INSTITUTO POLITÉCNICO NACIONAL

"ESCOM" (ESCUELA SUPERIOR DE CÓMPUTO)

DOCENTE: CECILIA ALBORTANTE MORATO

INTEGRANTES:

MONTEALEGRE ROSALES DAVID URIEL

VÁZQUEZ BLANCAS CÉSAR SAID

U.A.: ANÁLISIS Y DISEÑO DE ALGORITMOS

GRUPO: 3CM6

PRÁCTICA 1: RESOLUCIÓN DE PROBLEMAS

- Elegir 3 de los siguientes 6 problemas y resolverlos en lenguaje C.

1. Word
2. Angry Students
3. What´s Next
4. Sticks
5. Different Digits
6. Cryptograpy

Práctica 1
Resolución de problemas

Problemas por resolver

A través de los seis problemas propuestos en clase para ser resueltos, decidimos que lo más conveniente sería resolver los problemas:

- Word
- Sticks
- Cryptograpy

Esto debido a que nos parecieron los problemas más factibles para resolver, tanto en grado de complejidad y en grado de enseñanza. A continuación, explicaremos cómo fue resuelto cada uno de estos problemas gracias a la programación.

Desarrollo de la práctica

Problema N°1: Word. El problema nos solicita implementar un algoritmo el cuál pueda ser capaz de cambiarnos una oración escrita con mayúsculas y minúsculas a una oración escrita únicamente con mayúsculas o minúsculas, esto dependiendo si hay mayor cantidad de mayúsculas o minúsculas en la palabra, en dado caso que existiera la misma cantidad de ambas, el programa lo detectará y nos imprimirá la palabra escrita únicamente con minúsculas.

El código utilizado para resolver el problema es el siguiente:

```c
//Montealegre Rosales David Uriel
//Vázquez Blancas César Said
#include <stdio.h>
int mayuscula(char c[]){//funcion para contar mayusculas
    int j=0;
    for(int i=0;c[i]!='\0';i++){ //iteramos hasta que se acabe la cadena
        if(c[i]>=65 && c[i]<=90){//comparamos si la letra esta en el rango de las mayusculas
            j++;//si esta en el rango aumentamos j
        }
    }
    return j;//cuando acabe de comparar la cadena se retorna el valor de las mayusculas
}
int minuscula(char c[]){ //funcion para contar minusculas
    int j=0;
    for(int i=0;c[i]!='\0';i++){//iteramos hasta el final de la cadena
        if(c[i]>=97 && c[i]<=122){//si esta en el rango de minusculas
            j++;//aumentamos j
        }
    }
    return j;//retornamos el valor de las minusculas
}
void convmayus(char c[]){//funcion para convertir en mayusculas
    char*p;//declaramos un apuntador
    p=&c[0];//lo inicializamos al inicio de la cadena
    while(*p!='\0'){//mientras el apuntador sea diferente del fin de cadena se itera
        if(*p>=97 && *p<=122){//si la letra esta en el rango de las minusculas entra en el if
            *p-=32;//si es minuscula reduce en 32 por lo que ahora sera la misma letra pero en mayuscula
        }
        p++;//aumenta p
    }
}
```

```c
void convminus(char c[]){//funcion para convertir en minusculas
    char*p;//declaramos un apuntador
    p=&c[0];//el apuntador se inicializa en la primer letra
    while(*p!='\0'){//mientras p sea difernete al fin de cadena se va a iterar
        if(*p>=65 && *p<=90){//si la letra a la que apunta p esta en el rango de las mayusculas
            *p+=32;//si entra en el if aumenta en 32 la letra, osea va a aumentar a una minuscula
        }
        p++;//aumenta la posicion
    }
}
int main()
{
    char c[100];//declaramos la cadena
    printf("Escribe la cadena con mayusculas y minusculas: \n");//pedimos una palabra con mayusculas y minusculas
    scanf("%s",c);//leemos la cadena
    if((minuscula(c))<(mayuscula(c))){//si hay mas myusculas en la cadena
        convmayus(c);//convertimos toda la cadena a mayusculas
    }
    else{// si hay mas minusculas
        convminus(c);//convertimos la cadena a minusculas
    }
    printf("%s",c);//imprimimos la cadena modificada
    return 0;
}
```

En el código se puede observar que se utilizan cuatro funciones:

- ✓ Las funciones "mayuscula" y "minuscula" son utilizadas para contar cuantas mayúsculas y cuantas minúsculas existen en la palabra escrita por el usuario, ambas ocupan un ciclo for para que se revise cada caracter de la cadena, también contienen una condicional la cual nos ayudará a detectar si se trata de una mayúscula o una minúscula (esto gracias al uso del código ASCII), ambas tienen un contador para que puedan ser contadas las mayúsculas y minúsculas que contiene la palabra.
- ✓ Las funciones "covmayus" y "conminus" serán utilizadas para convertir toda la palabra en letras mayúsculas o en letras minúsculas (esto dependerá de cuál de las dos sea llamada por el programa inicial), esto será posible de nueva cuenta a una condicional que utiliza al código ASCII para convertir las letras en mayúsculas o minúsculas según sea el caso.

El programa inicial nos solicitará la palabra a evaluar, posterior a esto nos encontramos con una condicional, la cuál será la que nos indique si existen más letras mayúsculas o minúsculas en la palabra (esto llamando a las funciones mayuscula y minuscula), dependiendo de cuál tenga mayor cantidad de letras, el programa llamará a las funciones "convmayus" o "convminus" para que las letras sean escritas de una sola manera, finalmente el programa nos imprimirá la cadena corregida de la forma que nos indica el problema.

Problema N°2 Sticks. Este problema nos plantea que se tiene una n cantidad de palos a utilizar, a partir de que el usuario nos indique cuántos palos tiene y la longitud de cada uno de ellos, se deberá de crear un algoritmo capaz de detectar si es posible formar un rectángulo con los palos que contiene el usuario, de ser así, también deberá indicar el área máxima que se puede obtener con cuatro palos de valores previamente dados por el usuario, todo esto sin alterar la longitud de estos.

El código utilizado para resolver el problema es el siguiente:

```c
//Montealegre Rosales David Uriel
//Vázquez Blancas César Said
#include <stdio.h>
#include <stdlib.h>

void rapido(int arr[], int inicio, int fin) { //metodo de ordenamiento para los palos dados por el usuario
    int izquierda, derecha, pivote, auxiliar;

    izquierda = inicio;
    derecha = fin;
    pivote = arr[(inicio + fin) / 2];

    do {
        while (arr[izquierda] < pivote) {
            izquierda++;
        }
        while (arr[derecha] > pivote) {
            derecha--;
        }
        if (izquierda <= derecha) {
            auxiliar = arr[izquierda];
            arr[izquierda] = arr[derecha];
            arr[derecha] = auxiliar;
            izquierda++;
            derecha--;
        }
    } while (izquierda <= derecha);

    if (inicio < derecha) {
        rapido(arr, inicio, derecha);
    }
    if (izquierda < fin) {
        rapido(arr, izquierda, fin);
    }
}

void invertir(int arreglo[], int longitud){
    int inicio=0, fin=longitud-1;
    while(inicio < fin){
        int temp = arreglo[inicio];
        arreglo[inicio] = arreglo[fin];
        arreglo[fin] = temp;
        inicio++;
        fin--;
    }
}

int main()
{
    int longitud;
    int *palos;
    int max_area = 0;
    int base = 0;
    int altura = 0;
    int area_actual = 0; //variables a utilizar para la resolución del problema

    printf("Indique el numero de palos que tiene a su disposicion: ");
    scanf("%d", &longitud); //lectura del numero de palos que tiene el usuario

    if (longitud < 4)
    {
        printf("No es posible formar un rectangulo con menos de 4 palos");
        return 1; //restriccion del programa, si no se tienen minimo cuatro palos es imposible formar un rectangulo
    }

    palos = (int *)malloc(longitud * sizeof(int)); //uso de memoria dinamica

    for (int n = 0; n < longitud; n++)
```

```
    for (int n = 0; n < longitud; n++)
    {
        printf("\nIndique la longitud del palo %d: ", n + 1);
        scanf("%d", &palos[n]); //lectura de la longitud de cada palo guardadoz en un arreglo
    }

    rapido(palos, 0, longitud - 1); //llamada a funcion de ordenamiento rapido
    invertir(palos, longitud);

    for (int i = 0; i < longitud - 1; i++)
    {
        if (palos[i] == palos[i + 1]) //comparacion entre si de los palos, esto para ir formando parejas, ya que un rectangulo siempre t
        {
            if (base == 0)
            {
                base = palos[i]; //si se encuentra una pareja de palos con longitud igual se guarda en la variable base
            }
            else if (altura == 0)
            {
                altura = palos[i]; //si la variable base ya se habia utilizado pero si se vuelve a encontrar una pareja de palos con igu
            }

            if (base > 0 && altura > 0) //si fueron encontradas dos parejas de palos se entra a la condicion if
            {
                area_actual = base * altura; //se obtiene el area que se puede formar con los cuatro palos que se tiene
                if (area_actual > max_area) //comparacion para saber si se esta encontrando el area maxima
                {
                    max_area = area_actual; //si fue encontrada, se guardara el area maxima en la variable max_area
                }
                //base = 0;
                //altura = 0; //reset de variables base y altura para ser utilizadas nuevamente
            }

            i++;
```

```
        }
    }

    if (max_area > 0)
    {
        printf("\nEl area maxima posible del rectangulo es: %d\n", max_area); //esto pasa si fue encontrada la posibilidad de formar un re
    }
    else
    {
        printf("\nNo es posible formar un rectangulo con los palos dados.\n"); //si no existe la psoibilidad de formar el rectangulo
    }

    free(palos); //libera memoria dinamica
    return 0; //fin del programa
}
```

En la primera parte del algoritmo podemos observar la función "rapido", esta función se trata de un tipo de ordenamiento el cuál nos ayudará a tener los números ordenados y que la solución del problema se vuelva más sencilla.

También podemos observar que tenemos una función la cuál nos ayudará a invertir el arreglo de la longitud de los palos con el fin de que resulte más sencillo encontrar el área máxima posible a formar.

En el código inicial podemos observar que le solicitará al usuario que indique cuántos palos posee y cuál es la longitud de cada uno de ellos. Si el usuario indica que tiene menos de cuatro palos, el programa arrojará un mensaje diciendo que no es posible formar un rectángulo con menos de cuatro palos. El programa mandará a llamar al método de ordenamiento "rapido", después de ello a la función "invertir" para que posterior a esto entre en un ciclo for, básicamente este ciclo for será el encargado de detectar si existen dos parejas de dos palos de igual medida (ya que un rectángulo tiene dos parejas de dos longitudes distintas), además de esto, este ciclo será el encargado de detectar cuál es la combinación de parejas que nos dará la mayor área posible que se puede formar al construir el rectángulo. Finalmente, el programa nos indicará si es posible o no formar el rectángulo, de ser así, nos indicará el área máxima que se puede formar.

Problema N°3 Cryptograpy. El problema nos plantea diseñar un algoritmo el cual solicite una contraseña de números al usuario, al ingresar la contraseña el algoritmo se encargara de encriptar la contraseña de tal forma que cada número ingresado será equivalente al número primo que le corresponda ese número, es decir, si ingresamos el número "2" entonces el programa nos mandará como impresión el número "3", ya que es el segundo número primo que existe.

El código utilizado para resolver el problema planteado es el siguiente:

```c
//Montealegre Rosales David Uriel
//Vázquez Blancas César Said
#include<stdio.h>
#include<stdlib.h>
void LlenaArreglo(unsigned long long int *a, int n){//funcion para llenar el arreglo
   int i;
   for(i=0;i<n;i++){//iteramos la posicion desde 0 hasta la que nos dio el usuario
      scanf("%llu",&a[i]);//pedimos los valores, como pueden ser grandes se ocupa un long long int
   }
}
void ImpArreglo(unsigned long long int *a, int n){//funcion para imprimir el arreglo
    int i;
    for(i=0;i<n;i++)//iteramos la posicion de 0 hasta la que nos dio el usuario
       printf("%llu\n",a[i]);//imprimimos los valores
}
int esPrimo(int num) {//funcion para verificar si es primo un numero
    if (num == 2) {//si el numero es 2
        return 1;//retornamos el 1, que es el verdadero o es decir, que el numero es primo
    } else {//si no es 2
        for (int i = 2; i*i <= num; i++) {//iteramos la posicion de 2 hasta i*i sea menor o igual al numero esto porque i*i se utiliza co
            if (num % i == 0) {//si en la iteracion el modulo de num e i es 0, entonces no es primo porque encontro un numero divisible d
                return 0;//retorna un valor de falso
            }
        }
        return 1;//si no encuentra nada entonces quiere decir que es primo y por lo tanto retorna un verdadero
    }
}
void primo(unsigned long long int *a,unsigned long long int *b,int n){//b esta funcion convierte el numero del arreglo en la poscion del
    int j=0;
    for(int i=0;i<n;i++){//iteramos desde la posicion uno del arreglo hasta la final
    unsigned long long int cont = 0;//este contador contara el numero de primos que llevamos
    int num = 2;//iniciamos un num en 2 que sera el primer primo en ser considerado
        while(1){//bucle que se iterara hasta que haya un break, ya que no sabemos cuando acabaremos
        if (esPrimo(num)) {//si el num es un primo entra en el if
            cont++;//cuando entra el contador se aumenta indicando que es el n primo en ser considerado primo
            if(cont==a[i]){//si el contador es igual al numero del arreglo, es decir si los numero primos que llevamos es igual al nu
            b[j]=num;//el numero primo que fue el ultimo se guaradara en la posicion j del arreglo
            j++;//aumentamos j por lo que se aumenta la posicion
            break;//rompemos el ciclo pues se encontro el numero primo en la posicion requerida
            }
        }
        num++;//en caso de que no sea primo se aumenta al siguiente numero natural y se comprueba constantemente por el while
}
}
}
int main(){
int n;
printf("Escribe el tamaño de tu contraseña en numeros enteros: \t");
scanf("%d",&n);//leemos el tamaño del arreglo
unsigned long long int *a=(unsigned long long int *)malloc(n*sizeof(unsigned long long int)) ;//guardamos las posiciones en un arregl
unsigned long long int *b=(unsigned long long int *)malloc((n)*sizeof(unsigned long long int));//guardamos las posiciones en otro arr
printf("Escribe los numeros de tu contraseña uno por uno: \n");
LlenaArreglo(a,n);//pedimos los valores al usuario
primo(a,b,n);//llamamos primo para convertir a primo las posiciones del arreglo original
printf("\n");//salto de linea para visualizar el arreglo original del encriptado
ImpArreglo(b,n);//imprimimos arreglo de primos
free(a);//liberamos el arreglo dinamico a
free(b);//liberamos el arreglo dinamico b
return 0;
}
```

En la primera parte del código hay funciones para recorrer el arreglo y escanear los números que quiere que el usuario quiere que se modifique, eso en llena arreglo, en imprime recorre el arreglo e imprime el mismo uno por uno, la primera función que se ve es "EsPrimo", esta función hace que si es primo devuelva un 1, empezando por un 2, si es 2, devuelve un 1, si no, entra en el for donde inicia en 2, hasta que i*i sea mayor o igual al número, esto hace que tome la raíz del número, pues al multiplicar el i*i si es mayor asegura que esa es la raíz más cercana al número, esto asegura que no recorra todos los números, si no hasta la raíz aproximada de este, ya que esta es una regla matemática que dice que un primo se puede probar hasta la raíz entera más próxima a este, después dice que si el módulo de num e i es 0, entonces no es primo, y sale del for retornando un 0, en cambio sí es primo solo se ejecutara sin entrar nunca en este if, la función "Primo" ya convierte el entero en la posición del primo de este entero, esto quiere decir que si pones un 5, lo convertirá a la quinta posición de este mismo, esta inicia con un ciclo for, que este recorrerá el arreglo iniciando en el 0 y terminando en el n-1, se le asignara un contador que inicie en 0 y un entero que será el que vaya aumentando de uno en uno hasta la raíz del número y que será el que lleve el control para saber que numero es primo y cual no, ahora un ciclo while que es verdadero e iterara hasta que se tope con un break, aquí verificara el número y si es primo entrara en el if, en el if el contador se aumenta y entra en otro if, este será el que compare el número de primos con la posición del arreglo, esto hará ver si ya se llegó al número de primos que requería la entrada inicial, cuando entra en el if guardara el num que está en el arreglo b, y aumentara la posición del arreglo de b y rompe el while, si no entra en el primer if, solo aumentara el num y comparara otra vez. Al final se imprime el arreglo b y se libera espacio de la memoria requerida

Resultados.

Aquí se observará que cada algoritmo realizado puede resolver el problema planteado para el que fue diseñado:

N°1 - Word.

```
C:\Users\alexl\Downloads>gcc Word.c -o word

C:\Users\alexl\Downloads>word
Escribe la cadena con mayusculas y minusculas:
PALAbra
PALABRA
C:\Users\alexl\Downloads>
```

```
C:\Users\alexl\Downloads>gcc Word.c -o word

C:\Users\alexl\Downloads>word
Escribe la cadena con mayusculas y minusculas:
PALAbra
PALABRA
C:\Users\alexl\Downloads>word
Escribe la cadena con mayusculas y minusculas:
camiNaTa
caminata
C:\Users\alexl\Downloads>
```

```
C:\Users\alexl\Downloads>gcc Word.c -o word

C:\Users\alexl\Downloads>word
Escribe la cadena con mayusculas y minusculas:
PALAbra
PALABRA
C:\Users\alexl\Downloads>word
Escribe la cadena con mayusculas y minusculas:
camiNaTa
caminata
C:\Users\alexl\Downloads>word
Escribe la cadena con mayusculas y minusculas:
raYO
rayo
C:\Users\alexl\Downloads>
```

En las capturas de pantalla se puede observar que el programa cumple con las condiciones que tiene el problema:

- ✓ Cuando contiene más mayúsculas que minúsculas entonces la palabra se escribe en mayúsculas completamente.
- ✓ Cuando contiene menos mayúsculas que minúsculas entonces la palabra se escribe en minúsculas completamente.
- ✓ Si contiene la misma cantidad de mayúsculas y minúsculas, la palabra se escribe en minúsculas completamente.

N°2 – Sticks.

```
C:\Users\alexl\Downloads>Rectangulo.c -o sticks

C:\Users\alexl\Downloads>gcc Rectangulo.c -o sticks

C:\Users\alexl\Downloads>sticks
Indique el numero de palos que tiene a su disposicion: 5

Indique la longitud del palo 1: 3

Indique la longitud del palo 2: 3

Indique la longitud del palo 3: 4

Indique la longitud del palo 4: 5

Indique la longitud del palo 5: 5

El area maxima posible del rectangulo es: 15
```

```
C:\Users\alexl\Downloads>sticks
Indique el numero de palos que tiene a su disposicion: 3
No es posible formar un rectangulo con menos de 4 palos
C:\Users\alexl\Downloads>sticks
Indique el numero de palos que tiene a su disposicion: 6

Indique la longitud del palo 1: 2

Indique la longitud del palo 2: 3

Indique la longitud del palo 3: 4

Indique la longitud del palo 4: 5

Indique la longitud del palo 5: 6

Indique la longitud del palo 6: 7

No es posible formar un rectangulo con los palos dados.
```

Podemos observar que el programa cumple con su tarea, cuando detecta que son menos de cuatro palos entonces manda un mensaje de que no es posible formar un rectángulo, si detecta que si es posible formar un rectángulo entonces analiza cuál sería el mayor área posible que se desea formar y se lo imprime al usuario, si detecta que no es posible formar un rectángulo con los palos que tiene el usuario debido a que no encuentra parejas de palos con longitudes iguales entonces manda un mensaje de que no fue posible formar un rectángulo.

N°3 – Cryptograpy.

```
C:\Users\alexl\Downloads>gcc Cryptograpy.c -o cryp

C:\Users\alexl\Downloads>cryp
Escribe el tamano de tu contrasena en numeros enteros:  10
Escribe los numeros de tu contrasena uno por uno:
2
3
5
1
71
12300
7
170
18
9

3
5
11
2
353
131701
17
1013
61
23

C:\Users\alexl\Downloads>
```

```
C:\Users\alexl\Downloads>gcc Cryptograpy.c -o cryp

C:\Users\alexl\Downloads>cryp
Escribe el tamano de tu contrasena en numeros enteros:  5
Escribe los numeros de tu contrasena uno por uno:
15000
14500
13200
10000
1200

163841
157739
142169
104729
9733

C:\Users\alexl\Downloads>
```

Se puede observar que el algoritmo diseñado funciona de manera correcta, ya que solicita al usuario que ingrese cuántos números tendrá la contraseña a ingresar y a su vez que números son, posterior a esto el programa encripta dicha contraseña y nos imprime el número primo correspondiente a cada número ingresado de la contraseña haciendo que nuestra contraseña se encuentre "segura" de cierta manera.