```
//
//  main.cpp
//  Final
//
//  Created by Chris West on 10/4/16.
//  Copyright © 2016 Chris West. All rights reserved.
//
#include <algorithm>
#include <ctype.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <numeric>
#include <pthread.h>
#include <sstream>
#include <stdio.h>
#include <stdlib.h>
#include <string>
    //#include <uninstd.h> gives errors on mac
#include <vector>
#include <map>
using namespace std;
void* decoder(void* message); // divides the message into 3 parts
void* sifter(void*); // takes user input; message must be valid
void* substitute (void* input); // part 1
void* hill (void* input); // part 2
void* pinnacol (void* input); // part 3
map<char,int> alphabet = {{'A',0},{'B',1},{'C',2},{'D',3},{'E',4},{'F',5},{'G',6}
    ,{'H',7},{'I',8},{'J',9},{'K',10},{'L',11},
    {'M',12},{'N',13},{'O',14},{'P',15},{'Q',16},{'R',17},{'S',18},{'T',19},{'U',
        20},{'V',21},{'W',22},{'X',23},{'Y',24},{'Z',25}};
map<int,char> numAlpha = {{0,'A'},{1,'B'},{2,'C'},{3,'D'},{4,'E'},{5,'F'},{6,'G'}
    ,{7,'H'},{8,'I'},{9,'J'},{10,'K'},{11,'L'},{12,'M'},
    {13,'N'},{14,'O'},{15,'P'},{16,'Q'},{17,'R'},{18,'S'},{19,'T'},{20,'U'},{21,
        'V'},{22,'W'},{23,'X'},{24,'Y'},{25,'Z'}};
int main (){
    pthread_t sifters;
    string message;
    int result = pthread_create(&sifters, NULL, sifter, NULL);

    if (result){
        printf("Error - pthread_create() return code: %d\n", result);
        exit(EXIT_FAILURE);
    }
    pthread_join(sifters, NULL);
    exit(EXIT_SUCCESS);
}
void* sifter(void*){
    pthread_t decoders;

    int fails = 0, star = 0, starONE = 0, starTWO = 0, starTHREE = 0; // starONE,
        etc is total # of stars;
    bool inputFAIL = false, inputPASS = false, starMORE = false, inputKEYWORD =
        false;
    string input;
    while((fails < 3) && (inputKEYWORD != true)){
```

```
cout << "\nEnter a character string(You got " << 3 - fails << " tries
    left): ";
getline(cin,input);
inputFAIL = false;
if(input == "DAM"){
    inputKEYWORD = true;
    cout<< "\nKeyword entered. Now Exiting program!!" << endl;
    break;
}
else if(input.empty()){ // checks to see if string is empty
    inputFAIL = true;
    fails++;
}

while (inputFAIL != true && inputPASS != true && inputKEYWORD != true)
    { // checks string: approves or fails

    for (int x = 0; x < input.size(); x++){
        star = 0;

        while (input[x] == '*'){ // Find stars
            star++;
            x++;
        }

        bool notNULL = true, breakOUT = false;
        if (star > 0){

            while ((x < input.size()) && breakOUT != true){
                char value = input[x];

                if (isalnum(value) && (x >! input.size() || x != input.
                    size())){
                    breakOUT = true;

                }else if(value == '*'){
                    breakOUT = true;
                    notNULL = false;

                }else if(x >= input.size()){
                    breakOUT = true;
                    notNULL = false;
                }
                x++;
            }

            if (notNULL != false){

                if(star == 1){
                    starONE++;
                }
                else if (star == 2){
                    starTWO++;
                }
                else if (star == 3){
                    starTHREE++;
```

```
                        }
                        else if (star > 3){
                            starMORE = true;
                        }
                    }

                }


            }// exit for loop

            if((starONE != 1 && 1 != starTWO && 1 != starTHREE)){

                if (input == "DAM"){
                    inputKEYWORD = true;
                }else{
                    fails++;
                    inputFAIL = true;
                    input.clear();
                }

            }
            else if ((starONE == 1) && (1 == starTWO) && (1 == starTHREE)){
                inputPASS = true;
                if(inputPASS){

                    int result = pthread_create(&decoders, NULL, decoder,(void*)
                        &input);
                    if(result){
                        printf("Error - pthread creation failed, return code is:
                            %d\n", result);
                        exit(EXIT_FAILURE);
                    }

                    pthread_join(decoders, NULL);
                }
                starONE = starTWO = starTHREE = 0;
                inputPASS = false;
                inputFAIL = true;
                input.clear();
            }

        } // exit for loop

        inputFAIL = true; // make exit true to leave loop

    } // leave while loops

    return 0;

}
void *decoder(void *messages){
    pthread_t substitute1T, hill1T, pinnacol1T;
    string *(mess) = static_cast<string*>(messages);

    int count = 1, stars = 0, start = 0;
```

```
char asterisk = 42;
string message = *mess;
string messageSub, messageHil, messagePin;
for (int x = 0; x < message.size(); x++){
    bool charFOUND = false;
    stars = 0, count =0;
    while (message[x] == '*'){
        stars++;
        x++;
    }
    if (stars > 0 && stars < 4){
        while (message[x] != asterisk && (x < message.size())){ // Future
            reminder Thread 1: Exc_bad_access remember to make x < size
            if (isspace(message[x]) && charFOUND != true){
                x++;
            }else if (charFOUND != true){
                start = x;
                charFOUND = true;
                count++;
                x++;

            }else {
                count++;
                x++;
            }
        }
        if (stars == 1){
            messageSub = message.substr(start,count);

        }else if (stars == 2){
            messageHil = message.substr(start,count);
        }else{
            messagePin = message.substr(start,count);
        }
        --x;
    }
}
int result = pthread_create(&substitute1T, NULL, &substitute, &messageSub);
if (result){
    printf("Error - pthread_create() return code: %d\n", result);
    exit(EXIT_FAILURE);
}
pthread_join(substitute1T, NULL);
int result2 = pthread_create(&hill1T, NULL, &hill, &messageHil);
if (result2){
    printf("Error - pthread_create() return code: %d\n", result2);
    exit(EXIT_FAILURE);
}
pthread_join(hill1T, NULL);

int result3 = pthread_create(&pinnacol1T, NULL, &pinnacol, &messagePin);

if (result3){
    printf("Error - pthread_create() return code: %d\n", result3);
    exit(EXIT_FAILURE);
}
```

```
    pthread_join(pinnacol1T, NULL);
    pthread_exit(NULL);
}
void* substitute (void* input){ // part1
    string *inputOLD = static_cast<string*>(input);
    vector <char> message;
    string source = *inputOLD;
    int key = 0, placeOfKey = 0;
    bool keyFound = false;
    for(int h = 0; h < source.size(); h++){

        if (isalpha(source.at(h)) && keyFound != true){
            source[h] = toupper(source.at(h));
            key = alphabet.at(source.at(h));
            placeOfKey = h+1;
            keyFound = true;
                }

            }

    for(int i = placeOfKey; i < source.size(); i++){
        char l = source.at(i);
        message.push_back(l);
    }
    bool failed_part1 = false;
    for(int k=0; k < message.size(); k++)
    {
        char alpha = message.at(k);
        if ((isspace(alpha) || isalpha(alpha)) && failed_part1 != true){
            failed_part1 = false;
        }
        else{
            failed_part1 = true;
        }
    }

    if (failed_part1 == false){

        key = (key % 10 + 2);
        int letterNum = 0;

        for(int k=0; k < message.size(); k++)
        {
            char alpha = message.at(k);

            if (!isspace(alpha)){

                letterNum = toupper(alpha) - key;
                if( letterNum < 'A' ){
                    letterNum = letterNum + 26;
                }

                alpha = letterNum;
                message.at(k) = alpha;
            }
        }
```

```
        cout << "\nSubstitute: ";
        for (int m = 0; m < message.size(); m ++){
            char kim = message.at(m);
            cout << kim;
        }
    }
    else{
        cout << "\nSubstitute Fails: Contains non-alphabet letters";
    }

    return (0);
}

void* hill (void* input){ // part2
    int count2, start2 = 0, countSECT1 = 0, begin = 0, end = 0;
    int m = 0,n = 0,o = 0,p = 0;
    string *inputOLD = static_cast<string*>(input);

    string message = *inputOLD, sectionONE, sectionTWO, values;
    vector<string> numbers;
    vector<string> newMessage;
    bool sectionFOUND = false, section2FOUND = false, sectONEFAIL = false,
        sectTWOFAIL = false;
    string temp = "";
    for(int x = 0; x < message.size(); x++){ // break sections
        if((isdigit(message[x]) ||  message[x] == '-') && sectionFOUND != true)
            { // section 1 is finished
            sectionFOUND = true;
            if (section2FOUND != true){
                section2FOUND = true;
                count2++;
                start2 = x;
            }
        } else if (sectionFOUND != true){
            if (isspace(message[x])){ // section 1 start

            }else{ // section 1 end
                temp += message[x];
            }

        } else {
            if((isdigit(message[x]) ||  message[x] == '-') && section2FOUND !=
                true){

            }else{
                count2++;
            }
        }
    }

    sectionONE = temp;

    sectionTWO = message.substr(start2, count2);
    for (int x = 0; x < sectionONE.size(); x++){
        if (isalpha(sectionONE[x])){
            countSECT1++;
```

```
        }else{
            sectONEFAIL = true;
        }
    }
    if (countSECT1 % 2 != 0){ // one is odd and zero is even
        sectONEFAIL = true;
    }
    for (int y = 0; y < sectionTWO.size(); y++){
        if (isdigit(sectionTWO[y]) || sectionTWO[y] == '-' || isspace(sectionTWO
            [y])){

        }else{
            sectONEFAIL = true;
        }
    }
    if(!sectONEFAIL && !sectTWOFAIL){
        for (int z = 0; z < sectionTWO.size(); z++){
            if(isdigit(sectionTWO[z]) || sectionTWO[z] == '-'){

                if (end == 0){
                    begin = z;
                    end++;
                }else{
                    end++;
                }
            }else{
                if (end > 0){
                    values = sectionTWO.substr(begin,end);
                    numbers.push_back(values);
                    end = 0;
                }
            }

        }
        if (end > 0){
            values = sectionTWO.substr(begin,end);
            numbers.push_back(values);
            end = 0;
        }

    }

    if (numbers.size() != 4){
        sectTWOFAIL = true;
    }
    if(!sectONEFAIL && !sectTWOFAIL){
        m = stoi(numbers.at(0));
        n = stoi(numbers.at(1));
        o = stoi(numbers.at(2));
        p = stoi(numbers.at(3));
        for (int t = 0; t < sectionONE.size(); t++){
            int temp = t;

            sectionONE[t] = toupper(sectionONE.at(t));
            sectionONE[t+1] = toupper(sectionONE.at(t+1));
```

```
            char a = sectionONE.at(t), b = sectionONE.at(temp+1);

            int aValue = alphabet.at(a), bValue = alphabet.at(b);
            int Avalue = (m * aValue) + (n * bValue), Bvalue = (o * aValue) + (p
                * bValue);
            Avalue = Avalue % 26;
            Bvalue = Bvalue % 26;

            while (Avalue < 0 || Bvalue < 0){
                if (Avalue < 0)
                    Avalue += 26;
                else if (Bvalue < 0)
                    Bvalue += 26;
            }

            a = numAlpha.at(Avalue);
            b = numAlpha.at(Bvalue);
            if (t == 0){
                cout << "\nHill: " << a << b;
            }else{
                cout << a << b;
            }

            t+= 1;

        }

    }
    else if(sectONEFAIL){
        cout << "\nHill Fails: Section 1 includes characters other than alphabet
            characters or number of characters is not even.";
    }else if (sectTWOFAIL){
        cout << "\nHill Fails: Either the number of tokens in section 2 is not 4
            or they are not made up of only digits.";
    }

    return 0;
}
void* pinnacol (void* input){ // part3
    int count2, start2 = 0, countSECT1 = 0, begin = 0, end = 0;
    int m = 0,n = 0,o = 0,p = 0, q = 0, r = 0, s = 0, t = 0, u = 0;
    string *inputOLD = static_cast<string*>(input);
    string message = *inputOLD, sectionONE, sectionTWO, values;
    vector<string> numbers;
    vector<string> newMessage;
    bool sectionFOUND = false, section2FOUND = false, sectONEFAIL = false,
        sectTWOFAIL = false;
    string temp = "";
    for(int x = 0; x < message.size(); x++){ // break sections
        if((isdigit(message[x]) ||  message[x] == '-') && sectionFOUND != true)
            { // section 1 is finished
            sectionFOUND = true;
            if (section2FOUND != true){
                section2FOUND = true;
                count2++;
                start2 = x;
```

```
            }
        } else if (sectionFOUND != true){
            if (isspace(message[x])){ // section 1 start

            }else{ // section 1 end
                temp += message[x];
            }

        } else {
            if((isdigit(message[x]) ||  message[x] == '-') && section2FOUND !=
                true){

            }else{
                count2++;
            }
        }
    }

    sectionONE = temp;
    sectionTWO = message.substr(start2, count2);
    for (int x = 0; x < sectionONE.size(); x++){
        if (isalpha(sectionONE[x])){
            countSECT1++;
        }else{
            sectONEFAIL = true;
        }
    }
    if (countSECT1 % 3 != 0){ // one is odd and zero is even
        sectONEFAIL = true;
    }
    for (int y = 0; y < sectionTWO.size(); y++){
        if (isdigit(sectionTWO[y]) || sectionTWO[y] == '-' || isspace(sectionTWO
            [y])){

        }else{
            sectONEFAIL = true;
        }
    }
    if(!sectONEFAIL && !sectTWOFAIL){
        for (int z = 0; z < sectionTWO.size(); z++){
            if(isdigit(sectionTWO[z]) || sectionTWO[z] == '-'){

                if (end == 0){
                    begin = z;
                    end++;
                }else{
                    end++;
                }
            }else{
                if (end > 0){
                    values = sectionTWO.substr(begin,end);
                    numbers.push_back(values);
                    end = 0;
                }
            }
        }
```

```
        if (end > 0){
            values = sectionTWO.substr(begin,end);
            numbers.push_back(values);
            end = 0;
        }
    }

    if (numbers.size() != 9){
        sectTWOFAIL = true;
    }
    if(!sectONEFAIL && !sectTWOFAIL){
        m = stoi(numbers.at(0));
        n = stoi(numbers.at(1));
        o = stoi(numbers.at(2));
        p = stoi(numbers.at(3));
        q = stoi(numbers.at(4));
        r = stoi(numbers.at(5));
        s = stoi(numbers.at(6));
        t = stoi(numbers.at(7));
        u = stoi(numbers.at(8));
        for (int t = 0; t < sectionONE.size(); t++){
            int temp = t;
            sectionONE[t] = toupper(sectionONE.at(t));
            sectionONE[t+1] = toupper(sectionONE.at(t+1));
            sectionONE[t+2] = toupper(sectionONE.at(t+2));
            char a = sectionONE.at(t), b = sectionONE.at(temp+1), c = sectionONE.
                at(temp+2);
            int aValue = alphabet.at(a), bValue = alphabet.at(b), cValue =
                alphabet.at(c);
            int Avalue = (m * aValue) + (n * bValue) + (o * cValue), Bvalue = (p
                * aValue) + (q * bValue) + (r * cValue),
            Cvalue = (s * aValue) + (t * bValue) + (u * cValue);
            Avalue = Avalue % 26;
            Bvalue = Bvalue % 26;
            Cvalue = Cvalue % 26;
            while (Avalue < 0 || Bvalue < 0 || Cvalue < 0){
                if (Avalue < 0)
                    Avalue += 26;
                else if (Bvalue < 0)
                    Bvalue += 26;
                else if (Cvalue < 0)
                    Cvalue += 26;
            }
            a = numAlpha.at(Avalue);
            b = numAlpha.at(Bvalue);
            if (t == 0){
                cout << "\nPinnacol: " << a << b << c;
            }else{
                cout << a << b << c;
            }

            t = temp+2;
        }
    }
    else if (sectONEFAIL){
        cout << "\nPinnacol Fails: " << "Section 1 includes characters other than
```

```
            alphabet characters or the number of characters in the section 1 is
            not even."<< endl;
        }else if(sectTWOFAIL){
            cout << "\nPinnacol Fails: Either the number of tokens in section 2 is
            not 9 or they are not made up of only digits." << endl;
        }
        return 0;
    }
```