4

```
//Encuentra el únicoelememto de un arreglo
#include <stdio.h>

int getSingle(int arr[], int n)
{
int ones = 0, twos = 0;

int common_bit_mask;

//Se usará el ejemplo de {3,3,2,3} para explicar el programa.
for (int i = 0; i < n; i++) {
/*La expresión "one & arr[i]" da los bits que se encuentra en ambas variables, es decir, en "ones" y "arr[i]".

Se suman estos bits a "twos" usando el operador OR.

El valor de "twos" será establecido como 00, 11, 11 y 01 en las iteraciones correspondientes.*/

twos = twos | (ones & arr[i]);

/*Se aplica XOR a los nuevos bits con los "ones" anteriores
para obtener los bits que aparecen un número impar de veces.

El valor de "ones" será 11, 00, 10 y 11 después de las iteraciones correspondientes*/
ones = ones ^ arr[i];
printf("Iteracion: %d \n", i);
printf ("ones: %d twos: %d iniciales \n", ones, twos);

```

```
/* Los bits comunes son aquellos que aparecen una tercera vez.
Por lo que estos bits no deberán estar en ambas variables "ones" y "twos".
La variable "common_bit_mask" contiene todos los bits como 0, de modo
que los bits puedan ser removidos de "ones" y "twos".

El valor de "common_bit_mask" será 11, 11, 01 y 10 después de
las iteraciones correspondientes*/

common_bit_mask = ~(ones & twos);
/* Se remueven los bits en común (aquellos bits que aparecen por tercera
vez) en "ones".

El valor de "ones" será 11, 00, 00 y 10 después de las iteraciones
correspondientes. */

ones &= common_bit_mask;
/* Se remueven los bits en común (los bits que aparecen por tercera
vez en "twos".

El valor de "twos" será 00, 11, 01 y 00 después de las iteraciones
correspondientes. */

twos &= common_bit_mask;

printf ("ones: %d twos: %d finales \n", ones, twos);
}

return ones;
}
```

```
int main()
{
int arr[] = { 3, 3, 2, 3};
int n = sizeof(arr) / sizeof(arr[0]);
printf("El elemento con multiplicidad unitaria es: %d ",
getSingle(arr, n));
return 0;
}
```

4.1

```
//Multiplicaciones y divisiones
#include <stdio.h>
#include <stdlib.h>
int main(){
int
a,b,resultado,operacion,temp,
cuenta;

printf("Ingrese el número
entero 'a':\n");
scanf("%d", &a);
printf("Ingrese el número
entero 'b':\n");
scanf("%d", &b);

printf("Ingrese 0 para
multiplicacion a*b o 1 para
division a/b:\n");
scanf("%d", &operacion);
resultado = 0;
```

	`if(operacion==0){`
	`while (b != 0)` `// El ciclo se ejecuta mientras "b" sea diferente de "0"`
	`{`
	`printf("El valor actual de b es: %d \n",b);`
	`if (b & 1)` `// Al aplicar "b & 1" se puede determinar si b es número impar`
	`{`
	`resultado = resultado + a;  // Se añade el valor de "a" al resultado para considerar que "b" es impar`
	`printf("El valor acumulado de resultado es: %d\n", resultado);`
	`}`
	`a <<= 1;` `// Se hace un corrimiento de 1 bit a la izquierda para multiplicar por "2" el valor de "a"`
	`printf("El valor actual de a es: %d\n", a);`
	`b >>= 1;` `// Se hace un corrimiento de 1 bit a la derecha para dividir entre "2" el valor de "b"`
	`printf("El valor actual de b es: %d\n", b);`
	`}`
	`printf("El resultado de la multiplicacion es: %d\n",resultado);`
	`}else{`

	while(a >= b){ //El ciclo se ejecuta mientras el dividendo "a" sea mayor o igual al divisor "b"
	temp = b; // Se inicializa el valor de temp con "b"
	cuenta = 1; //Se inicializa el valor de cuenta con "1"
	while(temp <= a) //Se ejecuta mientras "temp" sea menor o igual a "a"
	{
	temp <<= 1; //En este ciclo se registra cuantas veces cabe el divisor "b" en el dividendo "a" y se registra en cuenta en multiplos de 2 al utilizar el corrimiento de 1 bit a la izquierda
	cuenta <<= 1;
	}
	resultado = resultado + (cuenta >> 1); //Se almacena el último valor válido de cuenta en el resultado, para obtener el último valor valido de cuenta se hace un corrimiento de 1 bit a la derecha
	a = a - (temp >> 1); //Se asigna el valor restante al dividendo "a" al restar el último valor acumulado de temp, para obtener el último valor válido de temp se hace un corrimiento de 1 bit a la derecha
	}

```
                                printf("El resultado de la division es: %d\n",resultado);
                                
                                }
                                }
```

4.2

```
//Calcule el cuadrado de un número sin utilizar ningún operador numérico.
                                //Dado un entero N, calcule el cuadrado de un numero sin utilizar *, / o librerías de potencias.
                                #include <stdio.h>
                                
                                int main() {
                                int temp, num;
                                int cuadrado = 0;
                                int cuenta = 0;
                                printf("Ingresa el numero del que deseas saber su cuadrado: \n");
                                scanf("%d", &num);
                                temp = num;
                                //Se asigna a temp el valor ingresa a num
                                while(temp > 0)
                                //Se ejecuta mientras temp sea mayor a 0
                                {
                                if(temp & 1)
                                //Este ciclo se ejecuta mientras el valor de temp sea diferente de 0
```

```
{
//En la primera iteración la cuenta vale 0 por lo que el corrimiento de "cuenta" bits no modifica el valor de "num" y solo se suma a "cuadrado"
cuadrado += num << cuenta;
printf("El valor actual de cuadrado es: %d \n", cuadrado);
}
temp = temp >> 1;
//Se le aplica un corrimiento de 1 bit a la derecha a temp de modo que se divide entre 2 y se actualiza su valor
printf("El valor actual de temp es: %d \n", temp);
cuenta++;
//El valor de cuenta se incrementa en 1, esto nos servirá en la siguiente iteración para añadir a el doble de veces el valor de "num" a "cuadrado"
printf("El valor actual de cuenta es: %d \n", cuenta);
// y esto se repetirá hasta que temp se vuelva 0
}
// de modo que se habrán sumado "num" veces "num" a "cuadrado" y de esta forma se obtiene el cuadrado del número ingresado
printf("El cuadrado de %d es: %d\n", num, cuadrado );
}
```

4.3

```
//Generador de secuencias de Grey Code
```

```
#include <stdio.h>
int decimalToBinaryNumber(int x, int n);    //Función prototipo
int generateGrayarr(int n); //Función prototipo


int main(){
int n;
printf("Ingrese el numero del que se quiere generar su secuencia en Grey Code: \n");
scanf("%d", &n);
generateGrayarr(n);
}

int decimalToBinaryNumber(int x, int n)
{
int* binaryNumber = new int(x);
int i = 0;
while (x > 0) {
//printf("xwhile = %d\n", x);
binaryNumber[i] = x % 2;
//4. Se toma el valor de "x" generado previamente y se asigna a la posición i actual el residuo de "x % 2"
//printf("binaryNumber i = %d\n", x%2) ;
x = x / 2;
//5. Se divide entre dos para evaluar el siguiente valor mientras x sea mayor a 0
i++;
//6. Se incrementa "i" en 1 para avanzar a la siguiente posición del arreglo
```

```
}

for (int j = 0; j < n - i; j++){                    //7. En este ciclo for se asignan los 0s a la izquierda del arreglo de bits para completar la representación del número en binario
printf("0");
}

for (int j = i - 1; j >= 0; j--){                    //8. Este ciclo for toma e imprime los valores del arreglo de la posición MAYOR a la MENOR, por ejemplo, si el arreglo es {0,1}, este ciclo lo imprime como 10
printf("%d", binaryNumber[j]);
//9. De esta forma se logra generar los patrones de bits de 0 a 2^n-1, donde cada patrón sucessivo difiere en 1 bit
}

}


int generateGrayarr(int n)
{
int N = 1 << n;
//1. N se inicializa con 1*2^n, donde n es el número que se ingresó del cual se desea saber su secuencia en Grey Code
```

	`for (int i = 0; i < N; i++) {`
	`int x = i ^ (i >> 1);` `//2. El valor de x que se genere de esta línea es el que nos permite que cada arreglo de bits tenga un bit diferente al arreglo pasado por el XOR que produce un toggle`
	`//printf("x = %d\n",x);`
	`decimalToBinaryNumber(x, n);` `//3. Se ejecuta la función de decimal a binario proporcionando los valores de "x" y "n"`
	`printf("\n");`
	`}`
	`}`