**Project 5 Report**

**Notable Obstacles**

1. Creating the map function comparing only corresponding words between simplifiedCrib and line, rather than consecutive corresponding chars was fairly important to maintain. Certain zyBook excerpts helped me tweak my program and overcome this obstacle.
2. I forgot to reset the array values to '\0' while testing map1 and map2. This was not immediately obvious to me.
3. Designing test data to ensure functions worked properly was not easy on this project. It was difficult to build an all-encompassing set of test data.

**Pseudocode**

**Function duplicateLine(const char \*&source, char \*destination)**

duplicate chars from source onto destination until '\n' or '\0' is reached

increment source once more upon reaching '\n'

**Function simplify(const char source[], char destination[])**

repeatedly:

x = the next char in source

if x is alphabetic:

if destination will grow bigger than MAX\_LENGTH:

return -1

append lowercase of x to destination

else if there isn't already a space at the end of destination:

append ' ' to destination

remove any trailing spaces

return the length of destination

**Function boundaryTest(const char line[], int lineLength, int start, int subLength)**

start and subLength represents a substring within line

if the substring is preceded or followed by a char other than ' ', assuming

that a preceding or following char exists:

return false

else

return true

**Function map(const char \*sub, const char \*crib, int len, char map1[], char map2[])**

reset the mapping arrays map1 and map2

for each pair of chars x, y in sub and crib respectively:

if x, y are both alphabetic:

if neither c nor p maps to anything:

record the mapping x-y

else if x-y contradicts an existing mapping:

return false

else if c and p aren't both spaces, i.e. the word boundaries in sub

and crib doesn't match:

return false

return true

**Function decrypt(const char ciphertext[], const char crib[])**

simplify crib and check its validity

cribLength = the length of crib

for each line in ciphertext:

simplify the line

for each substring of length cribLength in the line:

ensure that the substring starts and ends on word boundaries

if the substring is a possible encryption of crib:

print the deciphered text

return true

return false

**Test Cases**

**Function duplicateLine(const char \*&source, char \*destination)**

| **Reason** | **source** | **source** | **destination** |
| --- | --- | --- | --- |
| Typical case | "hello\nworld" | "world" | "hello" |
| Final line ending with \n | "hello\n" | "" | "hello" |
| Final line ending with \0 | "world" | "" | "world" |
| Empty line ending with \n | "\n" | "" | "" |
| Empty line ending with \0 | "" | "" | "" |

**Function simplify(const char source[], char destination[])**

| **Reason** | **source** | **return** | **destination** |
| --- | --- | --- | --- |
| Already clean | "by the way" | 10 | "by the way" |
| Mixed case | "By tHe waY" | 10 | "by the way" |
| Leading/trailing garbage chars | " /42 By tHe waY 0] " | 10 | "by the way" |
| Garbage chars around words | " /2By ;tHe-- [waY] " | 10 | "by the way" |
| Garbage chars within words | "By\n7\t 7tHe~@]~waY]" | 10 | "by the way" |
| Alpha chars only | "ByThEwAY" | 8 | "bytheway" |
| Garbage chars only | " \n 4-`" | 0 | "" |
| Empty string | "" | 0 | "" |

**Function boundaryTest(const char line[], int lineLength, int start, int subLength)**

| **Reason** | **line** | **lineLength** | **start** | **subLength** | **return** |
| --- | --- | --- | --- | --- | --- |
| True case in the middle of line | "aft fat taf" | 11 | 4 | 3 | true |
| True case at the beginning of line | "aft fat taf" | 11 | 0 | 7 | true |
| True case at the end of line | "aft fat taf" | 11 | 8 | 3 | true |
| False case in the middle of line | "aft fat taf" | 11 | 3 | 4 | false |
| False case at the beginning of line | "aft fat taf" | 11 | 0 | 2 | false |
| False case at the end of line | "aft fat taf" | 11 | 2 | 9 | false |
| sub\_len = line\_len | "aft fat taf" | 11 | 0 | 11 | true |
| sub\_len = 0 | "aft fat taf" | 11 | 7 | 0 | unhandled |
| line\_len = 0 | "aft fat taf" | 0 | 0 | 0 | unhandled |
| sub\_len < 0 | "aft fat taf" | 11 | 4 | -1 | unhandled |
| line\_len < 0 | "aft fat taf" | -5 | 4 | 3 | unhandled |
| start out of bounds | "aft fat taf" | 11 | -5 | 8 | unhandled |

**Function map(const char \*sub, const char \*crib, int len, char map1[], char map2[])**

| **Reason** | **sub** | **crib** | **len** | **return** | **map1** | **map2** |
| --- | --- | --- | --- | --- | --- | --- |
| Valid mapping exists | undefined | "hello world" | 11 | true | "...ef..i...m..p..s....x..." | "....de..h...l..o..r....w.." |
| Exact match | "hello world" | "hello world" | 11 | true | "...de..h...l..o..r....w..." | "...de..h...l..o..r....w..." |
| Duplicate mapping in sub | "ifmmp xasme" | "hello world" | 11 | false | undefined | undefined |
| Duplicate mapping in crib | "ifmmp xpsme" | "hello world" | 11 | false | undefined | undefined |
| Word boundary mismatch | "ifmm pxpsme" | "hello world" | 11 | false | undefined | undefined |

**Function decrypt(const char ciphertext[], const char crib[])**

| **Reason** | **ciphertext** | **crib** | **return** | **Output** |
| --- | --- | --- | --- | --- |
| Clean, decipherable | "xibu epft\n\nuif gpy tbz" | "the fox" | true | "xHbT eOEt\n\nTHE FOX tbz" |
| Unclean, decipherable | "|Xibu \*ePft\n\nuiF99GpY TBZ? " | "9-thE)fOx0" | true | "|xHbT \*eOEt\n\nTHE99FOX tbz? " |
| Empty crib | "|Xibu \*ePft\n\nuiF99GpY TBZ? " | "" | false | none |
| Nonexistent crib | "|Xibu \*ePft\n\nuiF99GpY TBZ? " | "the foe" | false | none |
| Empty ciphertext | "" | "the fox" | false | none |