except that the extension of the file will be changed
//                  to .jpg.  If this file already exists, then a 0 will be
//                  added on to the end of the file name, just before the
//                  extension.  If this file already exists, we will keep
//                  incrementing this number and checking, until the new file
//                  name does not previously exist.
//
//-----------------------------------------------------------------------------
int ise::make_output_file_name()
{
        char* index;
        // size equals length of extention number
        size_t length, size;
        // used to find the name extention number
        int number, temp, remainder, count, digit;
        char letter = ’0’;
        // stores name extention number
        stack<int> file_index;
        ifstream InFile;

        // set an output file name from the ise file name
        number = 0;
        length = strlen(input_file_name);
        output_file_name = new (nothrow) char[length + 1];
        if (output_file_name == NULL)
        {
                return 1;
        }
        strcpy(output_file_name, input_file_name);
        // check the ise file extention
        index = strstr(output_file_name, ".is");
        // check if the extention is .ISE
        if (index == NULL)
        {
                index = strstr(input_file_name, ".IS");
        }
        // check if not a valid ise file
        if (index == NULL)
        {
                return 1;
        }
        // add jpeg extention
        *(index+1) = ’j’;
        *(index+2) = ’p’;
        *(index+3) = ’g’;
        *(index+4) = ’\0’;

        InFile.open(output_file_name);

        // if file name already exists, make a new file name
        while (InFile.good())
```

```
        {
                InFile.close();
                number++;
                temp = number;
                // calculate name extention number
                while (temp != 0)
                {
                        remainder = temp % 10;
                        file_index.push(remainder);
                        temp = temp / 10;
                }

                // create output file name
                if (output_file_name != NULL)
                {
                        delete [] output_file_name;
                }
                size = file_index.size();
                output_file_name = new (nothrow) char[length + size + 1];
                if (output_file_name == NULL)
                {
                        return 1;
                }
                strcpy(output_file_name, input_file_name);
                index = strstr(output_file_name, ".is");
                // check if file extention is ".ISE"
                if (index == NULL)
                {
                        index = strstr(input_file_name, ".IS");
                }

                // index offset
                count = 0;

                // convert top of stack to a character
                while (!file_index.empty())
                {
                        digit = file_index.top();
                        file_index.pop();
                        switch (digit)
                        {
                        case 0:
                                letter = ’0’;
                                break;
                        case 1:
                                letter = ’1’;
                                break;
                        case 2:
                                letter = ’2’;
                                break;
                        case 3:
                                letter = ’3’;
                                break;
                        case 4:
                                letter = ’4’;
                                break;
                        case 5:
                                letter = ’5’;
                                break;
                        case 6:
                                letter = ’6’;
                                break;
                        case 7:
                                letter = ’7’;
                                break;
                        case 8:
                                letter = ’8’;
                                break;
                        case 9:
                                letter = '9';
                                break;
                        }
                        // add extention number
                        *(index + count) = letter;
                        count++;
                }
                // add jpeg extetion
                *(index + size) = '.';
                *(index + size + 1) = 'j';
                *(index + size + 2) = 'p';
                *(index + size + 3) = 'g';
                *(index + size + 4) = '\0';

                InFile.open(output_file_name);
        }
        return 0;
}

//-----------------------------------------------------------------------------
//
// Pre-conditions:      None.
// Post-conditions:     None.
// Parameters:      None.
// Return values:   The method will return the key character string.
//                  If the key is not set, the method will return
//                  NULL.
// Description:     This is the accessor method for the key.
//
//-----------------------------------------------------------------------------
char* ise::get_key()
{
        // check that the key is not NULL
        if (key == NULL)
        {
                return NULL;
        }
        return key;
}

//-----------------------------------------------------------------------------
//
// Default Constructor
// Pre-conditions:      None.
// Post-conditions:     None.
// Parameters:          None.
// Return values:       Constructor, no return type.
// Description:     Default constructor is not used by users.
//
//-----------------------------------------------------------------------------
jpeg_ise::jpeg_ise() : ise()
{
}

//-----------------------------------------------------------------------------
//
// Overloaded Constructor
// Pre-conditions:      The key must be a pointer to a character string.
// Post-conditions: An JPEG_ISE object is created containing the specified
//                  data members.
// Parameters:      The first argument is a pointer to the key.
//                  The second argument is the name and path of the input file
//                  to be encrypted or decrypted.  The third argument is
//                  the file name and path for the output file generated by
//                  encryption or decryption.
// Return values:       Constructor, no return type.
// Description:     An ISE object is constructed with the data necessary to
//                  encrypt or decrypt a file.  This overloaded
//                  constructor only requires that the first argument
```

```
//                  be provided.  The second and third arguments are optional
//                  and will be set to a default value of NULL.
//
//-----------------------------------------------------------------------------
jpeg_ise::jpeg_ise(char* key, char* input_file_name, char* output_file_name)
: ise(key, input_file_name, output_file_name)
{
}

jpeg_ise::~jpeg_ise()
{
}

//-----------------------------------------------------------------------------
//
// Pre-conditions:      The input_file_name and key must be set using either
//                  the overloaded constructor or the
//                  set_input_file_name(char* name) and set_key(char* key)
//                  functions prior to calling this method.
//                  This code requires that the input and ouput file pointers
//                  are at the head of the file.
// Post-conditions: An encrypted file will be created with the name and path
//                  specified by the output_file_name data
//                  member.  If this data member is NULL, then a default file
//                  name will be created based upon the input_file_name
//                  data member.
// Parameters:      None.
// Return values:   An integer is returned indicating a success or failure.
//                  A zero will indicate a success.
//                  A one will indicate could not open input file name
//                  A two will indicate could not create ise file name
//                  A three will indicate could not open ise file
//                                      A four will indicate the jpeg file is no
t baseline
// Description:     The encrypt_file method will take a standard baseline
//                  compression JPEG file and selectively encrypt the
//                  Huffman Table frames found within the file.
//                  If the file already exists, the existing file will
//                  be overwritten.  A new, encrypted file will be
//                  created for the selectively encrypted JPEG image.
//
//-----------------------------------------------------------------------------
int jpeg_ise::encrypt_file()
{
        // check if the input file exists
        ifstream infs(jpeg_ise::get_input_file_name(), ios::binary);
    if (infs.good() == false)
    {
                return 1;
    }

        // Check if ise_file_name is empty
        if (jpeg_ise::get_output_file_name() == NULL)
        {
                // create the ise output file
                jpeg_ise::make_ise_file_name();
                if (jpeg_ise::get_output_file_name() == NULL)
                {
                        return 2;
                }
        }

        // check if output file can open
    ofstream outfs(jpeg_ise::get_output_file_name(), ios::binary);
    if (outfs.good() == false)
    {
                return 3;
    }
```

```
   //output jpeg identifier to head of file
   char file_type;
   file_type = JPEG_FILE_TYPE;
   outfs.write(&file_type,sizeof(file_type));

   bool encrypt_huffman_table, encrypt_encoded_data;
   encrypt_huffman_table = encrypt_encoded_data = false;

   bool ff,inhuff,stop_encrypt, is_baseline, is_ffda;
   ff = inhuff = stop_encrypt = false;
   is_baseline = is_ffda = false;        //check if file contains FFC0, FFC4 or
FFDA
   int keyLength = 128;
   unsigned char plain_text[BUFFER_LENGTH];
   memset(plain_text,0,BUFFER_LENGTH);
   unsigned char cipher_text[BUFFER_LENGTH];
   memset(cipher_text,0,BUFFER_LENGTH);
   char cipher_text_output[BUFFER_LENGTH];
   memset(cipher_text_output,0,BUFFER_LENGTH);
   keyInstance keyinst;
   cipherInstance cipherinst;
       makeKey(&keyinst, DIR_ENCRYPT, keyLength, jpeg_ise::get_key());
   char iv[BUFFER_LENGTH];
   memset(iv,0,BUFFER_LENGTH);
   cipherInit(&cipherinst, MODE_ECB, iv);

   int pt_counter = 0;

   char b,c;
       // begin the ise selective encryption algorithm
   while (infs.read(&b,sizeof(b)))
       {
               // send unencrypted data to output file
       if (inhuff == false && stop_encrypt == false)
       {
           if ((byte)b == 0xFF)
           {
               outfs.write(&b,sizeof(b));
               infs.read(&b,sizeof(b));
               if ((byte)b == 0xC4 || (byte)b == 0xC0 )
               {
                                       // begin encrypting
                   inhuff = true;
                   is_baseline = true;
               }
               // non baseline jpeg marker
               else if ((byte)b == 0xC1 || (byte)b == 0xC2 || (byte)b == 0xC3 |
|
                        (byte)b == 0xC5 || (byte)b == 0xC6 || (byte)b == 0xC7 |
|
                        (byte)b == 0xC8 || (byte)b == 0xC9 || (byte)b == 0xCA |
|
                        (byte)b == 0xCB || (byte)b == 0xCC || (byte)b == 0xCD |
|
                        (byte)b == 0xCE || (byte)b == 0xCF)
                 {
                   return 4;
                 }
           }
           outfs.write(&b,sizeof(b));
       }

               // fill last buffer to be encrypted
       else if (inhuff == false && stop_encrypt == true)
       {
                       // fill last encryption buffer
           while (pt_counter < BUFFER_LENGTH)
           {
               plain_text[pt_counter++] = b;
```

```
                if(pt_counter < BUFFER_LENGTH) infs.read(&b,sizeof(b));
            }

                        // encrypt the buffer
            blockEncrypt(&cipherinst,&keyinst,plain_text,keyLength,cipher_text);
            // send encrypted data to output file
                        for (int i = 0; i < BUFFER_LENGTH; i++)
            {
                cipher_text_output[i]=(char)cipher_text[i];
                outfs.write(&cipher_text_output[i],sizeof(cipher_text_output[i])
);
            }
                        // reset the buffer
            memset(plain_text,0,BUFFER_LENGTH);
            memset(cipher_text,0,BUFFER_LENGTH);
            memset(cipher_text_output,0,BUFFER_LENGTH);
            pt_counter = 0;
                        // done encrypting
            stop_encrypt = false;
        }

                // encrypt huffman data of input file
        else
        {
                        // look for the begining of jpeg marker
            if ((byte)b == 0xFF)
            {
                infs.read(&c,sizeof(c));
                                // look for the non huffman marker
                if ((byte)c == 0xDA)
                {
                                        // go to fill last buffer
                    inhuff = false;
                    stop_encrypt = true;
                    is_ffda = true;
                }
                // check if file contains non baseline marker while encrypting
                if ((byte)c == 0xC1 || (byte)c == 0xC2 || (byte)c == 0xC3 ||
                    (byte)c == 0xC5 || (byte)c == 0xC6 || (byte)c == 0xC7 ||
                    (byte)c == 0xC8 || (byte)c == 0xC9 || (byte)c == 0xCA ||
                    (byte)c == 0xCB || (byte)c == 0xCC || (byte)c == 0xCD ||
                    (byte)c == 0xCE || (byte)c == 0xCF)
                  {
                    return 4;
                  }
                                // if huffman marker found, continue encryption
                if (pt_counter < BUFFER_LENGTH)
                {
                                        // add to the buffer
                    plain_text[pt_counter++] = b;
                }

                                // if huffman marker found and buffer is full, c
ontinue encryption
                else
                {
                                        // encrypt
                    blockEncrypt(&cipherinst,&keyinst,plain_text,keyLength,ciphe
r_text);
                    for (int i = 0; i < BUFFER_LENGTH; i++)
                    {
                                                // send to output file
                        cipher_text_output[i]=(char)cipher_text[i];
                        outfs.write(&cipher_text_output[i],sizeof(cipher_text_ou
tput[i]));
                    }
                                        // reset the buffer
                    memset(plain_text,0,BUFFER_LENGTH);
                    memset(cipher_text,0,BUFFER_LENGTH);
                    memset(cipher_text_output,0,BUFFER_LENGTH);
                    pt_counter = 0;
                    plain_text[pt_counter++] = b;
                }
                                // continue filling buffer
                if (pt_counter < BUFFER_LENGTH)
                {
                    plain_text[pt_counter++] = c;
                                        // encrypt if the buffer is full
                                        if(pt_counter == BUFFER_LENGTH)
                                        {
                                                blockEncrypt(&cipherinst,&keyins
t,plain_text,keyLength,cipher_text);
                                                for (int i = 0; i < BUFFER_LENGT
H; i++)
                                                {
                                                        // send to output file
                                                        cipher_text_output[i]=(c
har)cipher_text[i];
                                                        outfs.write(&cipher_text
_output[i],sizeof(cipher_text_output[i]));
                                                }
                                                // reset the buffer
                                                memset(plain_text,0,BUFFER_LENGT
H);
                                                memset(cipher_text,0,BUFFER_LENG
TH);
                                                memset(cipher_text_output,0,BUFF
ER_LENGTH);
                                                pt_counter = 0;
                                                stop_encrypt = false;
                                        }
                }

                                // if the buffer is full, encrypt and add c to b
uffer
                else
                {
                                        // encrypt
                    blockEncrypt(&cipherinst,&keyinst,plain_text,keyLength,ciphe
r_text);
                    for (int i = 0; i < BUFFER_LENGTH; i++)
                    {
                                                // send to output file
                        cipher_text_output[i]=(char)cipher_text[i];
                        outfs.write(&cipher_text_output[i],sizeof(cipher_text_ou
tput[i]));
                    }
                                        // reset the buffer
                    memset(plain_text,0,BUFFER_LENGTH);
                    memset(cipher_text,0,BUFFER_LENGTH);
                    memset(cipher_text_output,0,BUFFER_LENGTH);
                    pt_counter = 0;
                                        // add second half of marker to new buff
er
                    plain_text[pt_counter++] = c;
                }
            }

                        // if no jpeg marker, fill a buffer and encrypt
            else
            {
                                // continue to fill buffer
                if (pt_counter < BUFFER_LENGTH)
                {
                    plain_text[pt_counter++] = b;
                }
                                // encrypt if the buffer is full
                else
```

```
                {
                    blockEncrypt(&cipherinst,&keyinst,plain_text,keyLength,ciphe
r_text);
                    for (int i = 0; i < BUFFER_LENGTH; i++)
                    {
                                                // send to output file
                        cipher_text_output[i]= (char)cipher_text[i];
                        outfs.write(&cipher_text_output[i],sizeof(cipher_text_ou
tput[i]));
                    }
                    // reset the buffer
                    memset(plain_text,0,BUFFER_LENGTH);
                    memset(cipher_text,0,BUFFER_LENGTH);
                    memset(cipher_text_output,0,BUFFER_LENGTH);
                    pt_counter = 0;
                                        // add b to new buffer
                    plain_text[pt_counter++] = b;
                }
            }
        }
        }

    if (!is_baseline || !is_ffda)
      {
        return 4;
      }

    infs.close();
    outfs.close();

    return 0;
}

//-------------------------------------------------------------------------------
//
// Pre-conditions:      The input_file_name and key must be set using either
//                  the overloaded constructor or the
//                  set_input_file_name(char* name) and set_key(char* key)
//                  functions prior to calling this method.
//                  This code requires that the input and ouput file pointers
//                  are at the head of the file.
// Post-conditions: An decrypted file will be created with the name and path
//                  specified by the output_file_name data
//                  member.  If this data member is NULL, then a default file
//                  name will be created based upon the input_file_name
//                  data member.
// Parameters:      None.
// Return values:   An integer is returned indicating a success or failure.
//                  A zero will indicate a success.
//                  A one will indicate input file is not a jpeg ise file
//                  A two will indicate could not open ise file
//                  A three will indicate could not create output jpeg file
//                  A four will indicate could not open output jpeg file
// Description:     The decrypt_file method will take a JPEG ise file and
//                  selectively decrypt the Huffman Table frames found
//                  within the file.
//                  If the file already exists, the existing file will
//                  be overwritten.  A new, encrypted file will be
//                  created for the selectively new decrypted JPEG image.
//
//-------------------------------------------------------------------------------
int jpeg_ise::decrypt_file()
{
        // check if input file is not a jpeg ise file
        if (jpeg_ise::get_ise_file_type(jpeg_ise::get_input_file_name()) != JPEG
_TYPE)
    {
                return 1;
    }
```

```
        ifstream infs(jpeg_ise::get_input_file_name(), ios::binary);

        // check if input file could not open
    if (infs.good() == false)
    {
                return 2;
    }

    // check if ise_file_name is NULL
    if (jpeg_ise::get_output_file_name() == NULL)
    {
                // create output jpeg file
                jpeg_ise::make_output_file_name();
                if (jpeg_ise::get_output_file_name() == NULL)
                {
                        return 3;
                }
    }

        // check if output file could not open
    ofstream outfs(jpeg_ise::get_output_file_name(), ios::binary);
    if (outfs.good() == false)
    {
                return 4;
    }

    //output jpeg identifier to head of file
    char file_type;
    infs.read(&file_type,sizeof(file_type));
        // check if file type of ise is
    /*if (file_type != ’1’)
    {
        return 1;
    }*/

    bool decrypt_huffman_table, decrypt_encoded_data;
    decrypt_huffman_table = decrypt_encoded_data = false;

    bool ff,inhuff,split_block;
    ff = inhuff = split_block = false;
    int keyLength = 128;
    unsigned char plain_text[BUFFER_LENGTH];
    memset(plain_text,0,BUFFER_LENGTH);
    unsigned char cipher_text[BUFFER_LENGTH];
    memset(cipher_text,0,BUFFER_LENGTH);
    char plain_text_output[BUFFER_LENGTH];
    memset(plain_text_output,0,BUFFER_LENGTH);
    keyInstance keyinst;
    cipherInstance cipherinst;
        makeKey(&keyinst, DIR_DECRYPT, keyLength, jpeg_ise::get_key());
    char iv[BUFFER_LENGTH];
    memset(iv,0,BUFFER_LENGTH);
    cipherInit(&cipherinst, MODE_ECB, iv);

    int ct_counter = 0;

    char b;
        // begin ise selective decryption algorithm
    while (infs.read(&b,sizeof(b)))
    {
                // send unencrypted data to output file
        if (inhuff == false && split_block == false)
        {
            if ((byte)b == 0xFF)
            {
                outfs.write(&b,sizeof(b));
                infs.read(&b,sizeof(b));
                if ((byte)b == 0xC4 || (byte)b == 0xC0)
                {
```

```
                    inhuff = true;
                }
            }
            outfs.write(&b,sizeof(b));
        }
                // if half of a jpeg marker was found
                // split block case
        else if (inhuff == true && split_block == true)
        {
                        // fill buffer to be decrypted
            while (ct_counter < BUFFER_LENGTH)
            {
                cipher_text[ct_counter++] = b;
                if(ct_counter < BUFFER_LENGTH) infs.read(&b,sizeof(b));
            }
                        // decrypt buffer
            blockDecrypt(&cipherinst,&keyinst,cipher_text,keyLength,plain_text);

                        // if first byte is not second half of huffman marker
                        if (plain_text[0] == 0xDA)
            {
                                // stop decryption
                inhuff = false;
            }
            split_block = false;

                        // send decrypted data to output file
                        for (int i = 0; i < BUFFER_LENGTH; i++)
            {
                plain_text_output[i]=(char)plain_text[i];
                outfs.write(&plain_text_output[i],sizeof(plain_text_output[i]));
            }
                        // reset the buffer
            memset(plain_text,0,BUFFER_LENGTH);
            memset(plain_text_output,0,BUFFER_LENGTH);
            memset(cipher_text,0,BUFFER_LENGTH);
            ct_counter = 0;
        }


                // in the huffman table
        else if(inhuff == true)
        {
                        // fill the buffer to be decrypted
            while (ct_counter < BUFFER_LENGTH)
            {
                cipher_text[ct_counter++] = b;
                                if(ct_counter < BUFFER_LENGTH) infs.read(&b,size
of(b));
            }

                        // decrypt the buffer
            blockDecrypt(&cipherinst,&keyinst,cipher_text,keyLength,plain_text);

                        // search through decrypted data
            for (int i = 0; i < BUFFER_LENGTH; i++)
            {
                                // if marker found
                if (plain_text[i] == 0xFF && i != 15)
                {
                                        // if not huffman marker
                    if (plain_text[i+1] == 0xDA)
                    {
                                                // stop decryption
                        inhuff = false;
                        break;
                    }
                }
                                // if half of jpeg marker found
                else if (plain_text[i] == 0xFF && i == 15)
                {
                                        // go to split block case
                    split_block = true;
                }
            }
                        // send decrypted data to output file
            for (int i = 0; i < BUFFER_LENGTH; i++)
            {
                plain_text_output[i]=(char)plain_text[i];
                outfs.write(&plain_text_output[i],sizeof(plain_text_output[i]));
            }
                        // reset the buffer
            memset(plain_text,0,BUFFER_LENGTH);
            memset(plain_text_output,0,BUFFER_LENGTH);
            memset(cipher_text,0,BUFFER_LENGTH);
            ct_counter = 0;
        }
    }

    infs.close();
    outfs.close();

    return 0;
}
```