

```

//mostrar los n primeros capicuas
//n se pide por el teclado
//version con funciones
#include<iostream>
using namespace std;

//devuelve un numero
//formado por las cifras del parametro
//en orden inverso
int reves(int n)
{
    //res es el resultado
    int res=0;
    while(n!=0)
    {
        //añadir la cifra mas a la derecha
        res=res*10+n%10;
        //eliminar la cifra mas a la derecha
        n=n/10;
    }
    return res;
}

//devuelve true si es capicua,
//false en caso contrario
bool capicua(int n)
{
    //la expresion es booleana
    //coincide con el tipo de retorno
    //si es cierta es capicua
    //si es falsa no lo es
    return n==reves(n);
}

int main()
{
    //notar la inicializacion de n, comenzamos a probar
    //por el primero, el cero
    int n=0,contador=0,cantidad;
    cout<<"Cuantos capicuas quieres:";
    cin>>cantidad;
    //mientras no encuentre 'cantidad' capicuas
    while(contador<cantidad)
    {
        if (capicua(n))
        {
            //lo cuento
            contador++;
            //lo muestro
            cout<<n<<endl;
        }
        //siguiente numero para probar
        n++;
    }
}

```

```

//mostrar los n primeros numeros
//que contienen la cifra x
//n y x se piden por el teclado
#include<iostream>
using namespace std;

//devuelve true si n contiene x
bool contienex(int n,int x)
{
    while(n!=0)
    {
        if (n%10==x)
            return true;
        n/=10;
    }
    return false;
}

//pide un entero entre 0 y 9
//es decir, una cifra
int pide_cifra(void)
{
    int num;
    do{
        cin>>num;
    }while(num<0||num>9);
    return num;
}

int main()
{
    int n=0,conta=0,cuantos,cifra;
    cout<<"Cuantos quieres:";
    cin>>cuantos;
    cout<<"Cifra a buscar:";
    cifra=pide_cifra();
    while(conta<cuantos)
    {
        if(contienex(n,cifra))
        {
            cout<<n<<endl;
            conta++;
        }
        n++;
    }
}

```

```

//Buscar el mayor número entero
//(menor que 45000 por cuestiones de rango)
// cuyo cuadrado se escriba con
//dígitos distintos.
//Ejemplos de cuadrados que cumplen esa condición:
//13^2 = 169 286^2=81796 32043^2=1026753849
//version con funciones

#include<iostream>
using namespace std;

//devuelve true si n contiene x
bool contienex(int n,int x)
{
    while(n!=0)
    {
        if (n%10==x)
            return true;
        n/=10;
    }
    return false;
}

//devuelve true si todos los digitos de n
//son distintos
bool digitos_distintos(int n)
{
    //mientras queden cifras en n
    while(n!=0)
    {
        //cifra menos significativa de n
        int cifra=n%10,
            //resto de las cifras
            resto_cifras=n/10;
        //si resto_cifras contiene a cifra,
        //cifra se repite
        if (contienex(resto_cifras,cifra))
            return false;
        //siguiente cifra de n
        n/=10;
    }
    //si llega aqui es que no se ha encontrado
    //ninguna repetida, luego se devuelve true
    return true;
}

//devuelve el mayor numero que cumple la condicion
int mayor_numero_bla_bla(int limite)
{
    for (int i=limite;i>0;i--)
    {
        int cuadrado=i*i;
        //si digitos_distintos es true
        if (digitos_distintos(cuadrado))
            //se retorna ese numero
            return i;
    }
    //si no se encuentra ninguno
    //usamos -1 para indicarlo
    //en este caso esto no se va a producir
    return -1;
}

int main()
{
    cout<<mayor_numero_bla_bla(45000);
}

```

```

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

//devuelve un vector con las copias unicas,
//es decir, se expurgan las repeticiones
vector<int> unicos1(const vector<int>&a)
{
    //vector de booleanos, false es que esta repetido
    //el elemento correspondiente de a
    vector<bool> b(a.size(),true);
    //se compara cada elemento con los siguientes
    for (int i=0;i<a.size()-1;i++)
        for (int j=i+1;j<a.size();j++)
            //si esta repetido se marca a false
            if (a[i]==a[j])
                b[j]=false;
    //calculo el numero de unicos = numero de true
    int nunicos=0;
    for (int i=0;i<b.size();i++)
        if (b[i]) nunicos++;
    //los que se corresponden con true son unicos
    vector<int> unicos(nunicos);
    int j=0;
    for (int i=0;i<a.size();i++)
        if (b[i])
            {
                unicos[j]=a[i];
                j++;
            }
    return unicos;
}

//devuelve un vector de copias unicas,
//version con push_back
vector<int> unicos2(const vector<int>&a)
{
    //vector de booleanos, false es que esta repetido
    //el elemento correspondiente de a
    vector<bool> b(a.size(),true);
    //se compara cada elemento con los siguientes
    for (int i=0;i<a.size()-1;i++)
        for (int j=i+1;j<a.size();j++)
            //si esta repetido se marca a false
            if (a[i]==a[j])
                b[j]=false;
    //los que se corresponden con true son unicos
    //unicos se declara sin tamaño inicial
    vector<int> unicos;
    int j=0;
    for (int i=0;i<a.size();i++)
        if (b[i])
            //Se añaden al vector
            unicos.push_back(a[i]);
    return unicos;
}

```

```

//alternativa a return, paso por referencia NO constante
//devuelve un vector de copias unicas.
//como es void, NO se puede usar la llamada
//como parte de una expresion
//version con push_back
void unicos3(const vector<int>&a, vector<int>&unicos)
{
    //vector de booleanos, false es que esta repetido
    //el elemento correspondiente de a
    vector<bool> b(a.size(),true);
    //se compara cada elemento con los siguientes
    for (int i=0;i<a.size()-1;i++)
        for (int j=i+1;j<a.size();j++)
            //si esta repetido se marca a false
            if (a[i]==a[j])
                b[j]=false;
    //los que se corresponen con true son unicos
    //unicos debe de estar vacio OJO
    //se podría utilizar clear
    unicos=vector<int>();
    int j=0;
    for (int i=0;i<a.size();i++)
        if (b[i])
            //Se añaden al vector
            unicos.push_back(a[i]);
}

int main()
{
    vector<int>a(10);
    //prueba de la version sin push_back
    for (int i=0;i<a.size();i++)
        cin>>a[i];
    vector<int>b;
    b=unicos1(a);
    for (int i=0;i<b.size();i++)
        cout<<b[i]<<' ';
    cout<<endl;
    //prueba de la version con push_back
    b=unicos2(a);
    for (int i=0;i<b.size();i++)
        cout<<b[i]<<' ';
    cout<<endl;
    //prueba de la version sin return
    unicos3(a,b);
    for (int i=0;i<b.size();i++)
        cout<<b[i]<<' ';
}

```

```

//Sobrecarga de operadores para matrices
#include<iostream>
using namespace std;
#include<vector>
//sobrecarga del operador * para matrices
vector<vector<float> > operator*(const vector<vector<float> > &a,
                                const vector<vector<float> > &b)
{
    //el vector resultado tiene de dimensiones el numero de filas
    //de a y el numero de columnas de b
    vector<vector<float> > c(a.size(),vector<float>(b[0].size()));
    //se recorre la matriz resultado
    for (int i=0;i<a.size();i++)
        for (int j=0;j<b[0].size();j++)
            {
                //se inicializa el elemento i j del resultado a cero
                c[i][j]=0;
                //se recorre la fila i de a y la columna j de b
                //multiplicandolas escalarmente
                for (int k=0;k<a[0].size();k++)
                    c[i][j]+=a[i][k]*b[k][j];
            }
    return c;
}

//sobrecarga del operador << para matrices
//se pasa por referencia no constante un flujo de salida
//se devuelve el flujo de salida modificado
ostream& operator<<(ostream& o,const vector<vector<float> > &a)
{
    for (int i=0;i<a.size();i++)
        {
            for (int j=0;j<a[0].size();j++)
                //se insertan el elemento en el flujo de salida
                o<<a[i][j]<<' ';
            o<<endl;
        }
    //se devuelve el flujo de salida
    return o;
}

//sobrecarga del operador >> para matrices
//se pasa por referencia no constante un flujo de entrada
//se devuelve el flujo de entrada modificado
istream& operator>>(istream& in, vector<vector<float> > &a)
{
    for (int i=0;i<a.size();i++)
        {
            for (int j=0;j<a[0].size();j++)
                {
                    cout<<'<<i<<"["<<j<<"=";
                    //se extrae el elemento del flujo de entrada
                    in>>a[i][j];
                }
        }
    //se devuelve el flujo de entrada
    return in;
}

int main()
{
    int fa,ca,cb;
    cout<<"filas de a, columnas de a, columnas de b:";
    cin>>fa>>ca>>cb;
    vector<vector<float> > a(fa,ca),b(ca,cb);
    //solo se puede hacer si se han sobrecargado los operadores
    cin>>a;
    cin>>b;
    cout<<a*b;
}

```

```

//Mezcla de dos vectores ordenados en un tercero
//también ordenado
#include <iostream>
using namespace std;
#include<vector>

//version sin push_back
void mezcla1(const vector<int>&x,const vector<int>&y,vector<int>&sal)
{
    int i_x,i_y,i_sal;
    i_x=0;
    i_y=0;
    i_sal=0;
    //se asigna el tamaño al resultado
    sal=vector<int>(x.size()+y.size());
    //mientras no acabe con alguno
    while(i_x<x.size()||i_y<y.size())
    {
        //si no he acabado con ninguno
        if (i_x<x.size()&&i_y<y.size())
        {
            //si el actual de x es menor que
            //el actual de y
            if (x[i_x]<y[i_y])
            {
                sal[i_sal]=x[i_x];
                i_sal++;
                i_x++;
            }
            else
            {
                sal[i_sal]=y[i_y];
                i_sal++;
                i_y++;
            }
        }
        //he acabado con alguno
        else
        {
            //si no he acabado con x
            if (i_x<x.size())
            {
                sal[i_sal]=x[i_x];
                i_sal++;
                i_x++;
            }
            //si no he acabado con y
            if (i_y<y.size())
            {
                sal[i_sal]=y[i_y];
                i_sal++;
                i_y++;
            }
        }
    }
}

```

```

//version con push_back
void mezcla2(const vector<int>&x,const vector<int>&y,vector<int>&sal)
{
    int i_x,i_y,i_sal;
    i_x=0;
    i_y=0;
    i_sal=0;

    //cuidado, si se añaden elementos con push_back
    //el vector sal debe estar vacío
    //se puede hacer así:
    //sal=vector<int>();
    //o bien
    sal.clear();
    while(i_x<x.size()||i_y<y.size())
    {
        if (i_x<x.size()&&i_y<y.size())
        {
            if (x[i_x]<y[i_y])
            {
                sal.push_back(x[i_x]);
                i_x++;
            }
            else
            {
                sal.push_back(y[i_y]);
                i_y++;
            }
        }
        else
        {
            if (i_x<x.size())
            {
                sal.push_back(x[i_x]);
                i_x++;
            }
            if (i_y<y.size())
            {
                sal.push_back(y[i_y]);
                i_y++;
            }
        }
    }
}

int main()
{
    vector<int> x(5),y(3),z1,z2;

    x[0]=1; x[1]=3; x[2]=5; x[3]=7; x[4]=8;

    y[0]=2; y[1]=2; y[2]=10;

    mezcla1(x,y,z1);

    for (int i=0;i<z1.size();i++)
        cout<<z1[i]<<' ';
    cout<<endl;

    mezcla2(x,y,z2);

    for (int i=0;i<z2.size();i++)
        cout<<z2[i]<<' ';
}

```

```

//máximo de la suma de elementos contiguos de un
//vector
#include<iostream>
#include<vector>
using namespace std;

//forma simple poco eficiente
int a(const vector<int> &x)
{
    int maximo=x[0];
    //primer extremo
    for (int i=0;i<x.size()-1;i++)
        //segundo extremo
        for(int j=i+1;j<x.size();j++)
            {
                int suma=0;
                //suma entre los extremos
                for (int k=i;k<=j;k++)
                    suma=suma+x[k];
                //comparacion de la suma con el maximo
                //fuera del bucle anterior
                if (suma>maximo)
                    maximo=suma;
            }
    return maximo;
}

//forma menos simple mas eficiente
int b(const vector<int> &x)
{
    int maximo=x[0];
    //inicio
    for (int i=0;i<x.size();i++)
        {
            int suma=0;
            //suma hasta el fin
            for(int j=i;j<x.size();j++)
                {
                    suma=suma+x[j];
                    //comparación de la suma con el maximo
                    //en cada iteracion, importante
                    if (suma>maximo)
                        maximo=suma;
                }
        }
    return maximo;
}

int main()
{
    vector<int>x(10);
    x[0]=43; x[1]=-52; x[2]=78; x[3]=15; x[4]=-67;
    x[5]=39; x[6]=93; x[7]=-99;x[8]=-32; x[9]=48;
    cout<<a(x)<<' '<<b(x);
}

```

```

//Función que comprime un vector "casi vacío" y su contrapartida,
//descomprime un vector "casi vacío" (ejercicio de examen).
//Un vector "casi vacío" es aquel que contiene mayoritariamente
//elementos nulos. Para ahorrar espacio se guarda el tamaño del
//vector y (a continuación) la posición y el contenido de los
//elementos no nulos en un vector convencional. Por ejemplo el
//vector {0,0,0,1,0,0,-7,0,0} se podría guardar como
//{9,3,1,6,-7}. La función comprime se ha escrito usando
// push_back y sin usarla
#include<iostream>
#include<vector>
using namespace std;

//descomprime un vector casi vacío en un vector convencional
vector<int> descomprime(const vector<int> &a)
{
    //inicializo el vector de longitud a[0] a 0
    vector<int>b(a[0],0);
    //recorro el comprimido
    //desde la posición 1, van alternandose
    //posición y contenido, el índice aumenta
    //de dos en dos
    for (int i=1;i<a.size()-1;i=i+2)
        b[a[i]]=a[i+1];
    return b;
}

//descomprime con push_back es un dolor, no la hago

//comprime un vector convencional
//version sin push_back, es tambien un dolor pero menos
vector<int> comprime1(const vector<int> &a)
{
    //cuento los no nulos
    int conta=0;
    for (int i=0;i<a.size();i++)
        if (a[i]!=0)
            conta++;
    //guardo los indices de los no nulos
    vector<int>indices_no_nulos(conta);
    int indice=0;
    for(int i=0;i<a.size();i++)
        if (a[i]!=0)
            {
                indices_no_nulos[indice]=i;
                indice++;
            }
    //el tamaño del vector comprimido
    vector<int>b(2*conta+1);
    //en el primer elemento va el tamaño original
    b[0]=a.size();
    //para los indices de los no nulos
    for (int i=0;i<indices_no_nulos.size();i++)
        {
            //alternativamente se guarda el indice
            b[i*2+1]=indices_no_nulos[i];
            //el contenido del elemento en ese indice
            b[i*2+2]=a[indices_no_nulos[i]];
        }
    return b;
}

```

```

//comprime un vector convencional
//version con push_back
vector<int> comprime2(const vector<int> &a)
{
    vector<int> b;
    //en el primer elemento de b va el tamaño de a
    b.push_back(a.size());
    //se recorre a y se guardan los no nulos
    for (int i=0;i<a.size();i++)
        //si a[i] no es nulo
        if (a[i]!=0)
            {
                //se guarda el indice
                b.push_back(i);
                //se guarda el elemento
                b.push_back(a[i]);
            }
    return b;
}

//sobregarga de << (no se pedia es para abreviar)
template<typename T> ostream& operator<<(ostream&o,const vector<T> &a)
{
    for (int i=0;i<a.size();i++)
        o<<a[i]<<" ";
    return o;
}

//sobregarga de >> (no se pedia es para abreviar)
template<typename T> istream& operator>>(istream &i,vector<T> &a)
{
    for (int j=0;j<a.size();j++)
        i>>a[j];
    return i;
}

int main()
{
    vector<int> a(8),b;
    cin>>a;
    //prueba de comprime1
    b=comprime1(a);
    cout<<b<<endl;
    cout<<a<<endl;
    //prueba de descomprime
    a=descomprime(b);
    cout<<a<<endl;
    //prueba de comprime2
    b=comprime2(a);
    cout<<b<<endl;
    cout<<a<<endl;
    a=descomprime(b);
    cout<<a<<endl;
}

```

```

//Analizar la paridad de un vector de booleanos
#include<iostream>
using namespace std;
#include<vector>

//version iterativa se cuentan los true
//se devuelve true si hay un numero par de true
bool es_par1(const vector<bool>&a)
{
    int conta=0;
    for (int i=0;i<a.size();i++)
        if(a[i])
            conta++;
    return conta%2==0;
}

//otra version iterativa que inspira la recursiva
bool es_par1(const vector<bool>&a)
{
    //inicialmente si el primero es true, es impar (1 true)
    //si es false es par (0 true)
    bool res=!a[0];
    //para los restantes
    for (int i=1;i<a.size();i++)
        //si encuentra un true, cambia res de true a false
        //y viceversa
        if (a[i])
            res=!res;
    return res;
}

int main()
{
    vector<bool> x(4);
    x[0]=true; x[1]=false; x[2]=false; x[3]=true;
    if (es_par1(x))
        cout<<"par"<<endl;
    else
        cout<<"impar"<<endl;
    if (es_par2(x))
        cout<<"par"<<endl;
    else
        cout<<"impar"<<endl;
    vector<bool> y(4);
    y[0]=true; y[1]=false; y[2]=true; y[3]=true;
    if (es_par1(y))
        cout<<"par"<<endl;
    else
        cout<<"impar"<<endl;
    if (es_par2(y))
        cout<<"par"<<endl;
    else
        cout<<"impar"<<endl;
}

```