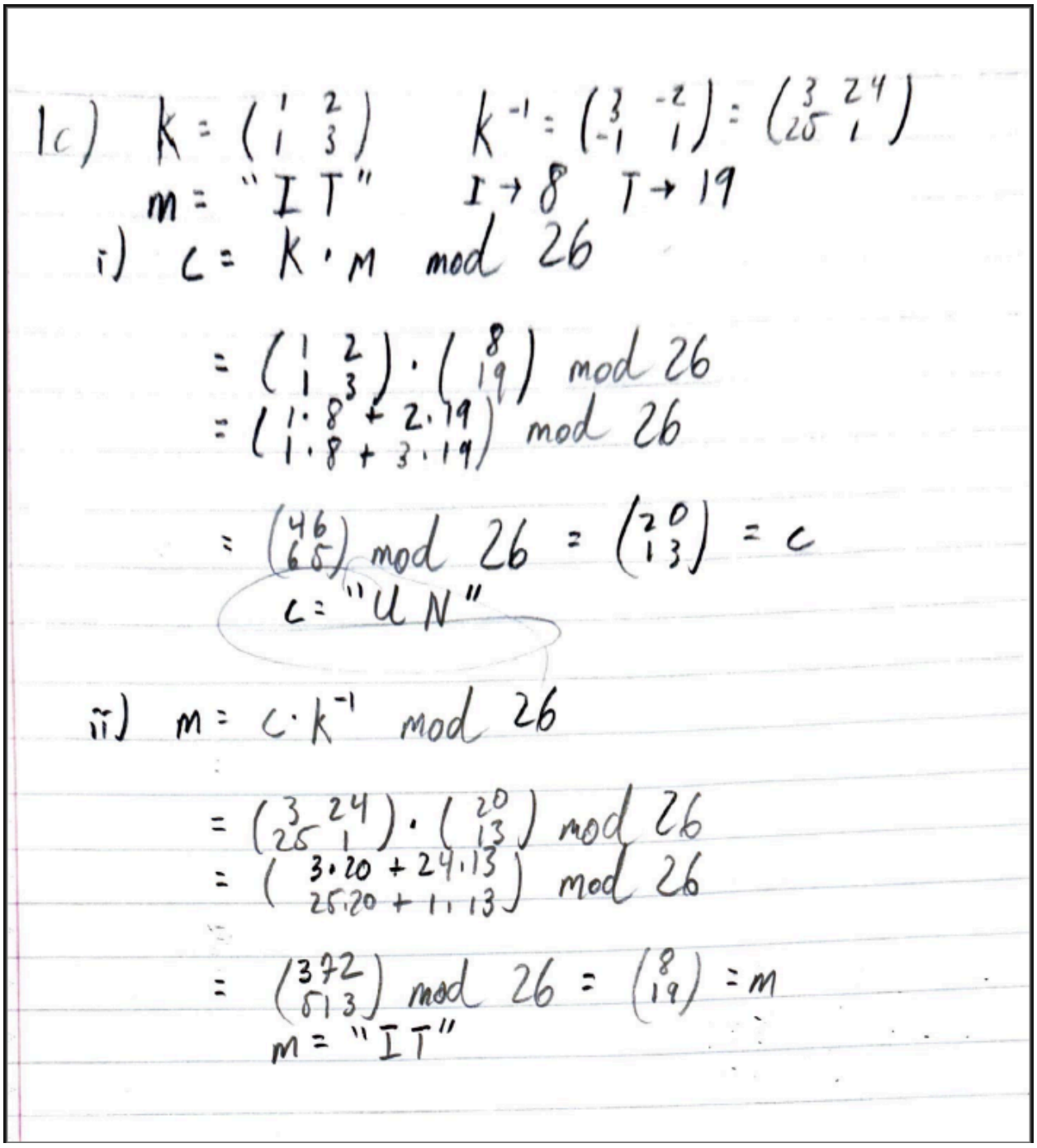
1:

(a) thisisaneasyproblem, key: 25

(b) this class is great

(c)



2:

(a) ECB OFB 12/15

(b) ECB

(c) CBC

(d) ECB

(e) OFB

3:

(a) (i) YES (ii) NO, BG can take the transmitted nonce and get the DES key, take the transmitted ciphertext and decrypt S from it. BG can fake being A by getting a new nonce from B, encrypting it using the same standard as A, and sending that to B to be authenticated. This lets BG masquerade as A. 16/20

(b) NO, bits can be easily reversed because there is no avalanche effect with the bits. BG can easily determine the plaintext by testing all XOR values.

No answer is correct, but the reasoning is not OK. BG can also calculate X from the date and then recove

4:

Self Critique:

I can confirm the program runs as intended for part (a). Using an external calculator I can confirm part (b) also works. I cannot confirm part (c) but believe it works as intended. The lack of confidence for part (b) and (c) comes from my paper calculations being wrong from what is expected.

a,b OK, c not OK

Output:

Running homework tests for SDES class 30/40

Running Requirement - InputA...

---

encrypt():

The intermediate value after the SW operation is: 11101100

The value of the ciphertext is: 01110101

---

decrypt():

The intermediate value after the SW operation is: 11001110

The value of the plaintext is: 10111101

Requirement- InputA -> Passed Test ✓

Running Requirement - InputB...

---
encrypt():
The intermediate value after the SW operation is: 11010110
The value of the ciphertext is: 11001101
---
decrypt():
The intermediate value after the SW operation is: 01101101
The value of the plaintext is: 01110111
Requirement- InputB -> Passed Test ✓

Running Requirement - InputC...
---
encrypt():
The intermediate value after the SW operation is: 11010100
The value of the ciphertext is: 00000101
--- For (c) ciphertext is 10011001
decrypt():
The intermediate value after the SW operation is: 01001101
The value of the plaintext is: 01110111 Not the plaintext you started with
Requirement- InputC -> Passed Test ✓

Estimated score on homework = 30/30 ???

Program Listing:

main.cpp

```
#include "UnitTest-SDES.hpp"

int main(){
   UnitTest_RunHomework();
   return 0;
}
```

UnitTest-SDES.hpp

```
#ifndef UNITTEST_SDES_HPP
#define UNITTEST_SDES_HPP

#include <iostream>
#include "SDES.hpp"
#include "mod-SDES.hpp"

using namespace std;

/// ------------------------------------------------------------------
/// @brief check the SDES class against question 4a and verify correctness
/// @return int 0 on pass, 1 on fail
/// ------------------------------------------------------------------
int UnitTest_InputA();

/// ------------------------------------------------------------------
/// @brief check the SDES class against question 4b and verify correctness
/// @return int 0 on pass, 1 on fail
/// ------------------------------------------------------------------
int UnitTest_InputB();

/// ------------------------------------------------------------------
/// @brief check the SDES class against question 4c and verify correctness
/// @return int 0 on pass, 1 on fail
/// ------------------------------------------------------------------
int UnitTest_InputC();

/// ------------------------------------------------------------------
/// @brief check the SDES class against question 4a and verify correctness
/// @return int 0 on pass, 1 on fail
/// ------------------------------------------------------------------
int UnitTest_RunHomework();

/// ------------------------------------------------------------------
/// @brief run all the unit tests for the SDES class
/// @return int 0 on pass, 1 on fail
/// ------------------------------------------------------------------
```

```
int UnitTest_RunAll();

#endif
```

UnitTest-SDES.cpp

```
#include "UnitTest-SDES.hpp"

/// ------------------------------------------------------------------------
/// @brief check the SDES class against question 4a and verify correctness
/// @return int 0 on pass, 1 on fail
/// ------------------------------------------------------------------------
int UnitTest_InputA(){
   try{
      const unsigned int plaintext = 0b10111101;
      const unsigned int key = 0b1010000010;
      const unsigned int ciphertext = 0b01110101;

      // create class with submission output on
      SDES cipher(key, true);
      const unsigned int resultCipher = cipher.encrypt(plaintext);
      if (resultCipher != ciphertext) throw "Returned cipher text is wrong!";

      const unsigned int resultPlain = cipher.decrypt(resultCipher);
      if (resultPlain != plaintext) throw "Returned plain text is wrong!";

   } catch(const char* txt) {
      cout << txt << endl; return 1;
   } catch (...) { return 1; }

   return 0;
}

/// ------------------------------------------------------------------------
/// @brief check the SDES class against question 4b and verify correctness
/// @return int 0 on pass, 1 on fail
/// ------------------------------------------------------------------------
int UnitTest_InputB(){
   try{
      const unsigned int plaintext = 0b01110111;
      const unsigned int key = 0b1101110111;
      const unsigned int ciphertext = 0b11001101;
```

```
        // create class with submission output on
        SDES cipher(key, true);
        const unsigned int resultCipher = cipher.encrypt(plaintext);
        if (resultCipher != ciphertext) throw "Returned cipher text is wrong!";

        const unsigned int resultPlain = cipher.decrypt(resultCipher);
        if (resultPlain != plaintext) throw "Returned plain text is wrong!";

    } catch(const char* txt) {
        cout << txt << endl; return 1;
    } catch (...) { return 1; }

    return 0;
}

/// ------------------------------------------------------------------------
/// @brief check the SDES class against question 4c and verify correctness
/// @return int 0 on pass, 1 on fail
/// ------------------------------------------------------------------------
int UnitTest_InputC(){
    try{
        const unsigned int plaintext = 0b01110111;
        const unsigned int key = 0b1101110111;
        const unsigned int ciphertext = 0b00000101;

        // create class with submission output on
        modSDES cipher(key, true);

        const unsigned int resultCipher = cipher.modEncrypt(plaintext);
        if (resultCipher != ciphertext) throw "Returned cipher text is wrong!";

        const unsigned int resultPlain = cipher.modDecrypt(resultCipher);
        if (resultPlain != plaintext) throw "Returned plain text is wrong!";
    } catch(const char* txt) {
        cout << txt << endl; return 1;
    } catch (...) { return 1; }

    return 0;
}
```

```
/// ------------------------------------------------------------------
/// @brief check the SDES class against question 4a and verify correctness
/// @return int 0 on pass, 1 on fail
/// ------------------------------------------------------------------
int UnitTest_RunHomework(){
    int points = 0;
    const int maxPoints = 30;

    cout << "Running homework tests for SDES class" << endl;
    cout << "-------------------------------------------------" << endl;

    cout << "Running Requirement - InputA..." << endl;
    if (UnitTest_InputA()) { cout << "Requirement - InputA -> Failed Test \u274c" << endl; }
    else { cout << "Requirement- InputA -> Passed Test \u2713" << endl; points += 10; }

    cout << "-------------------------------------------------" << endl;
    cout << endl;

    cout << "Running Requirement - InputB..." << endl;
    if (UnitTest_InputB()) { cout << "Requirement - InputB -> Failed Test \u274c" << endl; }
    else { cout << "Requirement- InputB -> Passed Test \u2713" << endl; points += 10; }

    cout << "-------------------------------------------------" << endl;
    cout << endl;

    cout << "Running Requirement - InputC..." << endl;
    if (UnitTest_InputC()) { cout << "Requirement - InputC -> Failed Test \u274c" << endl; }
    else { cout << "Requirement- InputC -> Passed Test \u2713" << endl; points += 10; }

    cout << "-------------------------------------------------" << endl;
    cout << endl;

    cout << "Estimated score on homework = " << points << "/" << maxPoints << endl;
    // Note: not really the accurate score but just a fun tracker

    return 0;
}
```

```
/// ------------------------------------------------------------------
/// @brief run all the unit tests for the SDES class
/// @return int 0 on pass, 1 on fail
/// ------------------------------------------------------------------
int UnitTest_RunAll(){
   int passed = 0, failed = 0;

   cout << "Running all unit tests for SDES class" << endl;
   cout << "-------------------------------------------------" << endl;

   cout << "Running Unit Test - InputA..." << endl;
   if (UnitTest_InputA()) { cout << "Unit Test - InputA -> Failed Test \u274c" << endl;
failed++;}
   else { cout << "Unit Test - InputA -> Passed Test \u2713" << endl; passed++; }

   cout << "-------------------------------------------------" << endl;

   cout << "Running Unit Test - InputB..." << endl;
   if (UnitTest_InputB()) { cout << "Unit Test - InputB -> Failed Test \u274c" << endl;
failed++;}
   else { cout << "Unit Test - InputB -> Passed Test \u2713" << endl; passed++; }

   cout << "-------------------------------------------------" << endl;

   cout << "Running Unit Test - InputC..." << endl;
   if (UnitTest_InputC()) { cout << "Unit Test - InputC -> Failed Test \u274c" << endl;
failed++;}
   else { cout << "Unit Test - InputC -> Passed Test \u2713" << endl; passed++; }



   cout << "-------------------------------------------------" << endl;
   cout << "Finished running all tests" << endl;
   cout << "\u2713 Passed Tests: " << passed << endl;
   cout << "\u274c Failed Tests: " << failed << endl;

   return 0;
}
```

SDES.hpp

```
#ifndef SDES_HPP
#define SDES_HPP

// bit masks used for keyGen()
#define hx2E0 0b1111100000
#define hx01F 0b0000011111

// bit masks used for encrypt()/decrypt()
#define hxF0 0b11110000
#define hx0F 0b00001111

// bit masks used for processKeyAndText()
#define hx9 0b1001
#define hx6 0b0110
#define hx1 0b0001


#include <string>
#include <iostream>
#include <iomanip>
#include <bitset>

using namespace std;

class SDES {
  protected:
    // encryption / decryption keys
    unsigned int keyOne = 0;
    unsigned int keyTwo = 0;

    // S-boxes used in processKeyAndText
    unsigned int S0[4][4] = {
      {0b01, 0b00, 0b11, 0b10},
      {0b11, 0b10, 0b01, 0b00},
      {0b00, 0b10, 0b01, 0b11},
      {0b11, 0b01, 0b11, 0b10}
    };
```

```
unsigned int S1[4][4] = {
  {0b00, 0b01, 0b10, 0b11},
  {0b10, 0b00, 0b01, 0b11},
  {0b11, 0b00, 0b01, 0b00},
  {0b10, 0b01, 0b00, 0b11}
};

// output flags
bool output = false;
bool verbose = false;

/// ---------------------------------------------------------------------
/// @brief Generates keyOne and keyTwo according to SDES standard
/// @param key (unsigned int) a 10-bit key that derives keyOne and keyTwo
/// ---------------------------------------------------------------------
void keyGen(unsigned int key);

/// ---------------------------------------------------------------------
/// @brief a 10-bit permutation for SDES key generation
///        the first step when encoding the keys
/// @param bits (unsigned int) a 10-bit number to permute
/// @return (unsigned int) the new 10-bit number changed according to SDES key
///         generation standards
/// ---------------------------------------------------------------------
unsigned int permute10(unsigned int bits);

/// ---------------------------------------------------------------------
/// @brief an 8-bit permutation for SDES key generation
///        the second permutation when encoding the keys
/// @param bits (unsigned int) an 8-bit number to permute
/// @return (unsigned int) the new 8-bit number changed from bits
///         according to SDES key generation standards
/// ---------------------------------------------------------------------
unsigned int permute8(unsigned int bits);

/// ---------------------------------------------------------------------
/// @brief a 5-bit left shift, where the left-most bit becomes the right-most bit
/// @param bits (unsigned int) a 5-bit number to shift
/// @return (unsigned int) the new 5-bit number changed from bits
```

```
    ///        with its bits shifted left and wrapped once
    /// ------------------------------------------------------------------------
    unsigned int leftShift(unsigned int bits);

    /// ------------------------------------------------------------------------
    /// @brief an 8-bit permutation for SDES encryption/decryption
    /// @param text (unsigned int) an 8-bit number to permute
    /// @return (unsigned int) the new 8-bit number changed from bits
    ///        according to SDES encryption/decryption standards
    /// ------------------------------------------------------------------------
    unsigned int initPermute(unsigned int text);

    /// ------------------------------------------------------------------------
    /// @brief the part of the SDES encryption/decryption where the keys are applied
    ///       to the message
    /// @note known as the function F in notes
    /// @param key (unsigned int) an 8-bit number to serve as the key (usually keyOne or
keyTwo)
    /// @param text (unsigned int) a 4-bit number to serve as the message to
encrypt/decrypt,
    ///            usually half of the message block
    /// @return (unsigned int) a 4-bit number that represents the message
    ///        when the given key is applied
    /// ------------------------------------------------------------------------
    unsigned int processKeyAndText(unsigned int key, unsigned int text);

    /// ------------------------------------------------------------------------
    /// @brief takes a 4-bit number and makes an 8-bit permutation from it
    ///       according to SDES encryption/decryption
    /// @param bits (unsigned int) a 4-bit number to permute
    /// @return (unsigned int) an 8-bit number expanded & permuted from bits
    /// ------------------------------------------------------------------------
    unsigned int expandPermute4(unsigned int bits);

    /// ------------------------------------------------------------------------
    /// @brief a 4-bit permutation for SDES encryption/decryption
    /// @param bits (unsigned int) a 4-bit number to permute
    /// @return (unsigned int) the new 4-bit number changed according to SDES
    ///        encryption/decryption
```

```
    /// ------------------------------------------------------------------
    unsigned int permute4(unsigned int bits);

    /// ------------------------------------------------------------------
    /// @brief an 8-bit permutation to end SDES encryption/decryption,
    ///        the inverse of initPermute()
    /// @param text (unsigned int) an 8-bit number to permute
    /// @return (unsigned int) the new 8-bit number changed from bits
    ///         according to SDES encryption/decryption standards
    /// ------------------------------------------------------------------
    unsigned int inverseInitPermute(unsigned int text);

  public:
    /// ------------------------------------------------------------------
    /// @brief SDES class constructor
    /// @param key (unsigned int) a 10-bit number that serves as the encryption key
    /// @param output (bool) flag to send homework outputs to console
    /// @param verbose (bool) flag to send debugging outputs to console
    /// ------------------------------------------------------------------
    SDES(const unsigned int key, bool output=false, bool verbose=false);
    ~SDES() = default;

    /// ------------------------------------------------------------------
    /// @brief the SDES encryption algorithm to obscure information
    /// @param plaintext (unsigned int) the 8-bit number considered as the message
    /// @return (unsigned int) the 8-bit number cipher text using keyOne and keyTwo
    /// ------------------------------------------------------------------
    unsigned int encrypt(const unsigned int plaintext);

    /// ------------------------------------------------------------------
    /// @brief the SDES decryption algorithm to read obscured information
    /// @param ciphertext (unsigned int) the 8-bit number considered as the cipher text
    /// @return (unsigned int) the 8-bit number plain text using keyTwo and keyOne
    /// ------------------------------------------------------------------
    unsigned int decrypt(const unsigned int ciphertext);
};

#endif
```

SDES.cpp

```
#include "SDES.hpp"

/// ------------------------------------------------------------------------
/// @brief SDES class constructor
/// @param key (unsigned int) a 10-bit number that serves as the encryption key
/// @param output (bool) flag to send homework outputs to console
/// @param verbose (bool) flag to send debugging outputs to console
/// ------------------------------------------------------------------------
SDES::
SDES(const unsigned int key, bool output, bool verbose) : output(output),
verbose(verbose) {
   keyGen(key); // on class creation, create the keys
}

/// ------------------------------------------------------------------------
/// @brief Generates keyOne and keyTwo according to SDES standard
/// @param key (unsigned int) a 10-bit key that derives keyOne and keyTwo
/// ------------------------------------------------------------------------
void SDES::
keyGen(const unsigned int key){
   // P10 -> SHIFT -> P8 -> keyOne
   // P10 -> SHIFT -> SHIFT-> P8 -> keyTwo

   if (verbose)
      cout << "-------------------------------------------------------\n" <<
      "keyGen():" << endl;

   // permute10 the given key
   unsigned int resultP10 = permute10(key);

   // grab the left most bits
   unsigned int leftBits = resultP10 & hx2E0;
   leftBits = leftBits >> 5;                // make bits be in least significant bit slot
   leftBits = leftShift(leftBits);          // left shift the left bits

   // grab the right bits and left shift them
   unsigned int rightBits = leftShift(resultP10 & hx01F);
```

```
    // recombine the bits into one variable
    unsigned int combined = (leftBits << 5) + rightBits;

    // the permute8 of the previous combination is the first key
    keyOne = permute8(combined);

    // left shift both the left and right bits by two and recombine them
    leftBits = leftShift(leftShift(leftBits));
    rightBits = leftShift(leftShift(rightBits));
    combined = (leftBits << 5) + rightBits;

    // the permute8 of the latest combination is the second key
    keyTwo = permute8(combined);

    if (verbose){
        cout << "-------------------------------------------------------" << endl;
        cout << "The value of KeyOne is: " <<
            right << setw(34) << bitset<8>(keyOne) << endl;
        cout << "The value of KeyTwo is: " <<
            right << setw(34) << bitset<8>(keyTwo) << endl;
    }
}

/// ------------------------------------------------------------------------
/// @brief a 10-bit permutation for SDES key generation
///        the first step when encoding the keys
/// @param bits (unsigned int) a 10-bit number to permute
/// @return (unsigned int) the new 10-bit number changed according to SDES key
///         generation standards
/// ------------------------------------------------------------------------
unsigned int SDES::
permute10(unsigned int bits){
    // The 10 bit representation:
    // k1 k2 k3 k4 k5 k6 k7 k8 k9 k10
    // becomes
    // k3 k5 k2 k7 k4 k10 k1 k9 k8 k6

    // create bit masks
```

```
    unsigned int
    k1 =  0b1000000000,
    k2 =  0b0100000000,
    k3 =  0b0010000000,
    k4 =  0b0001000000,
    k5 =  0b0000100000,
    k6 =  0b0000010000,
    k7 =  0b0000001000,
    k8 =  0b0000000100,
    k9 =  0b0000000010,
    k10 = 0b0000000001;

    // grab the respectve bits and put them into their new positions
    unsigned int permutation =
    ((bits & k3) << 2) +
    ((bits & k5) << 3) +
    ((bits & k2) >> 1) +
    ((bits & k7) << 3) +
    ((bits & k4) >> 1) +
    ((bits & k10) << 4) +
    ((bits & k1) >> 6) +
    ((bits & k9) << 1) +
    ((bits & k8) >> 1) +
    ((bits & k6) >> 4);

    if (verbose)
        cout << "The permute10 result of " << bitset<10>(bits) << " is: "
        << right << setw(19) << bitset<10>(permutation) << endl;

    return permutation;
}

/// ------------------------------------------------------------------
/// @brief an 8-bit permutation for SDES key generation
///        the second permutation when encoding the keys
/// @param bits (unsigned int) an 8-bit number to permute
/// @return (unsigned int) the new 8-bit number changed from bits
///         according to SDES key generation standards
/// ------------------------------------------------------------------
```

```
unsigned int SDES::
permute8(unsigned int bits){
   // The 10 bit representation:
   // k1 k2 k3 k4 k5 k6 k7 k8 k9 k10
   // becomes
   // 0  0  k6 k3 k7 k4 k8 k5 k10 k9

   // create bit masks
   unsigned int
   k3 =  0b0010000000,
   k4 =  0b0001000000,
   k5 =  0b0000100000,
   k6 =  0b0000010000,
   k7 =  0b0000001000,
   k8 =  0b0000000100,
   k9 =  0b0000000010,
   k10 = 0b0000000001;

   // grab the respectve bits and put them into their new positions
   // shifts are done to make k9 the least significant bit
   unsigned int permutation =
   ((bits & k6) << 3) +
   ((bits & k3) >> 1) +
   ((bits & k7) << 2) +
   ((bits & k4) >> 2) +
   ((bits & k8) << 1) +
   ((bits & k5) >> 3) +
   ((bits & k10) << 1) +
   ((bits & k9) >> 1);

   if (verbose)
      cout << "The permute8 result of " << bitset<10>(bits) << " is: "
      << right << setw(20) << bitset<8>(permutation) << endl;


   return permutation;
}

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

```
/// @brief a 5-bit left shift, where the left-most bit becomes the right-most bit
/// @param bits (unsigned int) a 5-bit number to shift
/// @return (unsigned int) the new 5-bit number changed from bits
///         with its bits shifted left and wrapped once
/// ----------------------------------------------------------------------
unsigned int SDES::
leftShift(unsigned int bits){
   // The 5 bit representation:
   // k1 k2 k3 k4 k5
   // becomes
   // k2 k3 k4 k5 k1

   // create bit mask
   unsigned int k1 = 0b10000;

   // shift bits left once and add masked bit to the beginning
   unsigned int shift =
   ((bits & k1) >> 4) + (bits << 1);

   if (verbose)
      cout << "The result of left shifting " << bitset<5>(bits) <<
      " is: " << right << setw(20) << bitset<5>(shift) << endl;


   return shift & hx01F; // remove extra leading 1s and return
}

/// ----------------------------------------------------------------------
/// @brief the SDES encryption algorithm to obscure information
/// @param plaintext (unsigned int) the 8-bit number considered as the message
/// @return (unsigned int) the 8-bit number cipher text using keyOne and keyTwo
/// ----------------------------------------------------------------------
unsigned int SDES::
encrypt(const unsigned int plaintext){
   // IP -> F w/ K1 -> SW -> F w/ K2 -> inverseIP

   if (output || verbose)
      cout << "-----------------------------------------------------\n" <<
      "encrypt():" << endl;
```

```
    // permute the initial bits through IP and separate left and right bits
    unsigned int result = initPermute(plaintext);
    unsigned int leftBits = (result & hxF0) >> 4; // keep start at least significant bit
    unsigned int rightBits = result & hx0F;

    // grab bits to xor the left bits
    result = processKeyAndText(keyOne, rightBits);
    leftBits = leftBits ^ result;

    // swap left and right bits here
    result = (rightBits << 4) + leftBits; // this line is used for outputs

    if (verbose)
        cout << "Swapping left and right bits..." << endl;
    if (output || verbose)
        cout << "The intermediate value after the SW operation is: " <<
        right << setw(0) << bitset<8>(result) << endl;

    // grab bits to xor with right bits (after swap)
    result = processKeyAndText(keyTwo, leftBits);
    rightBits = rightBits ^ result;

    // recombine the left and right bits and run them through inverse IP to get ciphertext
    unsigned int combined = (rightBits << 4) + leftBits;
    unsigned int ciphertext = inverseInitPermute(combined);

    if (output || verbose) {
        if (verbose) cout << "--------------------------------------------------------" << endl;
        cout << "The value of the ciphertext is: " <<
            right << setw(26) << bitset<8>(ciphertext) << endl;
     }

    return ciphertext;
}

/// ------------------------------------------------------------------------
/// @brief the SDES decryption algorithm to read obscured information
/// @param ciphertext (unsigned int) the 8-bit number considered as the cipher text
```

```
/// @return (unsigned int) the 8-bit number plain text using keyTwo and keyOne
/// ----------------------------------------------------------------------
unsigned int SDES::
decrypt(const unsigned int ciphertext){
   // IP -> F w/ K2 -> SW -> F w/ K1 -> inverseIP

   if (output|| verbose)
      cout << "-----------------------------------------------------\n" <<
      "decrypt():" << endl;

   // permute the initial bits through IP and separate left and right bits
   unsigned int result = initPermute(ciphertext);
   unsigned int leftBits = (result & hxF0) >> 4; // keep start at least significant bit
   unsigned int rightBits = result & hx0F;

   // grab bits to xor the left bits
   result = processKeyAndText(keyTwo, rightBits);
   leftBits = leftBits ^ result;

   // swap left and right bits here
   result = (rightBits << 4) + leftBits; // this line is used for outputs

   if (verbose)
      cout << "Swapping left and right bits..." << endl;
   if (output || verbose)
      cout << "The intermediate value after the SW operation is: " <<
      right << setw(0) << bitset<8>(result) << endl;

   // grab bits to xor with right bits (after swap)
   result = processKeyAndText(keyOne, leftBits);
   rightBits = rightBits ^ result;

   // recombine the left and right bits and run them through inverse IP to get plaintext
   unsigned int combined = (rightBits << 4) + leftBits;
   unsigned int plaintext = inverseInitPermute(combined);

   if (output || verbose) {
      if (verbose) cout << "-----------------------------------------------------" << endl;
      cout << "The value of the plaintext is: " <<
```

```
        right << setw(27) << bitset<8>(plaintext) << endl;
    }

    return plaintext;
}

/// ----------------------------------------------------------------------
/// @brief an 8-bit permutation for SDES encryption/decryption
/// @param text (unsigned int) an 8-bit number to permute
/// @return (unsigned int) the new 8-bit number changed from bits
///         according to SDES encryption/decryption standards
/// ----------------------------------------------------------------------
unsigned int SDES::
initPermute(unsigned int text) {
    // The 8 bit representation:
    // k1 k2 k3 k4 k5 k6 k7 k8
    // becomes
    // k2 k6 k3 k1 k4 k8 k5 k7

    // create bit masks
    unsigned int
    k1 =  0b10000000,
    k2 =  0b01000000,
    k3 =  0b00100000,
    k4 =  0b00010000,
    k5 =  0b00001000,
    k6 =  0b00000100,
    k7 =  0b00000010,
    k8 =  0b00000001;

    // grab the respectve bits and put them into their new positions
    unsigned int permutation =
    ((text & k2) << 1) +
    ((text & k6) << 4) +
    ((text & k3) << 0) +
    ((text & k1) >> 3 ) +
    ((text & k4) >> 1) +
    ((text & k8) << 2) +
    ((text & k5) >> 2) +
```

```
    ((text & k7) >> 1);

    if (verbose)
        cout << "The initPermute result of " << bitset<8>(text) << " is: "
        << right << setw(19) << bitset<8>(permutation) << endl;

    return permutation;
}

/// ------------------------------------------------------------------------
/// @brief the part of the SDES encryption/decryption where the keys are applied
///        to the message
/// @note known as the function F in notes
/// @param key (unsigned int) an 8-bit number to serve as the key (usually keyOne or
keyTwo)
/// @param text (unsigned int) a 4-bit number to serve as the message to
encrypt/decrypt,
///             usually half of the message block
/// @return (unsigned int) a 4-bit number that represents the message
///         when the given key is applied
/// ------------------------------------------------------------------------
unsigned int SDES::
processKeyAndText(unsigned int key, unsigned int text){
    // expand the 4-bit text to 8 bits and xor with key
    unsigned int result = expandPermute4(text);
    result = result ^ key;

    if (verbose)
        cout << "The XOR of expandPermute4 and key is: " << right << setw(20) <<
bitset<8>(result) << endl;

    // grab left and right bits
    unsigned int leftBits = (hxF0 & result) >> 4; // shift to start at least significant bit
    unsigned int rightBits = hx0F & result;

    if (verbose) {
        cout << "Indexing S0" << "[" << (leftBits & hx1) + ((leftBits & hx9) >> 2) << "]" << "[" <<
((leftBits & hx6) >> 1) << "]" <<
```

```
      " and S1" << "[" << (rightBits & hx1) + (rightBits >> 2) << "]" << "[" << ((rightBits & hx6)
>> 1) << "]" << "..."<< endl;
   }

   // get new bits to use by indexing the S-boxes
   // bit representation b1 b2 b3 b4 for left/right bits
   // left bits are received by indexing S0[b1 b4][b2 b3]
   // right bits are received by indexing S1[b1 b4][b2 b3]
   unsigned int combined =
   (S0[(leftBits & hx1) + ((leftBits & hx9) >> 2)][(leftBits & hx6) >> 1] << 2) +
   S1[(rightBits & hx1) + (rightBits >> 2)][(rightBits & hx6) >> 1];

   if (verbose)
      cout << "The combination of S0 and S1 is: " << right << setw(25) <<
bitset<4>(combined) << endl;

   // get the final result by permute4 the lastest combined bits
   result = permute4(combined);

   if (verbose)
      cout << "The processing of key " << bitset<8>(key) << " and text "<< bitset<4>(text)
<<
      " is: " << right << setw(9) << bitset<4>(result) << endl;

   return result;
}

/// ----------------------------------------------------------------------
/// @brief takes a 4-bit number and makes an 8-bit permutation from it
///        according to SDES encryption/decryption
/// @param bits (unsigned int) a 4-bit number to permute
/// @return (unsigned int) an 8-bit number expanded & permuted from bits
/// ----------------------------------------------------------------------
unsigned int SDES::
expandPermute4(unsigned int bits){
   // The 4 bit representation:
   // 0  0  0  0  k1 k2 k3 k4
   // becomes
   // k4 k1 k2 k3 k2 k3 k4 k1
```

```
    // create bit masks
    unsigned int
    k1 =  0b1000,
    k2 =  0b0100,
    k3 =  0b0010,
    k4 =  0b0001;

    // grab the respectve bits and put them into their new positions (with duplications)
    unsigned int permutation =
    ((bits & k4) << 7) +
    ((bits & k1) << 3) +
    ((bits & k2) << 3) +
    ((bits & k3) << 3) +
    ((bits & k2) << 1) +
    ((bits & k3) << 1) +
    ((bits & k4) << 1) +
    ((bits & k1) >> 3);

    if (verbose)
        cout << "The expandPermute4 result of " << bitset<4>(bits) << " is: "
        << right << setw(20) << bitset<8>(permutation) << endl;

    return permutation;
}

/// ----------------------------------------------------------------------
/// @brief a 4-bit permutation for SDES encryption/decryption
/// @param bits (unsigned int) a 4-bit number to permute
/// @return (unsigned int) the new 4-bit number changed according to SDES
///         encryption/decryption
/// ----------------------------------------------------------------------
unsigned int SDES::
permute4(unsigned int bits){
    // The 4 bit representation:
    // k1 k2 k3 k4
    // becomes
    // k2 k4 k3 k1
```

```
    // create bit masks
    unsigned int
    k1 =  0b1000,
    k2 =  0b0100,
    k3 =  0b0010,
    k4 =  0b0001;

    // grab the respectve bits and put them into their new positions
    unsigned int permutation =
    ((bits & k2) << 1) +
    ((bits & k4) << 2) +
    ((bits & k3) << 0) +
    ((bits & k1) >> 3);

    if (verbose)
        cout << "The permute4 result of " << bitset<4>(bits) << " is: "
        << right << setw(26) << bitset<4>(permutation) << endl;

    return permutation;
}

/// ------------------------------------------------------------------------
/// @brief an 8-bit permutation to end SDES encryption/decryption,
///        the inverse of initPermute()
/// @param text (unsigned int) an 8-bit number to permute
/// @return (unsigned int) the new 8-bit number changed from bits
///         according to SDES encryption/decryption standards
/// ------------------------------------------------------------------------
unsigned int SDES::
inverseInitPermute(unsigned int text){
    // The 8 bit representation:
    // k1 k2 k3 k4 k5 k6 k7 k8
    // becomes
    // k4 k1 k3 k5 k7 k2 k8 k6

    // create bit masks
    unsigned int
    k1 =  0b10000000,
    k2 =  0b01000000,
```

```
    k3 =  0b00100000,
    k4 =  0b00010000,
    k5 =  0b00001000,
    k6 =  0b00000100,
    k7 =  0b00000010,
    k8 =  0b00000001;

    // grab the respectve bits and put them into their new positions
    unsigned int permutation =
    ((text & k4) << 3) +
    ((text & k1) >> 1) +
    ((text & k3) << 0) +
    ((text & k5) << 1 ) +
    ((text & k7) << 2) +
    ((text & k2) >> 4) +
    ((text & k8) << 1) +
    ((text & k6) >> 2);

    if (verbose)
        cout << "The inverseInitPermute result of " << bitset<8>(text) << " is: "
        << right << setw(12) << bitset<8>(permutation) << endl;

    return permutation;
}
```

mod-SDES.hpp

```
#ifndef MOD_SDES_HPP
#define MOD_SDES_HPP

#include "SDES.hpp"

class modSDES : public SDES {
   protected:
      // modified S-box used in processKeyAndText
      unsigned int S1p[4][4] = {
         {0b10, 0b00, 0b01, 0b11},
         {0b00, 0b01, 0b10, 0b11},
         {0b11, 0b00, 0b01, 0b00},
         {0b10, 0b01, 0b00, 0b11}
      };

      /// ------------------------------------------------------------------
      /// @brief the part of the SDES encryption/decryption where the keys are applied
      ///        to the message
      /// @note known as the function F in notes, but uses S1p instead of S1
      /// @param key (unsigned int) an 8-bit number to serve as the key (usually keyOne
or keyTwo)
      /// @param text (unsigned int) a 4-bit number to serve as the message to
encrypt/decrypt,
      ///            usually half of the message block
      /// @return (unsigned int) a 4-bit number that represents the message
      ///         when the given key is applied
      /// ------------------------------------------------------------------
      unsigned int modProcessKeyAndText(unsigned int key, unsigned int text);

   public:
      /// ------------------------------------------------------------------
      /// @brief mod-SDES class constructor
      /// @param key (unsigned int) a 10-bit number that serves as the encryption key
      /// @param output (bool) flag to send homework outputs to console
      /// @param verbose (bool) flag to send debugging outputs to console
      /// ------------------------------------------------------------------
```

```
    modSDES(unsigned int key, bool output=false, bool verbose=false) : SDES(key,
output, verbose) {}
    ~modSDES() = default;

    /// ----------------------------------------------------------------------
    /// @brief the SDES encryption algorithm to encrypt information
    /// @param plaintext (unsigned int) the 8-bit number considered as the message to
encrypt
    /// @return (unsigned int) the 8-bit number cipher text using keyOne and keyTwo
    /// ----------------------------------------------------------------------
    unsigned int modEncrypt(const unsigned int plaintext);

    /// ----------------------------------------------------------------------
    /// @brief the SDES decryption algorithm to read encrypted information
    /// @param ciphertext (unsigned int) the 8-bit number considered as the cipher text
to decrypt
    /// @return (unsigned int) the 8-bit number plain text using keyTwo and keyOne
    /// ----------------------------------------------------------------------
    unsigned int modDecrypt(const unsigned int ciphertext);

};

#endif
```

mod-SDES.cpp

```
#include "mod-SDES.hpp"

/// ------------------------------------------------------------------
/// @brief the SDES encryption algorithm to encrypt information
/// @param plaintext (unsigned int) the 8-bit number considered as the message to
encrypt
/// @return (unsigned int) the 8-bit number cipher text using keyOne and keyTwo
/// ------------------------------------------------------------------
unsigned int modSDES::
modEncrypt(const unsigned int plaintext){
   // IP -> F w/ K1 -> SW -> F w/ K2 -> inverseIP

   if (output || verbose)
      cout << "------------------------------------------------------\n" <<
      "encrypt():" << endl;

   // permute the initial bits through IP and separate left and right bits
   unsigned int result = initPermute(plaintext);
   unsigned int leftBits = (result & hxF0) >> 4; // keep start at least significant bit
   unsigned int rightBits = result & hx0F;

   // grab bits to xor the left bits
   result = modProcessKeyAndText(keyOne, rightBits);
   leftBits = leftBits ^ result;

    // swap left and right bits here
   result = (rightBits << 4) + leftBits; // this line is used for outputs

   if (verbose)
      cout << "Swapping left and right bits..." << endl;
   if (output || verbose)
      cout << "The intermediate value after the SW operation is: " <<
      right << setw(0) << bitset<8>(result) << endl;

   // grab bits to xor with right bits (after swap)
   result = modProcessKeyAndText(keyTwo, leftBits);
   rightBits = rightBits ^ result;
```

```
    // recombine the left and right bits and run them through inverse IP to get ciphertext
    unsigned int combined = (rightBits << 4) + leftBits;
    unsigned int ciphertext = inverseInitPermute(combined);

    if (output || verbose) {
        if (verbose) cout << "----------------------------------------------------" << endl;
        cout << "The value of the ciphertext is: " <<
            right << setw(26) << bitset<8>(ciphertext) << endl;
     }

    return ciphertext;
}

/// ----------------------------------------------------------------------
/// @brief the SDES decryption algorithm to read encrypted information
/// @param ciphertext (unsigned int) the 8-bit number considered as the cipher text to
decrypt
/// @return (unsigned int) the 8-bit number plain text using keyTwo and keyOne
/// ----------------------------------------------------------------------
unsigned int modSDES::
modDecrypt(const unsigned int ciphertext){
        // IP -> F w/ K2 -> SW -> F w/ K1 -> inverseIP

    if (output|| verbose)
        cout << "----------------------------------------------------\n" <<
        "decrypt():" << endl;

    // permute the initial bits through IP and separate left and right bits
    unsigned int result = initPermute(ciphertext);
    unsigned int leftBits = (result & hxF0) >> 4; // keep start at least significant bit
    unsigned int rightBits = result & hx0F;

    // grab bits to xor the left bits
    result = modProcessKeyAndText(keyTwo, rightBits);
    leftBits = leftBits ^ result;

    // swap left and right bits here
    result = (rightBits << 4) + leftBits; // this line is used for outputs
```

```
    if (verbose)
        cout << "Swapping left and right bits..." << endl;
    if (output || verbose)
        cout << "The intermediate value after the SW operation is: " <<
        right << setw(0) << bitset<8>(result) << endl;

    // grab bits to xor with right bits (after swap)
    result = modProcessKeyAndText(keyOne, leftBits);
    rightBits = rightBits ^ result;

    // recombine the left and right bits and run them through inverse IP to get plaintext
    unsigned int combined = (rightBits << 4) + leftBits;
    unsigned int plaintext = inverseInitPermute(combined);

    if (output || verbose) {
        if (verbose) cout << "--------------------------------------------------------" << endl;
        cout << "The value of the plaintext is: " <<
            right << setw(27) << bitset<8>(plaintext) << endl;
     }

    return plaintext;
}

/// ------------------------------------------------------------------------
/// @brief the part of the SDES encryption/decryption where the keys are applied
///        to the message
/// @note known as the function F in notes, but uses S1p instead of S1
/// @param key (unsigned int) an 8-bit number to serve as the key (usually keyOne or
keyTwo)
/// @param text (unsigned int) a 4-bit number to serve as the message to
encrypt/decrypt,
///             usually half of the message block
/// @return (unsigned int) a 4-bit number that represents the message
///         when the given key is applied
/// ------------------------------------------------------------------------
unsigned int modSDES::
modProcessKeyAndText(unsigned int key, unsigned int text){
    // expand the 4-bit text to 8 bits and xor with key
    unsigned int result = expandPermute4(text);
```

```
    result = result ^ key;

    if (verbose)
        cout << "The XOR of expandPermute4 and key is: " << right << setw(20) <<
bitset<8>(result) << endl;

    // grab left and right bits
    unsigned int leftBits = (hxF0 & result) >> 4; // shift to start at least significant bit
    unsigned int rightBits = hx0F & result;

    if (verbose) {
        cout << "Indexing S0" << "[" << (leftBits & hx1) + ((leftBits & hx9) >> 2) << "]" << "[" <<
((leftBits & hx6) >> 1) << "]" <<
        " and S1p" << "[" << (rightBits & hx1) + (rightBits >> 2) << "]" << "[" << ((rightBits & hx6)
>> 1) << "]" << "..."<< endl;
    }

    // get new bits to use by indexing the S-boxes
    // bit representation b1 b2 b3 b4 for left/right bits
    // left bits are received by indexing S0[b1 b4][b2 b3]
    // right bits are received by indexing S1p[b1 b4][b2 b3]
    unsigned int combined =
    (S0[(leftBits & hx1) + ((leftBits & hx9) >> 2)][(leftBits & hx6) >> 1] << 2) +
    S1p[(rightBits & hx1) + (rightBits >> 2)][(rightBits & hx6) >> 1];

    if (verbose)
        cout << "The combination of S0 and S1p is: " << right << setw(24) <<
bitset<4>(combined) << endl;

    // get the final result by permute4 the lastest combined bits
    result = permute4(combined);

    if (verbose)
        cout << "The processing of key " << bitset<8>(key) << " and text "<< bitset<4>(text)
<<
        " is: " << right << setw(9) << bitset<4>(result) << endl;

    return result;
}
```