# Guía 2

Rodrigo Estay

Usario de Timus: RodrigoEstay

ID de Timus:

**Problema 1:**

```c
#include <stdio.h>
#include <stdlib.h>

/* Programa hecho sin ayuda. */

int main (){

    /* El arreglo "prec" almacenara los ultimos dos numeros de la sucesion,
    y tambien introducimos los primeros dos terminos de la sucesion.*/

    int n, prec[2]={1,1}, sum=0, i;
    do{ // Aqui seguiremos escaneando "n" hasta que introduzcan uno positivo y menor a 47.
        printf("Introduzca un numero POSITIVO (maximo 46 debido a overflow):\n");
        scanf("%d", &n);
    }while(n<=0 || n>46);
    if(n==1){
        printf("El primer termino de la sucesion de Fibonacci es:\n");
    }
    else{
        printf("los primeros %d terminos de la sucesion de Fibonacci son:\n", n);
    }
    for(i=0;i<n;++i){
        if(i<2){ // Con este if nos aseguramos que se impriman los primeros terminos.
            printf("%d ", prec[i%2]);
        }
        else{ //Le vamos asignando los terminos a la posicion 1 o 0 de "prec".
            sum=prec[1]+prec[0];
            prec[i%2]=sum;
            printf("%d ", sum);
        }
    }
    printf("\n");
    return 0;
}
```

**Problema 2:**

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Programa hecho sin ayuda. */

int main(){
    int n, i;
    do{ // Escaneamos el input hasta que se introduzca un "n" positivo o igual a 0 y menor a 32 para
        // que no se produzca un overflow.
        printf("Ingrese un numero que NO SEA NEGATIVO y menor a 32 (debido a overflow):\n");
        scanf("%d", &n);
    }while(n<0 || n>31);
    if(n==0){
        printf("No puefo imprimir las 0 primeras potencias de 2.\n");
        return 0;
    }
    if(n==1){
        printf("La primera potencia de 2:\n");
    }
    else{
        printf("Las %d primeras potencias de 2:\n", n);
    }
    for(i=0;i<n;++i){ // Imprimimos las "n" primeras potencias de 2 con la funcion pow.
        printf("%d ", (int)pow(2,i)); // Hacemos casting a int.
        }
    printf("\n");
    return 0;
}
```

**Problema 3:**

```c
#include <stdio.h>
#include <stdlib.h>

/* Programa hecho sin ayuda. */

int main(){
    int n, i, temp, k, sum;
    do{ // Nos aseguramos que el "n" ingresado sea mayor a 0.
        printf("Ingrese un numero MAYOR QUE 0:\n");
        scanf("%d", &n);
    }while(n<=0);
    int data[n];
    printf("Ingrese los %d numeros:\n", n);
    for(i=0;i<n;++i){
        scanf("%d", &data[i]);
    }

    /* Lo siguiente es un algoritmo creado para ordenar los numeros de mayor a menor, lo que
    se hace es ir comparando los elementos de los extremos del arreglo con los del centro,
    haciendo que el elemento del extremo derecho (data[n-i-1], el "-1" esta porque los arreglos
    empiezan del 0) sea el menor del grupo, mientras el del extremo izquierdo (data[i]) es el mayor,
    realizando un intercambio cada vez que se encuentra un elemento menor o mayor, y a medida que
    el "i" aumenta, el grupo se va achicando hasta que ya se ordenaron todos los elementos. */

    for(i=0;i<n;++i){
        for(k=i;k<n-i;++k){
            if(data[k]>data[i]){
                temp=data[i];
                data[i]=data[k];
                data[k]=temp;
            }
            if(data[k]<data[n-i-1]){
                if(data[k]<data[n-i-1]){
                    temp=data[n-i-1];
                    data[n-i-1]=data[k];
                    data[k]=temp;
                }
            }
        }
    }
    for(i=0, sum=0;i<n;++i){
        sum+=data[i]; // Calculamos la suma de todos los datos para luego sacar el promedio.
    }

    /* Como se dijo anteriormente, el menor de los elementos va a ser el elemento del extremo
    derecho, mientras el mayor va a ser el del extremo izquierdo, "sum/n" sera el promedio
    truncado a la unidad, finalmente usamos un operador ternario para ver si la cantidad de
    elementos ingresados es par o impar, si es par significa que la mediana sera igual al
    promedio de los dos elementos del medio, en caso de que "n" sea impar solo habra un
    elemento en el medio, y ese sera la mediana. */

    printf("El menor:\t%d\nEl mayor:\t%d\nElpromedio:\t%d\nLa mediana:\t%d\n", data[n-1],
        data[0], sum/n, (n%2==0)?(data[n/2]+data[n/2-1])/2:data[n/2]);
    printf("NOTA: Los valores anteriores estan truncados a la unidad.\n");
    return 0;
}
```

**Problema 4:**

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Programa hecho sin ayuda. */

int main(){
    int i, n, comp=0;
    printf("Ingrese un numero entero:\n");
    scanf("%d", &n);
    if(n<0){ // Si es negativo trabajamos con su valor absoluto.
        n=abs(n);
    }
    if(n==0 || n==1){ // Por definicion el 0 y 1 no son nada.
        printf("No es ni primo ni compuesto.\n");
        return 0;
    }
    for(i=2;i<sqrt(n);++i){ // Analizamos hasta la raiz para mas eficiencia.
        if(n%i==0){
            printf("Es compuesto.\n");
            return 0;
        }
    }
    printf("Es primo.\n"); // Si no se encontro divisor entonces es primo.
    return 0;
}

```

**Problema 5:**

```c
#include <stdio.h>
#include <stdlib.h>

/* Programa hecho sin ayuda. */

int main(){
    int n, i, temp, k;
    do{
        printf("Ingrese un numero MAYOR QUE 0:\n");
        scanf("%d", &n);
    }while(n<=0);
    int data[n], imp[n]; // Creamos un arreglo para todos los numeros y uno para los impares.
    printf("Ingrese los %d numeros:\n", n);
    for(i=0;i<n;++i){
        scanf("%d", &data[i]);
    }

    /* Este es el mismo algoritmo creado en el problema 3, para mas explicacion sobre
    como funciona este algoritmo revise la informacion en el problema 3.
    El unico cambio que se le efectuo fue el cambio de los signos de mayor o menor en
    los "if" para ordenarlos de menor a mayor en vez de mayor a menor. */

    for(i=0;i<n;++i){
        for(k=i;k<n-i;++k){
            if(data[k]<data[i]){
                temp=data[i];
                data[i]=data[k];
                data[k]=temp;
            }
            if(data[k]>data[n-i-1]){
                temp=data[n-i-1];
                data[n-i-1]=data[k];
                data[k]=temp;
            }
        }
    }
```

```
        }
    }
    printf("Pares de menor a mayor: ");

    /* Si el numero es par, se imprime, ya que todos los numeros ya estan ordenados de
    menor a mayor, si el numero es impar se almacena en el arreglo de "imp". */

    for(i=0,k=0;i<n;++i){
        if(data[i]%2==0){
            printf("%d ", data[i]);
        }
        else{
            imp[k]=data[i];
            ++k;
        }
    }
    printf("\nImpares de mayor a menor: ");

    /* Ya que ya almacenamos los impares en "imp" ordenados de menor a mayor,
    simplemente los imprimimos desde el final del arreglo hasta el principio
    para que esten ordenados de mayor a menor. */

    for(i=k-1;i>=0;--i){
        printf("%d ", imp[i]);
    }
    printf("\n");
    return 0;
}
```

**Problema 6:**

```
#include <stdio.h>
#include <stdlib.h>

/* Programa hecho sin ayuda. */

int main(){
    int n, i, j;
    do{ // Comprobamos que sea positivo el valor.
        printf("Introduzca un numero MAYOR A 0 (ADVERTENCIA, numeros altos pueden deformar la piramide):\n");
        scanf("%d", &n);
    }while(n<=0);
    int pascal1[n], pascal2[n];
    /* Designamos los primeros valores del triangulo de pascal. */
    pascal1[0]=1;
    pascal1[1]=1;
    /* Con este ciclo armamos cada piso del triangulo. */
    for(i=0;i<n;++i){
        /* Con este ciclo hacemos los espacios iniciales de cada linea para que el triangulo
        tome forma. */
        for(j=1;j<n-i;++j){
            printf("   ");
        }
        /* Aqui comprobamos si se tratan de los primeros dos pisos del triangulo,
        y les hacemos casos apartes ya que el algoritmo siguiente no es aplicable a
        estos dos primeros pasos. */
        if(i==0){
            printf("    1");
        }
        if(i==1){
            printf("    1     1");
        }
        if(i>1){
            /* Aqui generamos cada numero del respectivo piso. */
            for(j=0;j<=i;++j){
                /* Nos aseguramos que el principio y final del triangulo sean un 1. */
                if(j==0 || j==i){
```

```
                /* Nos aseguramos que el principio y final del triangulo sean un 1. */
                if(j==0 || j==i){
                    pascal2[j]=1;
                }
                /* Para cada posicion "j" del piso del triangulo, sera igual a la suma
                de los numeros del piso anterior en la misma posicion mas el anterior
                a este. */
                else{
                    pascal2[j]=pascal1[j]+pascal1[j-1];
                }
            }
            /*imprimimos los numeros del piso y asignamos los numeros al arreglo del piso
            anterior para ocuparlo en la siguiente iteracion. */
            for(j=0;j<=i;++j){
                pascal1[j]=pascal2[j];
                printf("%5d ", pascal1[j]); //el 5 es para darle forma al triangulo (genera espacios en blanco).
            }
        }
        printf("\n");
    }
    return 0;
}
```

**Problema 7:**

```
#include <stdio.h>
#include <stdlib.h>

/* Programa hecho sin ayuda. */

int main (){
    int n, i, j, k;
    /* Validamos la entrada. */
    printf("Ingrese un numero MAYOR QUE 2:\n");
    do{
        scanf("%d", &n);
    }while(n<=2);
    int num[n];
    printf("Ingrese los %d numeros:\n", n);
    for(i=0;i<n;++i){
        scanf("%d", &num[i]);
    }
    /* Aqui avanzamos por 3 posiciones distintas del arreglo, i que va desde el inicio hasta la posicion
    ante penultia, j que va desde la siguiente posicion de i hasta la penultima y k que va desde la
    siguiente posicion de j hasta la ultuma. Y si mientras imprimimos los valores en estas posiciones,
    terminaremos imprimiendo todas las posibles combinaciones de 3 de estos numeros, ya que no importa
    el orden de estas combinaciones. */
    printf("Las combinaciones posibles de 3 numeros de los %d numeros ingresados son:\n", n);
    for(i=0;i<n-2;++i){
        for(j=i+1;j<n-1;++j){
            for(k=j+1;k<n;++k){
                printf("%d %d %d\n", num[i], num[j], num[k]);
            }
        }
    }
    return 0;
}
```

**Problema 8:**

```c
#include <stdio.h>
#include <stdlib.h>

/* Programa hecho sin ayuda. */

int main (){
    int matrix[4][4], i, j, subDet1, subDet2, subDet3, subDet4;
    printf("Introduzca los 16 datos de la matriz 4x4:\n");
    for(i=0;i<4;++i){
        for(j=0;j<4;++j){
            scanf("%d", &matrix[i][j]);
        }
    }

    /* Ya que siempre sera una matriz 4x4, ocupamos la definicion de determinantes por submatrices,
    por lo tanto calculamos las 4 submatrices y las multiplicamos por su respectivo factor. */

    subDet1=matrix[1][1]*matrix[2][2]*matrix[3][3]+matrix[1][2]*matrix[2][3]*matrix[3][1]
           +matrix[1][3]*matrix[2][1]*matrix[3][2]-matrix[1][3]*matrix[2][2]*matrix[3][1]
           -matrix[1][1]*matrix[2][3]*matrix[3][2]-matrix[1][2]*matrix[2][1]*matrix[3][3];
    subDet2=matrix[1][0]*matrix[2][2]*matrix[3][3]+matrix[1][2]*matrix[2][3]*matrix[3][0]
           +matrix[1][3]*matrix[2][0]*matrix[3][2]-matrix[1][3]*matrix[2][2]*matrix[3][0]
           -matrix[1][0]*matrix[2][3]*matrix[3][2]-matrix[1][2]*matrix[2][0]*matrix[3][3];
    subDet3=matrix[1][0]*matrix[2][1]*matrix[3][3]+matrix[1][1]*matrix[2][3]*matrix[3][0]
           +matrix[1][3]*matrix[2][0]*matrix[3][1]-matrix[1][3]*matrix[2][1]*matrix[3][0]
           -matrix[1][1]*matrix[2][0]*matrix[3][3]-matrix[1][0]*matrix[2][3]*matrix[3][1];
    subDet4=matrix[1][0]*matrix[2][1]*matrix[3][2]+matrix[1][1]*matrix[2][2]*matrix[3][0]
           +matrix[1][2]*matrix[2][0]*matrix[3][1]-matrix[1][2]*matrix[2][1]*matrix[3][0]
           -matrix[1][1]*matrix[2][0]*matrix[3][2]-matrix[1][0]*matrix[2][2]*matrix[3][1];
    printf("El determinante es: %d\n", matrix[0][0]*subDet1-matrix[0][1]*subDet2+matrix[0][2]*subDet3-matrix[0][3]*subDet4);
    return 0;
}
```

## Problema 9:

```c
#include <stdio.h>
#include <stdlib.h>

/* El practicante me ayudo a entender el problema. El programa es hecho sin ayuda. */

int main (){
    int n, i, j, j1, i1, iMin, jMin, min;

    /* Validamos la entrada. */

    do{
        printf("Ingrese un numero MAYOR A 1:\n");
        scanf("%d", &n);
    }while(n<=1);
    int original[n][n], distancias[n][n];
    printf("Ingrese %d valores para la matriz %dx%d:\n", n*n, n, n);
    for(i=0;i<n;++i){
        for(j=0;j<n;++j){
            scanf("%d", &original[i][j]);
        }
    }
    printf("Matriz original:\n");
    for(i=0;i<n;++i){
        for(j=0;j<n;++j){
            printf("%d\t", original[i][j]);
        }
        printf("\n");
    }

    /* Recorremos la matriz dos veces para encontrar los valores minimos y sus respectivas distancias para cada
    valor de la matriz. */

    for(i=0;i<n;++i){
        for(j=0;j<n;++j){
            /* Asumimos valores iniciales, considerando de que i y j no sean considerados como posiciones
            asumidas. */
```

```
            /* Asumimos valores iniciales, considerando de que i y j no sean considerados como posiciones
            asumidas. */
            if(i==n-1 && j==n-1){
                min=abs(original[i][j]-original[n-2][n-2]);
                iMin=n-2;
                jMin=n-2;
            }
            else{
                min=abs(original[i][j]-original[n-1][n-1]);
                iMin=n-1;
                jMin=n-1;
            }
            for(i1=0;i1<n;++i1){
                for(j1=0;j1<n;++j1){
                    /* En el caso que llegamos a la posicion original del valor a comparar, nos los saltamos. */
                    if(i1==i && j1==j){
                        continue;
                    }
                    /* Si se encuentra un nuevo minimo, guardamos el minimo y su posicion. */
                    if(min>abs(original[i][j]-original[i1][j1])){
                        min=abs(original[i][j]-original[i1][j1]);
                        iMin=i1;
                        jMin=j1;
                    }
                    /* Si se encuentra una diferencia igual al minimo pero con una menor distancia, entonces
                    guardamos esta nueva distancia minima. */
                    else if(min==abs(original[i][j]-original[i1][j1]) && abs(i1-i)+abs(j1-j)<abs(iMin-i)+abs(jMin-j)){
                        iMin=i1;
                        jMin=j1;
                    }
                }
            }
            /* Guardamos la distancia minima respecto a las posiciones de la diferencia minima. */
            distancias[i][j]=abs(iMin-i)+abs(jMin-j);
        }
    }
    printf("Matriz distancias:\n");
    for(i=0;i<n;++i){
        for(j=0;j<n;++j){
            printf("%d\t", distancias[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

**Problema 10:**

```c
#include <stdlib.h>
#include <stdio.h>

/* Programa hecho sin ayuda. */

int main (){
    int n, a, sum=0, ai, i;
    scanf("%d%d", &n, &a);
    for(i=0;i<a;++i){ // Escaneamos el numero de autos para cada minuto.
        scanf("%d", &ai);

        /* Si la cantidad de autos supera el numero de autos que pueden pasar
        por minuto, significa que esa cantidad de autos se acomulan para el
        siguiente minuto, en caso contrario no se acomulan autos para el siguiente
        minuto. */

        if(ai-n+sum>0){
            sum=ai-n+sum;
        }
        else{
            sum=0;
        }
    }
    printf("%d\n", sum); // Imprimimos la cantidad de autos que quedaron acomulados.
    return 0;
}
```

**Problema 11:**

```c
#include <stdlib.h>
#include <stdio.h>

/* Programa hecho sin ayuda. */

int main (){
    int n, neg=0, sum=0, i;
    scanf("%d", &n);
    if(n==0){ // En el caso de que "n=0", se tendra que la respuesta es 1.
        printf("1\n");
        return 0;
    }
    /* Si el "n" ingresado es menor a n entonces hacemos que n sea positivo
    para que funcione el ciclo de mas adelante, y hacemos "neg=1", para que
    se cumpla el if de despues. */
    if(n<0){
        n=abs(n);
        neg=1;
    }
    for(i=1;i<=n;++i){ //Sumamos todos los numeros de 1 a "n".
        sum+=i;
    }

    /* En el caso de que el "n" ingresado era negativo se ejecutara el if siguiente
    que transformara la suma a negativo y luego le restamos 1, ya que la suma va de
    "n" a 1, por lo tanto se le suma 1 al final, lo cual en este casoes restarle 1 al
    positivo. */

    if(neg){
        sum=-(sum-1);
    }
    printf("%d\n", sum);
    return 0;
}
```

**Problema 12:**

```c
#include <stdlib.h>
#include <stdio.h>

/* Programa hecho sin ayuda. */

int main (){
    int n, k, ai, i, surv=0, left=0;
    scanf("%d%d", &n, &k);
    for(i=0;i<n;++i){ // Escaneamos la cantidad de booms booms de cada cargamento para cada bloque.

        /* Para cada cantidad de booms booms de cada cargamento, nos fijamos si
        es menor a la cantidad de androides en cada bloque, en tal caso, sumamos
        la resta de entre estos a la cantidad de supervivientes, en el caso contrario
        sumamos los booms booms restantes a "left". */

        scanf("%d", &ai);
        if(k-ai>0){
            surv=surv+k-ai;
        }
        else{
            left=left+ai-k;
        }
    }
    printf("%d %d\n", left, surv);
    return 0;
}
```

**Problema 13:**

```c
#include <stdio.h>
#include <stdlib.h>

/* Programa hecho sin ayuda. */

int main (){
    int n[3], i, j, k, eigenvalues=0;
    scanf("%d", &n[0]);
    int a[n[0]];

    /* Los siguientes 3 ciclos for son para escanear los "n" valores dados
    para cada caso, los asignamos a 3 arreglos diferentes en cada ciclo. */

    for(i=0;i<n[0];++i){
        scanf("%d", &a[i]);
    }
    scanf("%d", &n[1]);
    int b[n[1]];
    for(i=0;i<n[1];++i){
        scanf("%d", &b[i]);
    }
    scanf("%d", &n[2]);
    int c[n[2]];
    for(i=0;i<n[2];++i){
        scanf("%d", &c[i]);
    }

    /* Comparamos todos los valores del arreglo "b" con los del arreglo "a". */

    for(i=0;i<n[0];++i){
        for(j=0;j<n[1];++j){

            /* Si algun valor del arreglo "a" es igual al del arreglo "b"
            vemos si algun valor del arreglo "c" es tambien igual a este valor,
            en tal caso sumamos un "eigenvalue". */

            if(a[i]==b[j]){
                for(k=0;k<n[2];++k){
                    if(a[i]==c[k]){
                        ++eigenvalues;
                    }
                }
            }
        }
    }
    printf("%d\n", eigenvalues);
    return 0;
}
```

**Problema 14:**

```c
#include <stdlib.h>
#include <stdio.h>

/* Programa hecho sin ayuda. */

int main (){
    int n, i, maxMagic=0, maxSection; // "maxMagic" es 0 para que no tome basura en la linea 11.
    scanf("%d", &n);
    int magic[n];

    /* A medida que escaneamos datos, vamos trabajando con ellos, "maxSection=2" se debe que en
    el caso de que solo se ingresan 3 secciones, entonces la seccion del medio sera el 2. Luego
    se escanean datos y se suman para asignar como maxima magia la de los primeros 3 datos, una
    vez superados los 3 datos revisamos si la magia de las ultimas 3 secciones superan la magia
    maxima almacenada, en el caso de que si lo supere, lo asignamos como nueva magia maxima y la
    nueva seccion maxima sera la del medio de los 3, pero ya que los arreglos parten del 0, se
    le suma 1, lo cual queda solo en "i". */

    for(i=0, maxSection=2;i<n;++i){
        scanf("%d", &magic[i]);
        if(i<3){
            maxMagic+=magic[i];
            continue;
        }
        if(maxMagic<magic[i]+magic[i-1]+magic[i-2]){
            maxMagic=magic[i]+magic[i-1]+magic[i-2];
            maxSection=i;
        }
    }
    printf("%d %d\n", maxMagic, maxSection);
    return 0;
}
```

## En Timus:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7843436 | 08:12:27<br>9 Apr 2018 | RodrigoEstay | 1910. Titan Ruins: Hidden Entrance | GCC 7.1 | Accepted | | 0.015 | 208 KB |
| 7843418 | 07:34:47<br>9 Apr 2018 | RodrigoEstay | 1880. Psych Up's Eigenvalues | GCC 7.1 | Accepted | | 0.031 | 252 KB |
| 7843301 | 04:10:28<br>9 Apr 2018 | RodrigoEstay | 1991. The battle near the swamp | GCC 7.1 | Accepted | | 0.001 | 208 KB |
| 7843279 | 03:50:14<br>9 Apr 2018 | RodrigoEstay | 1068. Sum | GCC 7.1 | Accepted | | 0.015 | 208 KB |
| 7843266 | 03:32:00<br>9 Apr 2018 | RodrigoEstay | 1787. Turn for MEGA | GCC 7.1 | Accepted | | 0.015 | 208 KB |