CS 359 – Programming Paradigms

PEX 1 – Pontifex Encryption

Preliminary Submission Due: Taps, Lesson 9, Friday, 31 August

Final Submission Due: Taps, Lesson 11, Friday, 7 September

Help Policy:

AUTHORIZED RESOURCES: Any, except another cadet's program.
NOTE:

- Never copy another person's work and submit it as your own.
- Do not jointly create a program unless explicitly allowed.
- You must document all help received from sources other than your instructor or instructor-provided course materials (including your textbook).
- **DFCS will recommend a course grade of F for any cadet who egregiously violates this Help Policy or contributes to a violation by others.**

Documentation Policy:

- You must document all help received from any source other than your instructor.
- The documentation statement must explicitly describe WHAT assistance was provided, WHERE on the assignment the assistance was provided, and WHO provided the assistance.
- If no help was received on this assignment, the documentation statement must state "NONE."
- If you checked answers with anyone, you must document with whom on which problems. You must document whether or not you made any changes, and if you did make changes you must document the problems you changed and the reasons why.
- **Vague documentation statements must be corrected before the assignment will be graded and will result in a 5% deduction on the assignment.**

Turn-in Policies:

- On-time turn-in is at taps on the M-day of the lesson indicated on the course schedule.
- Late penalties accrue at a rate of 25% per 24-hour period past the on-time turn-in date and time. The late penalty is a cap on the maximum grade that may be awarded for late work.
- There is no early turn-in bonus or extra credit for this assignment.

1. OBJECTIVES

- Expand your range of programming language competence
- Be able to create, compile, link, and execute a C program
- Be able to use command-line arguments in a C program
- Be able to use a dynamic, linked data structure in a C program
- Be able to use both pass-by-value and pass-by-reference parameter passing in a C program
- Be able to learn and implement a stream cipher encryption algorithm

2. BACKGROUND

Your task for this assignment is to implement a solitaire encryption algorithm known as Pontifex. It was designed by Bruce Schneier and featured in Neal Stephenson's novel *Cryptonomicon*. Details of the algorithm can be found on this web page: http://www.schneier.com/solitaire.html.

Suppose Alice and Bob wish to exchange a secret message. One way for them to do this is to agree on a series of random numbers to be used as a *key*. If these numbers are truly random and the key is kept secret, they can communicate securely by adding those numbers to the message *modulo* 26 to encrypt and by subtracting *modulo* 26 to decrypt. Modulo 26 just means that if you get a number greater than or equal to 26, you subtract 26 from it. Subtracting *mod* 26 is like adding, except that when you get a negative number you add 26 to it. This should be very familiar from Math 340 and/or CS 431.

For example, suppose Alice and Bob agree on the key "25 13 12" and Alice wants to send Bob the message "CAT" (for simplicity we assume that the message consists only of upper case letters and contains no white space). Alice converts the message to numbers based on alphabetical ordering to obtain "2 0 19" (since this a computer science class, we count from zero). She then adds the key *mod* 26 to obtain "1 13 5", and transmits this to Bob. When Bob receives "1 13 5", he subtracts the key *mod* 26 and obtains "2 0 19" which converted back to letters equals "CAT", the original message.
DO NOT PROCEED FURTHER UNTIL YOU UNDERSTAND AND BELIEVE THIS EXAMPLE!

It is a theorem of mathematics that if the key is truly random, kept secret, and never reused, this system is perfectly secure. That is, the enemy gains zero information if they obtain access the encrypted message. This system is known as a "one-time pad". You may have seen it in spy thrillers and war movies from time to time.

"So," you think, "that's very nice, but how do Alice and Bob agree on a series of random numbers to use for the key?" Funny you should ask. They could carry computers around with encryption software, but that's cumbersome and suspicious and Alice and Bob are trying not to draw attention. A solitaire cryptographic algorithm, invented by crypto-guru Bruce Schneier, enables two people to generate secure keys from a perfectly ordinary and innocuous deck of cards. If you're caught with them, you can always claim you just use them for playing solitaire and hustling middle school kids out of their lunch money.

Once Alice and Bob have each put their decks in some identical order (there are various secure ways to do this, the details are not important here), they can use the Pontifex algorithm to generate random and highly secure sequences of identical random numbers to communicate securely. You will implement this algorithm in C.

The preliminary exercise will focus on creating the deck of cards and the rest of the programming exercise will focus on implementing the Pontifex algorithm.

Included with this program description is an executable solution to the programming exercise. You should run it several times and make your program emulate its functionality as closely as possible.

3. PRELIMINARY EXERCISE

Using the provided files as a starting point, implement a deck of cards as a circular, doubly-linked list of `Card` structures. That is, each `Card` structure will have a reference to the next and previous card in the deck. The top card's previous reference will be to the bottom card and the bottom card's next reference will be to the top card. (The circular nature of the list will be handy when implementing the Pontifex algorithm later.) The `number` field in each `Card` structure will be a unique value between 1 and `NUM_CARDS`, inclusive.

Each of the functions you will need to implement includes a comment describing what the function does and how it should accomplish this task. **Read these comments carefully and be sure your code is satisfying the requirements.** In particular, you will need to write the `new_deck`, `shuffle_deck`, `show_deck`, `kced_wohs`, `write_deck_to_file`, and `read_deck_from_file` functions. In `main.c`, implement the branches of the `main` function to create a new deck with all cards in ascending order, to create a new randomly shuffled deck, and to test reading a deck from a file.

To shuffle the deck, you must rearrange pointers. You **may not** change the values stored in the number field of the `Card` structures. To "shuffle" the deck, you should repeatedly remove the top card from the old deck and insert it into a random location in a "new" deck. Again, you must accomplish this using existing Card structures and rearranging pointers.

4. PRELIMINARY EXERCISE – HELPFUL HINTS

In C, dynamic storage is allocated with the `malloc( n )` function which returns a pointer to the allocated storage. The argument `n` is the number of bytes to be dynamically allocated. The value of `n` can be determined with the `sizeof` function. Thus, a new `Card` structure is created as follows:

```
   cardPtr card = (cardPtr) malloc( sizeof(struct Card) );
```

When shuffling the deck, each time a card is removed from the old deck and inserted into the new deck, you will need to ensure all of the `next` and `prev` pointers are updated so both decks remain circular.

C has a built-in random number function named `rand()` which returns a random int between `0` and the system defined constant `RAND_MAX`. Thus, to generate a random number between `0` and `NUM_CARDS`:

```
   int r = rand() % NUM_CARDS;
```

Prior to using the above line of code, you must seed the random number generator. Using the system clock to do this results in reasonably random numbers:

```
   srand( (int) time( NULL ) );
```

Note: The call to `srand` must only execute once. That is, it should **not** be inside a loop.

The functions declared and defined in `files.h` and `files.c` can be used to open input and output files. **However, be sure to close any files that have been opened once you are finished with them.** Files are closed with a call to `fclose( filePtr )`. Data is read from/written to files using the `fscanf()` and `fprintf()` functions, which are very similar to `scanf()` and `printf()`.

The `kced_wohs` function shows a deck in reversed order (get it?). This is not required at any point in the main program, but you should do this at various times during development to ensure all pointers are set.

5. Preliminary Submission Requirements

- **Fill in your name at the top of all provided source files!**
- **Fill in your *Prelim* documentation statement at the top of main.c!**
- Change the name of the project to be your own last name. **Accomplish this** by right-clicking on the project in the Project Explorer window **within Eclipse** and choosing rename.
- Before submitting, right-click on the project in Eclipse's Project Explorer and click Clean Project.
- Use the website to submit a **single zip file** of your entire project. **Do not email your project to me!**

6. Programming Exercise

Implement the remaining encryption and decryption functionality. You should assume all messages to be encrypted or decrypted include only upper-case characters 'A' through 'Z' with no white space.
If your solution to the preliminary exercise is not working, you will need to fix that first.

You have considerable freedom in how you design your solution; however, the required functionality of the main program should give you some guidance into reasonable design decisions on decomposition and use of subprograms. A good solution that shows an understanding of decomposition will not need global variables (variables with a scope of more than one subprogram). Therefore full credit will be awarded only for solutions that do not use them. If you find yourself using global variables, it means you ~~are just plain lazy~~ do not yet have a complete mastery of program decomposition and parameter passing. If that's the case, come see your instructor. It's what we're here for.

All deck manipulation must be done using pointers. You **may not** change the values stored in the number field of the Card structures.

7. Programming Exercise – Helpful Hints

You will want to invest a considerable amount of time studying Schneier's encryption algorithm, including executing the algorithm by hand with an actual deck of cards, to be sure you understand it.

Any deck manipulation function that could result in a different card being on top of the deck will need to return a pointer to the top of the deck and will be called as follows:

```
deck = move_jokers( deck );       or          deck = triple_cut( deck );
```

Many times the top of the deck will not actually change, but when it does this assignment is necessary.

8. Programming Exercise Submission Requirements

- **Fill in your name at the top of all provided source files!**
- **Fill in your *Pex* documentation statement at the top of main.c!**
- Change the name of the project to be your own last name. **Accomplish this** by right-clicking on the project in the Project Explorer window **within Eclipse** and choosing rename.
- Before submitting, right-click on the project in Eclipse's Project Explorer and click Clean Project.
- Use the website to submit a **single zip file** of your entire project. **Do not email your project to me!**

CS 359 – PEX 1 Prelim – Grade Sheet Name:________________________

Criteria		Points Earned	Points Available
Creates a new deck			**6**
Shuffles a deck			**6**
Shows a deck			**4**
Shows a deck reversed			**2**
Writes a deck to a file			**4**
Reads a deck from a file			**4**
Implements appropriate portions of main			**4**
	Subtotal:		**30**
Adjustments	**All code meets specified standards:**		**– 3**
	Vague/Missing Documentation:		**– 2**
	Submission Requirements Not Followed:		**– 2**
	Late Penalties:		**25/50/75%**
	Total w/adjustments:		

Comments from Instructor:

CS 359 – PEX 1 – Grade Sheet Name:_________________________

Criteria		Points Earned	Points Available
Encrypts a message (full credit or zero credit)			**6**
Decrypts a message (full credit or zero credit)			**6**
Able to display first 8 keys in key sequence			**6**
Able to move jokers and show resulting deck			**6**
Able to perform triple cut and show resulting deck			**6**
Able to perform count cut and show resulting deck			**6**
Able to apply complete Pontifex algorithm and show resulting deck			**6**
Able to find the output card in a given deck			**6**
Overall quality of software design (functional decomposition with appropriately named and sized subprograms, multiple files with logically grouped subprograms, main function not too long, no global variables, use of named constants instead of "magic" numbers, etc.)			**12**
	Subtotal:		**60**
Adjustments	**All code meets specified standards:**		**– 6**
	Vague/Missing Documentation:		**– 3**
	Submission Requirements Not Followed:		**– 3**
	Late Penalties:		**25/50/75%**
	Total w/adjustments:		

Comments from Instructor: