Michael Giglio
Assignment Documentation

S-DES

Solution Description

In order to create this program, I went through the pdf file on vUWS which described the process of S-DES. From there I turned each of the operations into functions in my program. This can be seen with my functions called:

```
/*
 * The following are the fucntions uses throughout the program.
 * Explained in moer detail at the beginning of each functions.
 */
void calcKey(string key, string& K1, string& K2);
void SDES(string input, string K1, string K2);
string perm(string number, int perm[], int size);
string LS(string key, int shiftNum);
void split(string key, string& H1, string& H2, int size);
string XOR(string input1, string input2);
string Sbox(string input, int (*box)[4]);
string funcK(string input, string key);
string SW(string input);
bool isZeroOrOne(string input);
// End function list
```

Where all the functions related to the pdf copy the name if not closely resemble the same

For the permutations and S-boxes I used the ones specified in the lecture slides however they are clearly labelled so that they can be changed easily

```
/*
 * The following Arrays contain the permutations
 * and S-boxes that are used throughout the program.
 */
int IP [8] = {2,6,3,1,4,8,5,7};
int IPR [8] = {4,1,3,5,7,2,8,6};
int P10 [10] = {3,5,2,7,4,10,1,9,8,6};
int P8 [8] = {6,3,7,4,8,5,10,9};
int EP [8] = {4,1,2,3,2,3,4,1};
int P4 [4] = {2,4,3,1};

int S0 [4][4] = {
    {1,0,3,2},
    {3,2,1,0},
    {0,2,1,3},
    {3,1,3,2}
};

int S1 [4][4] = {
    {0,1,2,3},
    {2,0,1,3},
    {3,0,1,0},
    {2,1,0,3}
};
// End permutations and S-boxes
```

Michael Giglio
Assignment Documentation

To create the functions, it was simply a matter of translating the functions outlined in the pdf into c++ code. Each function is explained in-depth in the c++ file.

Test Plan and Results

I used the values given in the assignment brief pdf as test data for the program to test both encryption and decryption:

Input			Output		
Input	**Key**	**Instruction**	**K1**	**K2**	**Output**
00100100	1000100011	1	10100010	00011011	00001010
00001010	1000100011	2	00011011	10100010	00100100

```
S-DES Encryption/Decryption

Input:
00100100
Key:
1000100011
Do you want to:
1. Encrypt
2. Decrypt
1

Input: 00100100
K1: 10100010
K2: 00011011
Instruction: Encrypt
Output: 00001010
Program ended with exit code: 0
```

```
S-DES Encryption/Decryption

Input:
00001010
Key:
1000100011
Do you want to:
1. Encrypt
2. Decrypt
2

Input: 00001010
K1: 00011011
K2: 10100010
Instruction: Decrypt
Output: 00100100
Program ended with exit code: 0
```

To check that if the program error checking methods where working the following test data was used

Variable tested	Input	Errors with input	Program Output
givenInput	01001	Length = 5	ERROR: Length
givenInput	1001001001	Length = 10	ERROR: Length
givenInput	12312312	Contains 2,3	ERROR: Contains other number
key	01000	Length = 5	ERROR: Length
key	1000100010001	Length = 13	ERROR: Length
key	1000100031	Contains 3	ERROR: Contains other number
choice	0	0 < 1	ERROR: Not 1 or 2
choice	4	4 > 2	ERROR not 1 or 2

```
Input:
01001
ERROR: The input length must be 8
Input:
1001001001
ERROR: The input length must be 8
Input:
12312312
ERROR: Can only contain 0's or 1's
```

```
Key:
01000
ERROR: The key length must be 10
Key:
1000100010001
ERROR: The key length must be 10
Key:
1000100031
ERROR: Can only contain 0's or 1's
```

```
Do you want to:
1. Encrypt
2. Decrypt
0
ERROR: Not a valid choice
Do you want to:
1. Encrypt
2. Decrypt
4
ERROR: Not a valid choice
```

Manual

1. Start the program
2. When the program loads it will first ask you for the input
3. Enter in plaintext if you want to encrypt or cipher text if you want to decrypt
4. The program will then ask you for the key
5. Enter the key that you want to use
6. The program will then ask if you want to encrypt or decrypt
7. Enter 1 if you want to encrypt or 2 if you want to decrypt
8. The program will then perform the S-DES on your inputted data and display the results
9. Repeat from step 1 to if you want to use different inputs

NOTE: You can refer to the test result images above to use a visual aid for the steps listed

Limitations

- The user is required to rerun the program to enter in different inputs. The program will not ask if you want to restart with different inputs.
- The program needs to be restarted from the beginning if you incorrectly input the key or data. The program only checks if the length of the inputs are correct and are only made of 0's and 1's. It will not pick up that you entered 10010110 when you meant 10010101.

Michael Giglio
Assignment Documentation

Diffie-Hellman

Solution Description

For this program I used the lecture slides to translate the process of the Diffie-Hellman key exchange into c++ code.

The webpage “Primitive root of a prime number n modulo n” on the website GeeksforGeeks was used to help with identifying if g (whether manually entered or randomly generated) was a primitive root. It has been referenced in code comments when it was used.

Test Plan and Results

I used the values given in the assignment brief pdf as test data for the program to test the key exchange for manual input:

Input				Output			
p	**g**	**xa**	**xb**	**ya**	**yb**	**ka**	**ka**
11	7	3	4	2	3	5	5

```
Diffie-Hellman Key Exchange

Do you want to use:
1. Manual Input
2. Randomly Generated Input
1
Enter a value p:11
11 is a prime number using 11

Enter a value g:7
7 is a primitive root using 7

Enter a value xa:3
Enter a value xb:4
ya = 2
yb = 3
ka = 5
kb = 5
```

Michael Giglio
Assignment Documentation

Here are results of randomly generated numbers. To know whether this part of the program was working the criteria ka = kb had to be met.

Input				Output			
p	**g**	**xa**	**xb**	**ya**	**yb**	**ka**	**ka**
19	14	8	2	4	6	16	16
7	3	5	2	5	2	4	4
13	11	1	1	11	11	11	11

```
Diffie-Hellman Key Exchange

Do you want to use:
1. Manual Input
2. Randomly Generated Input
2
19 is a prime number using 19

14 is a primitive root using 14

p = 19
g = 14
xa = 8
xb = 2
ya = 4
yb = 6
ka = 16
kb = 16
```

```
Do you want to use:
1. Manual Input
2. Randomly Generated Input
2
ERROR: 46 is not a prime number
ERROR: 12 is not a prime number
7 is a prime number using 7

ERROR: 2 is not a primitive root
3 is a primitive root using 3

p = 7
g = 3
xa = 5
xb = 2
ya = 5
yb = 2
ka = 4
kb = 4
```

```
Diffie-Hellman Key Exchange

Do you want to use:
1. Manual Input
2. Randomly Generated Input
2
ERROR: 6 is not a prime number
ERROR: 44 is not a prime number
13 is a prime number using 13

11 is a primitive root using 11

p = 13
g = 11
xa = 1
xb = 1
ya = 11
yb = 11
ka = 11
kb = 11
```

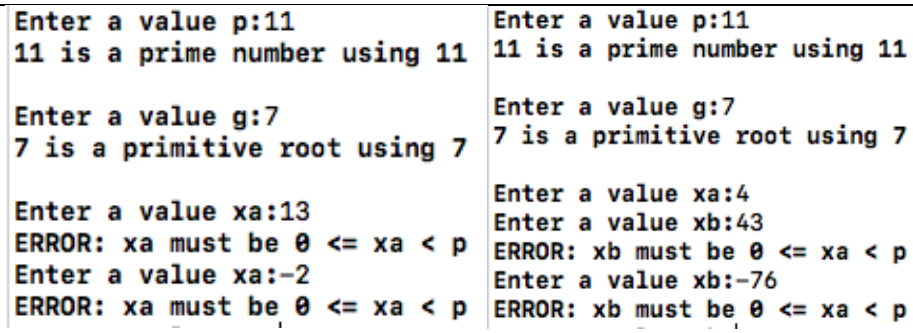
For error checking the following input was used

Variable tested	Input	Errors with input	Program Output
p	6	Not prime	ERROR: Not prime
p	10	Not prime	ERROR: Not prime
g when p = 11	5	Not primitive root	ERROR: Not primitive root
g when p = 11	4	Not primitive root	ERROR: Not primitive root
xa when p = 11	13	0 <= xa < p	ERROR: 0 <= xa < p
xa when p = 11	-2	0 <= xa < p	ERROR: 0 <= xa < p
xb when p = 11	43	0 <= xb < p	ERROR: 0 <= xb < p
xb when p = 11	-76	0 <= xb < p	ERROR: 0 <= xb < p

```
Enter a value p:6
ERROR: 6 is not a prime number
Enter a value p:10
ERROR: 10 is not a prime number
Enter a value p:
```

```
Enter a value p:11
11 is a prime number using 11

Enter a value g:5
ERROR: 5 is not a primitive root
Enter a value g:4
ERROR: 4 is not a primitive root
Enter a value g:
```



Manual

1. Start the program
2. When the program loads it will first ask you to choose manual input or randomly generated
3. Enter 1 if you want to manually enter in values or 2 if you want then randomly generated

Manual Input	Randomly generated
1. The program will ask you for p 2. Enter the value for p 3. The program will ask you for g 4. Enter the value for g 5. The program will ask you for xa 6. Enter the value for xa 7. The program will ask you for xb 8. Enter the value for xb 9. The program will then perform the necessary calculations 10. The program will then output ya,yb,ka and kb to the screen 11. Repeat from Step 1 – Start the program	1. The program will generate values for p, g, xa and xb 2. The program will then perform the necessary calculations 3. The program will then output ya,yb,ka and kb to the screen

NOTE: You can refer to the test result images above to use a visual aid for the steps listed

Limitations

- The user is required to rerun the program to enter in different inputs. The program will not ask if you want to restart with different inputs.
- When randomly generating numbers, p is limited to 30 because if xa and xb get too large when you try to do g^xa or g^xb the result is so large that it wraps around and becomes a negative. This then impacts the value of ya or yb and in turn affects ka and kb causing a mismatch.

Michael Giglio
Assignment Documentation

S-DES Code

```
//
//  main.cpp
//  S-DES
//
//  Created by Michael Giglio on 1/9/19.
//  Copyright © 2019 Michael Giglio. All rights reserved.
//

#include <iostream>

using namespace std;

/*
 * The following Arrays contain the permutations
 * and S-boxes that are used throughout the program.
 */
int IP [8] = {2,6,3,1,4,8,5,7};
int IPR [8] = {4,1,3,5,7,2,8,6};
int P10 [10] = {3,5,2,7,4,10,1,9,8,6};
int P8 [8] = {6,3,7,4,8,5,10,9};
int EP [8] = {4,1,2,3,2,3,4,1};
int P4 [4] = {2,4,3,1};

int S0 [4][4] = {
   {1,0,3,2},
   {3,2,1,0},
   {0,2,1,3},
   {3,1,3,2}
};

int S1 [4][4] = {
   {0,1,2,3},
   {2,0,1,3},
   {3,0,1,0},
   {2,1,0,3}
};
// End permutations and S-boxes

/*
 * The following are the functions uses throughout the program.
 * Explained in more detail at the beginning of each functions.
 */
void calcKey(string key, string& K1, string& K2);
void SDES(string input, string K1, string K2);
string perm(string number, int perm[], int size);
string LS(string key, int shiftNum);
void split(string key, string& H1, string& H2, int size);
string XOR(string input1, string input2);
string Sbox(string input, int (*box)[4]);
string funcK(string input, string key);
string SW(string input);
bool isZeroOrOne(string input);
// End function list

// Main function
int main(int argc, const char * argv[]) {
   // Title message.
   cout << "S-DES Encryption/Decryption" << endl << endl;
```

```
/*
 * Get the input that is going to be encrypted or decrypted.
 * Checks that the input is 8 bits long and are only 0's and 1's
 */
string givenText = "";
while (givenText.length() != 8 || !isZeroOrOne(givenText)) {
   cout << "Input: " << endl;
   cin >> givenText;

   if (givenText.length() != 8)
      cout << "ERROR: The input length must be 8" << endl;
}

/*
 * Get the key that is used.
 * K1 and K2 are created to hold the two separate keys needed during the process.
 * Checks that the key is 10 bits long and are only 0's and 1's
 */
string key = "", K1, K2;
while (key.length() != 10 || !isZeroOrOne(key)) {
   cout << "Key: " << endl;
   cin >> key;

   if (key.length() != 10)
      cout << "ERROR: The key length must be 10" << endl;
}

// Calculate the two keys from the original.
calcKey(key, K1, K2);

/*
 * Ask the user if they want to encrypt or decrypt.
 * Makes sure that the user enters a valid choice
 */
int choice = 0;
while (choice < 1 || choice > 2) {
   cout << "Do you want to: " << endl;
   cout << "1. Encrypt" << endl;
   cout << "2. Decrypt" << endl;

   // Get the users choice.
   cin >> choice;

   if (choice < 1 || choice > 2) {
      cout << "ERROR: Not a valid choice" << endl;
   }
}

// Display the input parameters.
cout << "\nInput: " << givenText << endl;

/* If else statement to encrypt or decrypt based on user selection.
 * Note that the only difference between encryption and decryption
 * is that K1 and K2 are switched. This is so that they are used in
 * the reverse order to encryption.
 */
if (choice == 1) {
   // Display the instruction selected and encrypt.
   cout << "K1: " << K1 << endl;
   cout << "K2: " << K2 << endl;
   cout << "Instruction: Encrypt" << endl;
   SDES(givenText, K1, K2);
} else {
   // Display the instruction selected and decrypt.
```

```
        cout << "K1: " << K2 << endl;
        cout << "K2: " << K1 << endl;
        cout << "Instruction: Decrypt" << endl;
        SDES(givenText, K2, K1);
    }

}

/*
 * ----- calcKey() -----
 *
 * -- Description
 * This function is used to calculate the two keys that will be used
 * in the encryption and decryption process. Using the method outlined
 * in the pdf.
 *
 * -- Inputs
 * Three inputs are taken by this function:
 * $ The key that was inputted by the user in the main() function.
 * $ The address to the K1 variable created in the main() function.
 * $ The address to the K2 variable created in the main() function.
 *
 * -- Outputs
 * This function outputs two keys that are stored in the K1 and K2
 * variables passed in by referance.
 */
void calcKey(string key, string& K1, string& K2) {
    // Permute all numbers of the key using the P10 array outlined above.
    key = perm(key, P10, 10);

    // Leftshift the key by one.
    key = LS(key, 1);

    /*
     * Select only 8 elements of the key using the P8 array outlined above.
     * The result is stored in the K1 varaible as the first key.
     */
    K1 = perm(key, P8, 8);

    // Leftshift the key by 2.
    key = LS(key, 2);

    /*
     * Select only 8 elements of the key using the P8 array outlined above.
     * The result is stored in the K2 varaible as the second key.
     */
    K2 = perm(key, P8, 8);
}
// end calcKey()

/*
 * ----- SDES() -----
 *
 * -- Description
 * This function is used to carry out the process of SDES as outlined
 * in the pdf. The output of the SDES will then be displayed to the screen.
 *
 * -- Inputs
 * Three inputs are taken by this function:
 * $ The input value to be encrypted or decrypted that was entered
 * by the user in the main() function.
 * $ The first key created by the calcKey() function.
 * $ The second key created by the calcKey() function.
 *
```

```
 * -- Outputs
 * The output of this function will be the encrypted ciphertext or
 * decrypted plaintext outputted to the screen.
 */
void SDES(string input, string K1, string K2) {

   // Permute the input using the IP array outlined above.
   input = perm(input, IP, 8);

   // Perform the k function on the input using the first key, K1.
   input = funcK(input, K1);

   // Switch the two halves of the input.
   input = SW(input);

   // Perform the k function on the input again using the second key, K2.
   input = funcK(input, K2);

   // Permute the input using the IPR array outlined above.
   input = perm(input, IPR, 8);

   // Outpute the result to the console.
   cout << "Output: " << input << endl;
}
// end SDES()

/*
 * ----- funcK() -----
 *
 * -- Description
 * This function is used to carry out the process of function k as outlined
 * in the pdf.
 *
 * -- Inputs
 * Two inputs are taken by this function:
 * $ The input value to be that function k will be performed on.
 * $ The key that will be used in a XOR function during the process.
 *
 * -- Outputs
 * The output of this function will be the an ecrypted first half of the
 * input that will be returned to the SDES() function.
 */
string funcK(string input, string key) {
   //Create variables to hold the different halfs of the original input.
   string half1, half2, half2Saved;

   //Splite the input and store each half in different varaibles.
   split(input, half1, half2, 8);

   /*
    * Copy half2 variable into the half2Saved variable so that its
    * orignal value can be preserved for later
    */
   half2Saved = half2;

   // Permute the and extend half2 using the EP array outlined above
   half2 = perm(half2, EP, 8);

   // XOR half2 with the key value.
   half2 = XOR(half2, key);

   // Create another two variables for when half2 is split again
   string half21, half22;
```

```
    // Splite half2 again and store the two values in separate variables
    split(half2, half21, half22, 8);

    // Perform the Sbox() function on both halves using the Sboxes outlined above
    half21 = Sbox(half21, S0);
    half22 = Sbox(half22, S1);

    // Clear half2 to receive new input
    half2 = "";

    // Combine half21 and half22 and store it in half2
    half2 += half21;
    half2 += half22;

    // Permute half2 using the P4 array outlined above
    half2 = perm(half2, P4, 4);

    // XOR half1 and half2 together
    half2 = XOR(half1, half2);

    // Clear the input variable to receive a new value
    input = "";

    // Combine half2 and half2Saved and store it in the input variable
    input += half2;
    input += half2Saved;

    // Return the input
    return input;
}
// end funcK()

/*
 * ----- XOR() -----
 *
 * -- Description
 * This function is used to perform a XOR on two binary strings of the
 * SAME length.
 *
 * -- Inputs
 * Two inputs are taken by this function:
 * $ The first string to be XORed.
 * $ The second string to be XORed.
 *
 * -- Outputs
 * The output of this function will be the result of the XOR operation.
 */
string XOR(string input1, string input2) {
    // Create a string to hold the result.
    string temp;

    /*
     * For loop that loops through both strings and compares
     * the value of each string at element i.  If they are
     * the same then a 0 is added to the temp variable.  Otherwise
     * a 1 is added to the variable
     */
    for (int i = 0; i < input1.size(); i++) {
        if (input1[i] == input2[i]) {
            temp += "0";
        } else {
            temp += "1";
        }
    }
```

```
    // Return the result
    return temp;
}
// end XOR()

/*
 * ----- perm() -----
 *
 * -- Description
 * This function is used to permute a given string using a given a
 * permutation order.
 *
 * -- Inputs
 * Three inputs are taken by this function:
 * $ The input value to be permuted.
 * $ The array that contains the permutation order.
 * $ The size of the permutation array.
 *
 * -- Outputs
 * The output of this function will be the permuted input string.
 */
string perm(string number, int perm[], int size) {
    // Create a temporary value to hold the result
    string temp;

    /*
     * A for loop that takes the number at the element specified
     * by the permutation order.  -1 as added to the end because
     * array indexing starts at 0 whereas the permutation array
     * values start at 1.  Therefore, 1 has to be taken away from
     * the permutation number to stop the program taking the wrong
     * bit.
     */
    for (int i = 0; i < size; i++) {
        temp += number[perm[i]-1];
    }

    // Return the result.
    return temp;
}
// end perm()

/*
 * ----- Sbox() -----
 *
 * -- Description
 * This function is used to carry out the process of the Sbox in the SDES
 * process.
 *
 * -- Inputs
 * Two inputs are taken by this function:
 * $ The input value to be passed through the Sbox.
 * $ The Sbox to be used.
 *
 * -- Outputs
 * The output of this function will be two bits returned to funcK().
 */
string Sbox(string input, int (*box)[4]) {
    // Create two strings to hold the column and row values
    string row, col;

    // Take the first and fourth bit and store them in the row variable
    row += input[0];
```

```
    row += input[3];

    // Take the second and third bit and store them in the col variable
    col = input.substr(1,2);

    // Create two int variables to hold the row and column values
    int r , c;

    /*
     * If statements used to store the index of the row value to access
     * the Sbox value.
     */
    if (row == "00")
        r = 0;
    else if (row == "01")
        r = 1;
    else if (row == "10")
        r = 2;
    else
        r = 3;

    /*
     * If statements used to store the index of the column value to access
     * the Sbox value
     */
    if (col == "00")
        c = 0;
    else if (col == "01")
        c = 1;
    else if (col == "10")
        c = 2;
    else
        c = 3;

    // Get the value stored in the Sbox located at the specified cell
    int result = box[r][c];

    // Turns the number retrieved back into binary form and returns it
    switch (result) {
        case 0:
            return "00";
            break;

        case 1:
            return "01";
            break;

        case 2:
            return "10";
            break;

        case 3:
            return "11";
            break;
        default:
            return "00";
            break;
    }
}
// end Sbox()

/*
 * ----- SW() -----
 *
```

```
 * -- Description
 * This function is used to switch two halves of a given input around.
 *
 * -- Inputs
 * One input is taken by this function:
 * $ The input value to be switched.
 *
 * -- Outputs
 * The output of this function will be the input value with its halves switched.
 */
string SW(string input) {
   // Create two temp variables to hold the halves
   string temp1, temp2;

   // Split the input and store them in the temp variables
   split(input, temp1, temp2, 8);

   // Clear the input to hold the result
   input = "";

   // Store the two halves in the reverse order in the input variable.
   input += temp2;
   input += temp1;

   // Return the switched input
   return input;
}
//end SW()

/*
 * ----- LS() -----
 *
 * -- Description
 * This function is used to carry out the Left shift process.
 *
 * -- Inputs
 * Two inputs are taken by this function:
 * $ The key value to be shifted.
 * $ The amount of times to be shifted.
 *
 * -- Outputs
 * The output of this function will be a single string
 * with both halves shifted.
 */
string LS(string key, int shiftNum) {
   // Create two variables to hold the split input
   string half1, half2;

   // Split the key into its halves and store them in the variables
   split(key, half1, half2, 10);

   /*
    * For loop to shift the value to the left.
    * Repeats the amount of times specified by the shiftNum variable
    */
   for (int i = 0; i < shiftNum; i++) {
      rotate(half1.begin(), half1.begin()+1, half1.end());
      rotate(half2.begin(), half2.begin()+1, half2.end());
   }

   // Clear key to hold the new values
   key = "";

   // Combine the new halves back into a single string
```

```
    key += half1;
    key += half2;

    // Return the shifted key.
    return key;
}
// end LS()

/*
 * ----- split() -----
 *
 * -- Description
 * This function is used to split a given string in half.
 *
 * -- Inputs
 * Four inputs are taken by this function:
 * $ The input value to be split.
 * $ A variable to store the first half of the string.
 * $ A variable to store the second half of the string.
 * $ The size of the string to be split.
 *
 * -- Outputs
 * The output of this function will be two variables each containing
 * a half of the orignal string.
 */
void split(string key, string& H1, string& H2, int size) {

    /*
     * A for loop that repeats for the size of the original string.
     * If statement is used to check which half of the input i is
     * in and stores the i element of the input in either H1 or H2
     * variable.
     */
    for (int i = 0; i < size; i++) {
        if (i > ((size/2)-1)) {
            H2 += key[i];
        } else {
            H1 += key[i];
        }
    }
}
// end spit()

/*
 * ----- isZeroOrOne() -----
 *
 * -- Description
 * This function is used to check that the inputs entered are made up
 * of only 0's and 1's
 *
 * -- Inputs
 * One input is taken by this function:
 * $ The string that needs to be checked.
 *
 * -- Outputs
 * This function will return false if any bits are neither a 0 or 1 or
 * will return true if all bits are either 0's or 1's.
 */
bool isZeroOrOne(string input) {

    /*
     * A for loop that looks at every bit of input to check that it is
     * a 0 or 1.
     */
```

```
    for (int i = 0; i < input.length(); i++) {
        if (!(input[i] == '0' || input[i] == '1')) {
            cout << "ERROR: Can only contain 0's or 1's" << endl;
            return false;
        }
    }

    return true;
}
//end isZeroOrOne()
```

Diffie-Hellman Code

```
//
//  main.cpp
//  Diffie-Hellman
//
//  Created by Michael Giglio on 3/9/19.
//  Copyright © 2019 Michael Giglio. All rights reserved.
//

#include <iostream>
#include <cmath>
#include <ctime>
#include <cstdlib>
#include <set>

using namespace std;

/*
 * Methods that are used in this program.
 * They are explained down below
 */
void getUserInput(long long int& p,long long int& g,long long int& xa,long long int& xb);
void keyExchange(long long int p,long long int g,long long int xa,long long int xb);
void generateNumb(long long int& p,long long int& g,long long int& xa,long long int& xb);
bool isPrime(long long int p);
bool isPrimeRoot(long long int p,long long int g);
void primeFactor(set<long long int> &s,long long int phi);
long long int power(long long int g, unsigned long long int y,long long int p);
// end methods

// Main function
int main(int argc, const char * argv[]) {
    /*
     * This line ensures that the numbers generated by
     * are random every time.
     */
    srand(time(NULL));

    // Title.
    cout << "Diffie-Hellman Key Exchange" << endl << endl;

    /*
     * Allows the user to select if they want to manually
     * enter values or generate random numbers.
     * Makes sure that the choice entered is valid.
     */
    int choice = 0;
    while (choice < 1 || choice > 2) {
        cout << "Do you want to use: " << endl;
        cout << "1. Manual Input" << endl;
```

```
        cout << "2. Randomly Generated Input" << endl;

        cin >> choice;

        if (choice < 1 || choice > 2) {
            cout << "ERROR: Is not a valid choice" << endl;
        }
    }

    /*
     * Varaibles that are used in the porgram.
     * long long ints are used because when using int the powers
     * where sometimes so big that they turned negative.
     */
    long long int p, g, xa, xb;

    // Allows manual input or generates number based on choice
    if (choice == 1) {
        getUserInput(p,g,xa,xb);
    } else {
        generateNumb(p, g, xa, xb);
    }

    //Perform key exchange
    keyExchange(p,g,xa,xb);
}

/*
 * ----- getUserInput() -----
 *
 * -- Description
 * This function is used get the users input for the 4 variables
 * p, g, xa and xb
 *
 * -- Inputs
 * Four inputs are taken by this function:
 * $ The memory address of variable p.
 * $ The memory address of variable g.
 * $ The memory address of variable xa.
 * $ The memory address of variable xb.
 *
 * -- Outputs
 * This function outputs the four variables each storing numbers
 * defined by the user.
 */
void getUserInput(long long int& p,long long int& g,long long int& xa,long long int& xb) {

    /*
     * Do while loops to ensure that the values entered by user comply
     * with the rules for a Diffie-Hellman key exchange.
     *
     * p - has to be prime
     * g - has to be a primitive root of p
     * xa - must be a value from 0 to p-1
     * xb - must be a value from 0 to p-1
     */

    // Get p
    do {
        cout << "Enter a value p:";
        cin >> p;
    } while (p <= 1 || !isPrime(p));

    // Get g
```

```
    do {
        cout << "Enter a value g:";
        cin >> g;
    } while (g >= p || !isPrimeRoot(p, g));

    // Get xa
    do {
        cout << "Enter a value xa:";
        cin >> xa;

        if (xa < 0 || xa > (p-1)) {
            cout << "ERROR: xa must be 0 <= xa < p" << endl;
        }
    } while (xa < 0 || xa > (p-1));

    // Get xb
    do {
        cout << "Enter a value xb:";
        cin >> xb;

        if (xb < 0 || xb > (p-1)) {
            cout << "ERROR: xb must be 0 <= xa < p" << endl;
        }
    } while (xb < 0 || xb > (p-1));
}
// end getUserInput()

/*
 * ----- keyExchange() -----
 *
 * -- Description
 * This function is used to perform the key exchange using the
 * variables defined by the user or randomly generated
 *
 * -- Inputs
 * Four inputs are taken by this function:
 * $ The variable p.
 * $ The variable g.
 * $ The variable xa.
 * $ The variable xb.
 *
 * -- Outputs
 * This function outputs the four variables to the screen
 * $ The variable ya.
 * $ The variable yb.
 * $ The variable ka.
 * $ The variable kb..
 */
void keyExchange(long long int p,long long int g,long long int xa,long long int xb) {

    /*
     * Varaibles that are used in the porgram.
     * long long ints are used because when using int the powers
     * where sometimes so big that they turned negative.
     */
    long long int ya, yb, ka, kb;

    // ya = g^xa mod p
    ya = pow(g, xa);
    ya %= p;

    // yb = g^xb mod p
    yb = pow(g, xb);
    yb %= p;
```

```
    // ka = yb^xa mod p
    ka = pow(yb, xa);
    ka %= p;

    // ka = yb^xb mod p
    kb = pow(ya, xb);
    kb %= p;

    // Display the resutls
    cout << "ya = " << ya << endl;
    cout << "yb = " << yb << endl;
    cout << "ka = " << ka << endl;
    cout << "kb = " << kb << endl;
}
// end keyExchange()

/*
 * ----- generateNumb() -----
 *
 * -- Description
 * This function is similar to the getUsersInput() function except
 * p, g, xa and xb are randomly genereated using rand() while also
 * following the rules need to perform the Diffie-Hellman key exchange
 *
 * -- Inputs
 * Four inputs are taken by this function:
 * $ The memory address of variable p.
 * $ The memory address of variable g.
 * $ The memory address of variable xa.
 * $ The memory address of variable xb.
 *
 * -- Outputs
 * This function outputs the four variables each storing numbers
 * generated by rand.
 */
void generateNumb(long long int& p,long long int& g,long long int& xa,long long int& xb) {

    /*
     * Do while loops to ensure that the values entered by user comply
     * with the rules for a Diffie-Hellman key exchange.
     *
     * p - has to be prime
     * g - has to be a primitive root of p and g < p
     * xa - must be a value from 0 to p-1
     * xb - must be a value from 0 to p-1
     */

    // Generate p
    do {
        /* p is limited to numbers between 0 - 30 because
         * if it is too large it can cause issues with numbers
         * wrapping around to negatives.
         */
        p = (rand() % 30);
    } while (p <= 5 || !isPrime(p));

    // Generate g
    do {
        g = (rand() % p) + 2;
    } while (g >= p || !isPrimeRoot(p, g));

    // Generate xa
    do {
```

```
      xa = (rand() % p);
   } while (xa < 0 || xa > (p-1));

   // Generate xb
   do {
      xb = (rand() % p);
   } while (xb < 0 || xb > (p-1));

   // Display the resutls
   cout << "p = " << p << endl;
   cout << "g = " << g << endl;
   cout << "xa = " << xa << endl;
   cout << "xb = " << xb << endl;
}
// end generateNumb()

/*
 * ----- isPrime() -----
 *
 * -- Description
 * This function checks to see if the passed varaible
 * is a prime number.
 *
 * -- Inputs
 * One input is taken by this function:
 * $ The value of variable p.
 *
 * -- Outputs
 * This function returns false if p is not a prime
 * number and true if it is.
 */
bool isPrime(long long int p) {

   /*
    * for loop that checks number between 2 and p/2
    * to make sure that p is a prime number
    * if there isnt any remainder for one of the numbers
    * then p cannot be prime.
    *
    * NOTES:
    * $ i starts at 2 because 1 is a multiple of all numbers and anything
    * divided by 0 is undefined.
    * $ The loop runs to p/2 because of the axb = bxa rule for
    * multiplication.
    */
   for (int i = 2; i <= p/2; i++) {
      if (p % i == 0) {
         cout << "ERROR: " << p << " is not a prime number" << endl;
         return false;
      }
   }

   // Specify the number being used and then return true.
   cout << p << " is a prime number using " << p << endl << endl;
   return true;
}
// end isPrime()

/*
 * ----- isPrimeRoot() -----
 *
 * -- Description
 * This function checks to see if the passed varaible
 * is a primitive root of p
```

```
 *
 * -- Inputs
 * Two inputs are taken by this function:
 * $ The value of variable p.
 * $ The value of variable g.
 *
 * -- Outputs
 * This function returns false if p is not a primative root
 * and true if it is.
 *
 * -- NOTE
 * The below function was inspired by the solution posted on
 * the website GeeksforGeeks.
 *
 * The webpage is titled "Primitive root of a prime number n modulo n"
 * and is avaialble at
 * https://www.geeksforgeeks.org/primitive-root-of-a-prime-number-n-modulo-n/
 */
bool isPrimeRoot(long long int p,long long int g) {

   // Create a set to hold the prime factos of phi.
   set<long long int> s;

   // A boolean used to determine what to return.
   bool isRoot = true;

   // Calculate the value of phi.
   long long int phi = p-1;

   // Find all the prime factors of phi and store them in the set.
   primeFactor(s,phi);

   /*
    * Iterator that goes through the set and checks if
    * there is a power with value of 1.
    */
   for (auto it = s.begin(); it != s.end(); it++) {

      // Check if g^(phi/primefactors) mod n is equal to one.
      if (power(g, phi/(*it), p) == 1) {
         isRoot = false;
         break;
      }
   }

   // Return false if g is not a prime root or return true if it is.
   if (isRoot == false) {
      cout << "ERROR: " << g << " is not a primitive root" << endl;
      return false;
   } else {
      cout << g << " is a primitive root using " << g << endl << endl;
      return true;
   }
}
// end isPrimeRoot()

/*
 * ----- isPrimeRoot() -----
 *
 * -- Description
 * This function finds all the prime factors of a number
 * and stores them in a set
 *
 * -- Inputs
```

```
 * Two inputs are taken by this function:
 * $ The memory address of a set.
 * $ The value of variable phi.
 *
 * -- Outputs
 * This function fills the set with primitive roots
 *
 * -- NOTE
 * The below function was inspired by the solution posted on
 * the website GeeksforGeeks.
 *
 * The webpage is titled "Primitive root of a prime number n modulo n"
 * and is avaialble at
 * https://www.geeksforgeeks.org/primitive-root-of-a-prime-number-n-modulo-n/
 */
void primeFactor(set<long long int> &s,long long int phi) {

   /*
    * Insert the number 2 into the set everytime it divides into phi.
    * We can continually insert 2 into a set without an repercustions because
    * sets will automatically disreguard any duplicate value added to it.
    */
   while (phi%2 == 0) {
      s.insert(2);
      phi /= 2;
   }

   /*
    * phi must be odd that is why we +=2.
    * For each number in the for loop divide see if it
    * fits into phi with no remainder if it does store i in the set
    * divide phi by i and repeat.
    */
   for (int i = 3; i <= sqrt(phi); i += 2) {

      /* Continue to divide phi by i and store it in the set while the
       * remainder equals 0
       */
      while (phi%i == 0) {
         s.insert(i);
         phi /= i;
      }
   }

   // Used to handle a case when n is a prime number greater than two
   if(phi > 2)
      s.insert(phi);
}
//end primeFactor()

/*
 * ----- power() -----
 *
 * -- Description
 * This function calculates (g^n)%p
 *
 * -- Inputs
 * Three inputs are taken by this function:
 * $ The value of variable g.
 * $ The value of variable y.
 * $ The value of variable p.
 *
 * -- Outputs
 * This function returns (g^n)%p
```

Michael Giglio
Assignment Documentation

```
 *
 * -- NOTE
 * The below function was inspired by the solution posted on
 * the website GeeksforGeeks.
 *
 * The webpage is titled "Primitive root of a prime number n modulo n"
 * and is available at
 * https://www.geeksforgeeks.org/primitive-root-of-a-prime-number-n-modulo-n/
 */
long long int power(long long int g, unsigned long long int y,long long int p) {

   // Initialise the result
   long long int result = 1;

   // Update g if it is more than or equal to p
   g %= p;


   while (y > 0) {

      // If y is odd then multiply x with the result
      if (y%2 == 1) {
         result = (result * g) % p;
      }

      // y must be even now
      y /= 2;
      g = (g*g) % p;
   }

   return result;
}
// end power()
```