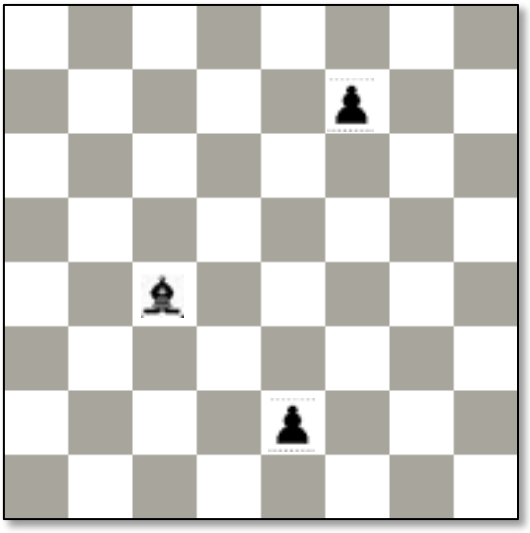
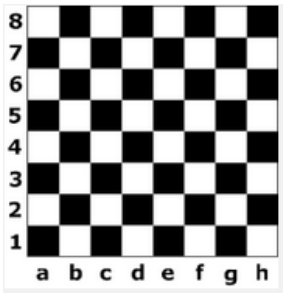
PROGRAMACIÓN 1
23 Enero de 2015

En las soluciones los ejercicios están completos para poderlos probar. En los ejercicios en los que sólo se solicitase el módulo, sólo había que hacer el módulo.

PREGUNTA 1 (2 puntos).

Tenemos un tablero de ajedrez, en el que están situadas las siguientes piezas: alfil, dos peones. A partir de la posición del alfil en el tablero debemos indicar a qué posiciones se puede desplazar siempre y cuando esas posiciones no estén ocupadas por los peones.





Realiza un programa en el que se soliciten las coordenadas de las figuras y con estos datos se llame a un módulo que muestre por pantalla las distintas posiciones del tablero que podría ocupar el alfil a partir de su posición actual. Escribe las estructuras de datos necesarias para ello.
Siguiendo el ejemplo anterior:

```
Introduce la posición del alfil: c4
Introduce la posición del peón 1: e2
Introduce la posición del peón 2: f7
Los posibles movimientos a realizar son:
b3  a2  d3  d5  e6  b5  a6
```

Importante: El tablero no es una matriz.

Solución:

```
#include<iostream>
using namespace std;

//Declaracion de tipos
typedef struct{
     char car;
     int pos;
}Tpunto;

void movimientos(Tpunto alfil, Tpunto peon1, Tpunto peon2) {
  char i;
  int j, ocupado;

  ocupado=0;
  i=alfil.car;
  j=alfil.pos;
  while (i>'a' && j>1 && !ocupado) {
    i--;
    j--;
    if ((i==peon1.car && j==peon1.pos) || (i==peon2.car && j==peon2.pos))
      ocupado=1;
    else cout << i << j << "  ";
  }

  ocupado=0;
  i=alfil.car;
  j=alfil.pos;
  while (i<'h' && j>1 && !ocupado) {
    i++;
    j--;
    if ((i==peon1.car && j==peon1.pos) || (i==peon2.car && j==peon2.pos))
      ocupado=1;
    else cout << i << j << "  ";
  }

  ocupado=0;
  i=alfil.car;
  j=alfil.pos;
  while (i<'h' && j<8 && !ocupado) {
    i++;
    j++;
    if ((i==peon1.car && j==peon1.pos) || (i==peon2.car && j==peon2.pos))
      ocupado=1;
    else cout << i << j << "  ";
  }
```

```
  ocupado=0;
  i=alfil.car;
  j=alfil.pos;
  while (i>'a' && j<8 && !ocupado) {
    i--;
    j++;
    if ((i==peon1.car && j==peon1.pos) || (i==peon2.car && j==peon2.pos))
      ocupado=1;
    else cout << i << j << "  ";
  }

}

int main(){

   Tpunto alfil, peon1, peon2;

  cout << "Introduce la posición del alfil: " ;
  cin >> alfil.car >> alfil.pos;
  cout << "Introduce la posición del peón 1: " ;
  cin >> peon1.car >> peon1.pos;
  cout << "Introduce la posición del peón 2: " ;
  cin >> peon2.car >> peon2.pos;

  cout << "Los posibles movimientos a realizar son: " << endl;
  movimientos(alfil, peon1, peon2);
  cout << endl;
}
```

PREGUNTA 2. (2 puntos)

Realiza una función recursiva en la que a partir de un número entero muestre por pantalla los divisores de ese número. La función sólo contará con un único parámetro.

Por ejemplo:

divisores(6) muestra 6, 3, 2, 1

Importante: este ejercicio sólo puntuará si se realiza de forma recursiva.
Importante: No se podrán utilizar arrays ni registros para almacenar los datos.

Solución ingenios para salvar el problema de un solo parámetro:

```
#include <iostream>
using namespace std;

void test(int n, int p) {

  if (p%n==0) {
   cout << n << " " ;
  }
  if (n!=1) {
    test(n-1,p);
   }
  }

void divisores(int n){
  int p;
  p=n;
  test(n,p);
}

int main() {
   int num;

   cout << "Número: ";
   cin >> num;
   divisores(num);
   cout << endl;
}
```

Posibles soluciones válidas para el examen:
// controlando los primos pero sin controlar cuando sus divisores son primos. Ejemplo 121

```
#include <iostream>
using namespace std;

void divisores (int n){

  cout << n << " ";

  if (n>1) {
    if (n%2!=0 && n%3!=0 && n%5!=0 && n%7!=0)  //para los números primos
       divisores(n/n);
    else{
      if (n%2==0) {
         divisores(n/2);
         cout << " " << 2  << " ";
      }
      if (n%3==0) {
         divisores(n/3);
         cout << " " << 3  << " ";
      }
      if (n%5==0){
         divisores(n/5);
         cout << " " << 5  << " ";
      }
      if (n%7==0){
         divisores(n/7);
         cout << " " << 7  << " ";
      }
    }
  }
}
```

```
int main() {
   int num;

   cout << "Número: ";
   cin >> num;
   divisores(num);
   cout << endl;
}
```

Posible solución: // controlando cuando sus divisores son primos. Ejemplo 121

```
#include <iostream>
#include <cmath>
using namespace std;

void divisores (int n){

  cout << n << " ";

  if (n>1) {
    if (n%2!=0 && n%3!=0 && n%5!=0 && n%7!=0)  //para los números primos
       if (sqrt(n)-trunc(sqrt(n))==0) divisores(sqrt(n));
       else divisores(n/n);
    else{
      if (n%2==0) {
         divisores(n/2);
         cout << " " << 2  << " ";
      }
      if (n%3==0) {
         divisores(n/3);
         cout << " " << 3  << " ";
      }
      if (n%5==0){
         divisores(n/5);
         cout << " " << 5  << " ";
      }
      if (n%7==0){
         divisores(n/7);
         cout << " " << 7  << " ";
      }
    }
  }
}

int main() {
   int num;

   cout << "Número: ";
   cin >> num;
   divisores(num);
   cout << endl;
}
```

PREGUNTA 3. (2 puntos)

Dos números se consideran "parientes" si contienen el mismo número de dígitos y además sus dígitos impares suman la misma cantidad. Por ejemplo: 2190 y 3470. Realiza un módulo que tome como parámetros los dos números y el total de dígitos de que constan y devuelva un valor lógico indicando si se trata de números parientes o no.

Importante: No se podrán utilizar arrays ni registros para almacenar los datos.

Posibles soluciones:

Solución en la que se supone que nos indican el total de dígitos y son los mismos en ambos números

```
#include  <iostream>
using namespace std;

bool parientes (int num1, int num2, int digitos) {
  int suma1, suma2;
  bool res;

  suma1=0;
  suma2=0;
  res=false;

  while (digitos>0) {
    if (num1%2 != 0)
      suma1 = suma1 + num1%10;
    if (num2%2 !=0)
      suma2 = suma2 + num2%10;
    num1=num1/10;
    num2=num2/10;
    digitos--;
  }

  if (suma1==suma2)
     res=true;

  return (res);
}

int main(){
  int n1, n2, digit;  // los números y su total de dígitos
  int p1, p2;         // variables auxiliares para contar sus dígitos
  bool mismos;

  digit=0;
  mismos=false;
```

```
  do{
    cout << "Introduce un número: ";
    cin >> n1;
    cout << "Introduce otro número: ";
    cin >> n2;

    p1=n1;
    p2=n2;
    while (p1>0 &&  p2>0) {
      digit++;
      p1=p1/10;
      p2=p2/10;
    }
    if (p1==p2)
       mismos=true;
    else
       cout << "Los números deben tener el mismo número de dígitos " << endl;
} while (mismos!=true);

if (parientes(n1, n2, digit))
  cout << "Los números son parientes" << endl;
else
  cout << "Los números no son parientes" << endl;
}
```

Solución en la que se supone que NO nos indican el total de dígitos

```
#include  <iostream>
using namespace std;

bool parientes (int num1, int num2) {
  int suma1, suma2;
  bool res;

  suma1=0;
  suma2=0;
  res=false;

  while (num1>0 && num2>0) {
    if (num1%2 != 0)
      suma1 = suma1 + num1%10;
    if (num2%2 !=0)
      suma2 = suma2 + num2%10;
    num1=num1/10;
    num2=num2/10;
  }
```

```
  if (num1==num2 && suma1==suma2)
     res=true;

  return (res);
}

int main(){
  int n1, n2;

  cout << "Introduce un número: ";
  cin >> n1;
  cout << "Introduce otro número: ";
  cin >> n2;

  if (parientes(n1, n2))
    cout << "Los números son parientes" << endl;
  else
    cout << "Los números no son parientes" << endl;
}
```

Solución en la que se supone que nos indican el total de dígitos y puede no coincidir

```
#include  <iostream>
using namespace std;

bool parientes (int num1, int num2, int digit1, int digit2) {
  int suma1, suma2;
  bool res;

  suma1=0;
  suma2=0;
  res=false;

  if (digit1 == digit2){
     while (digit1>0) {
       if (num1%2 != 0)
          suma1 = suma1 + num1%10;
       if (num2%2 !=0)
          suma2 = suma2 + num2%10;
       num1=num1/10;
       num2=num2/10;
       digit1--;
     }
     if (suma1==suma2)
        res=true;
  }
  return (res);
}
```

```
int main(){
  int n1, n2, digit1, digit2;    //valores que pasaremos al módulo
  int p1, p2;                     //variables auxiliares para no modificar los números

  digit1=0;
  digit2=0;

  cout << "Introduce un número: ";
  cin >> n1;
  cout << "Introduce otro número: ";
  cin >> n2;

  p1=n1;
  p2=n2;
  while (p1>0) {
    digit1++;
    p1=p1/10;
  }
  while (p2>0) {
    digit2++;
    p2=p2/10;
  }

if (parientes(n1, n2, digit1, digit2))
  cout << "Los números son parientes" << endl;
else
  cout << "Los números no son parientes" << endl;
}
```

PREGUNTA 4. (1+1+2 puntos)

Una cadena de televisión de pago a la carta nos contrata para crear un programa que controle los servicios prestados a sus 158 abonados. Queremos tener información del abonado (nombre, dirección, teléfono, cuenta cobro, etc.), de sus cuotas fijas y de sus consumos. Las cuotas fijas pueden ser de 3 tipos: Básica, Plata y Oro, y cada una tendrá un coste de 30€, 40€ y 50€ respectivamente. Además, un abonado puede ver un número limitado (100) de programas de pago que no entran dentro de su cuota (estrenos de cine, partidos de fútbol, etc.). De estos servicios nos interesa almacenar: El nombre del programa, la fecha y hora de inicio y finalización, así como el importe.

a) Define en C las estructuras de datos necesarias para almacenar la información anterior.

```
const int ktam=30;
const int kcuenta=25;
const int ktelef=10;
const int kmax=100;
const int ktope=158;

typedef char Tcadena[ktam];
typedef char Tcuenta[kcuenta];
typedef char Ttelefono[ktelef];

typedef struct{
  int dia, mes, anyo;
} Tfecha;

typedef struct{
  int hora, min;
}Thora;

typedef struct{
  Tcadena nombre;
  Tfecha fecha;
  Thora horaInic;
  Thora horaFin;
  int importe;
} Tprograma;
```

```
typedef struct {
  Tcadena nombre;
  Tcadena direccion;
  Ttelefono telefono;
  Tcuenta cuenta;
  char tipoContrato;  // b ó B Básica, P o p Plata, O ó o Oro
  int consumos;
  Tprograma programas[kmax];
}Tabonado;

  // se puede definir un nuevo tipo como vector de estructuras
  // typedef Tabonado Tabonados[158];
   // o bien crear el vector directamente en el main

int main() {
  Tabonado abonados[ktope];
}
```

b) Crea un módulo en C que reciba el número de abonado y mes, y devuelva el importe a pagar en este periodo (cuota fija + consumos).

```
int calcularImporte (Tabonado abonados[], int total, Ttelefono abon, int mes) {
 int i, j, suma;
 i=0;

 while (i<ktope && strcmp(abonados[i].telefono, abon)!=0)
   i++;

 suma=0;
 switch(abonados[i].tipoContrato) {
   case 'b': case 'B': suma = suma + 30; break;
   case 'p': case 'P': suma = suma + 40; break;
   case 'o': case 'O': suma = suma + 50; break;
   defaul: suma=suma;   //este caso no es necesario si se ha validado la entrada
 }
 for (j=0; j<abonados[i].consumos; j++) {
   if (abonados[i].programas[j].fecha.mes == mes)
     suma = suma + abonados[i].programas[j].importe;
 }
 return(suma);
}
```

c) Escribe un módulo en el que dado un año y un programa determinado devuelva al main el total de abonados que han visto ese programa en ese año.

```
//si un abonado lo ha visto más de una vez solo se le contará como una vez
int programaVisto(Tabonado abonados[], int total, int anyo, Tcadena programa){
 int i, j, suma;
 suma=0;

 for (i=0; i<total; i++) {
   j=0;
   while (j<abonados[i].consumos && strcmp(abonados[i].programas[j].nombre, programa)!=0)
     j++;
   if (j!= abonados[i].consumos)  //eso quiere decir que salío del bucle pq encontró el programa
     if (abonados[i].programas[j].fecha.anyo == anyo)
       suma=suma+1;
 }
 return(suma);
}
```