cin_cout.cpp

```
#include <iostream>

using namespace std;

int main()
{
    int edad;
    string nombre;

    cout << "¿Como te llamas? ";
    //cin >> nombre;
    getline(cin,nombre);

    cout << "Cual es tu edad? ";
    cin >> edad;

    cout << "Hola " << nombre <<" Tienes " << edad << " anios!" << endl;
    return 0;
}
```

```
// Ficheros. Leer de ficheros.
// Ojo Esta version no coge bien las cadenas con espacios
// Ver version fich_leer2.cpp para cadenas con espacios
// ifstream se utiliza para LEER ficheros

#include <fstream>
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;

int main()
{
    ifstream fichInClients("clientes.txt");

    // Si no se puede abrir el fichero para lectura
    if (!fichInClients)
    {
        // Muestra mensaje de error y termina
        cerr << "El fichero clientes.txt no puede abrirse para lectura" <<
endl;
        exit(1);
    }
    else //si no ha habido problemas en la apertura del fichero clientes.dat
    {
        while (fichInClients) //mientras quede algo que leer en fichero
        {
            // lo lee del fichero y lo visualiza por pantalla
            string strInput;
            fichInClients >> strInput;
            cout << strInput << endl;
        }

    }
    return 0;
}
```

```
// Ficheros. Leer de ficheros.
// Coge bien las cadenas con espacios con getline

#include <fstream>
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;

int main()
{
    ifstream fichInClients("clientes.txt");

    // Si no se puede abrir el fichero para lectura
    if (!fichInClients)
    {
        // Muestra mensaje de error y termina
        cerr << "El fichero clientes.txt no puede abrirse para lectura" <<
endl;
        exit(1);
    }
    else //si no ha habido problemas en la apertura del fichero clientes.dat
    {
        while (fichInClients) //mientras quede algo que leer en fichero
        {
            // lo lee del fichero y lo visualiza por pantalla
            string strInput;
            getline(fichInClients,strInput); //lee cadenas con espacios en
blanco
            cout << strInput << endl;
        }
    }
    return 0;
}
```

fich_escribir1.cpp

```
// Ilustra como escribir en un fichero
// abrir fichero prueba.txt para observaer resultado

#include <fstream>
#include <iostream>
#include <stdlib.h>

using namespace std;

int main()
{
    ofstream miFich("prueba.txt");

    if(!miFich.good())
    {
        cout << "No puedo abrir fichero prueba.txt";
        exit(1);
    }
    miFich << "Holaaaa";
    miFich << " soy yo";

    miFich.close();
    return 0;
}
```

```
// Ilustra el funcionamiento de los Ficheros.
// Escribe en ficheros. Cierra fichero y añade nuevo alumno

#include <fstream>
#include <iostream>
#include <stdlib.h>

using namespace std;

int main()
{
    ofstream fichClientes("clientes.dat"); // Crearemos un fichero para
ESCRITURA llamado clientes.dat

    if (!fichClientes) // Si no podemos abrir el fichero
    {
        cerr << "ERROR: clientes.dat NO PUEDE SER ABIERTO PARA ESCRITURA!" <<
endl;
        exit(1);
    }
    else //si no ha habido problemas en la apertura del fichero clientes.dat
    {
        // Escribiremos estas dos lineas en el fichero
        fichClientes << "Juan Lopez\n" << "12345678\n" <<"2500" << endl;
        fichClientes << "Jose Garcia\n"<< "13345666\n" <<"6000" << endl;
        cout << "Info clientes grabada en fichero con exito\n";
    }
    fichClientes.close(); //cierro el fichero

    //.....
    //.....
    //Ahora le vuelvo a abrir para meter mas info añadiendo a lo que había
    fichClientes.open("clientes.dat",ios::app);  //abro en modo append ios::app
porque es para añadir
    if (!fichClientes.good()) // otra forma de comprobar que se ha abierto ok
    {
        cerr << "ERROR: clientes.dat NO PUEDE SER ABIERTO PARA ESCRITURA!" <<
endl;
        exit(1);
    }
    else //si no ha habido problemas en la apertura del fichero clientes.dat
    {
        // Escribimos una nueva linea en el fichero
        fichClientes << "Antonio Fernandez\n"<< "17745666\n" <<"3000" << endl;
        cout << "Ultima linea aniadido en fichero con exito\n";
    }
    return 0;


}
```

```
// Cuando dividamos por cero se producirá una excepción

#include <iostream>

using namespace std;

int division(int a, int b)
{
    return a/b;
}

int main()
{
    int a, b;
    cout << "cociente: ";
    cin >> a;
    cout << "divisor: ";
    cin >> b;

    cout << division(a, b);

    return 0;
}
```

```
#include <iostream>
#include <exception>

using namespace std;

int division(int a, int b)
{
    if (b==0) throw "Division por cero!!";
    return a/b;
}

int main()
{
    int a, b;
    cout << "cociente: ";
    cin >> a;
    cout << "divisor: ";
    cin >> b;

    try
    {
        cout << division(a, b);
    }
    catch (char const* strExcepcion)
    {
        cout << "ERROR: " << strExcepcion;
    }
    cout << "\nfin programa.";

    return 0;
}
```

```
// Ilustra trabajo con arrays.
// Incrementa en una unidad cada elemento y visualiza resultado

#include <iostream>
#include <array>

using namespace std;

int main()
{
    array<int,5> miArray{0, 1, 2, 3, 4};

    // incremetamos en una unidad cada elemento
    for (int i=0; i<miArray.size(); ++i) miArray[i]++;

    //visualizacion
    for (int i=0; i<miArray.size(); ++i)  cout << miArray[i] << " " ;
    cout << endl;
}
```

```
// Ilustra trabajo con arrays.
// Incrementa en una unidad cada elemento y visualiza resultado
// trabaja con bucles a todo el rango

#include <iostream>

using namespace std;

int main()
{

  int miArray[]={0 , 1, 2, 3, 4};

  for (int &elem : miArray) elem++; //&elem porque vamos a modificarlos
(elem++)
  //visualizacion
  for (int elem : miArray)    cout << elem << " " ;
  cout << endl;
}
```

clienteBanco1.cpp

```
//Crea una sencilla clase para cliente de un banco
//con variables miembro, Constructor y metodo visualizacion

#include <iostream>
#include <string.h>

using namespace std;

class ClienteBanco
{

    char nombre[25];
    int DNI;
    double saldo;
public:
    // Constructor para inicializar la informacion del cliente
    ClienteBanco(const char strNombre[], int numDNI, double unSaldo=0)
    {
        strncpy(nombre, strNombre, 25);
        DNI = numDNI;
        saldo = unSaldo;
    }

    // Visualiza la información del cliente
    void Print()
    {
        cout << "Nombre: " << nombre << "  DNI: " <<
             DNI << "  Saldo (euros): " << saldo << endl;
    }
};

int main()
{
    // Declara dos Empleados
    ClienteBanco cliente1("Juan Lopez", 12345678, 2500);
    ClienteBanco cliente2("Jose Garcia", 13345666);

    // Imprime la información de Empleado
    cliente1.Print();
    cliente2.Print();

    return 0;
}
```

```
//Crea una sencilla clase para cliente de un banco
//Incluye sobrecarga operador << para la visualizacion de datos

#include <iostream>
#include <string.h>

using namespace std;

class ClienteBanco
{

    char nombre[25];
    int DNI;
    double saldo;
public:
    ClienteBanco(const char strNombre[], int numDNI, double unSaldo=0)
    {
        strncpy(nombre, strNombre, 25);
        DNI = numDNI;
        saldo = unSaldo;
    }
    // Creamos una funcion amiga para imprimir toda la información del cliente
    friend ostream& operator<< (ostream &out, const ClienteBanco &cliente);
};


ostream& operator<< (ostream &out, const ClienteBanco &cliente)
{
    // Puesto que operator<< es una funcion amiga de la clase ClienteBanco
podemos acceder a sus miembros directamente.
    out << "Nombre: " << cliente.nombre << ".\nDNI: " <<  cliente.DNI <<
".\nSaldo: " << cliente.saldo << " euros\n";
    return out;
}


int main()
{
    // Declara dos clientes
    ClienteBanco cliente1("Juan Lopez", 12345678, 2500);
    ClienteBanco cliente2("Jose Garcia", 13345666);

    // Imprime simplemente con el operador << toda la información de los
clientes;
    cout << cliente1 <<endl;
    cout << cliente2 <<endl;

    return 0;
}
```

```
//Incluye operador + sobrecargado para sumar saldo a un cliente

#include <iostream>
#include <string.h>

using namespace std;

class ClienteBanco
{
    string nombre;
    int DNI;
    double saldo;
public:
    ClienteBanco(const string strNombre="\0", int numDNI=0, double unSaldo=0)
    {
        nombre=strNombre;
        DNI = numDNI;
        saldo = unSaldo;
    }
    friend ClienteBanco operator+ (const ClienteBanco &cliente, const double
&variaSaldo);
    friend istream& operator>> (istream &in, ClienteBanco &cliente);
    friend ostream& operator<< (ostream &out, const ClienteBanco &cliente);
};

ClienteBanco operator+ (const ClienteBanco &cliente, const double &variaSaldo)
{
    ClienteBanco clienteConNvoSaldo(cliente.nombre, cliente.DNI,
cliente.saldo+variaSaldo);
    return clienteConNvoSaldo;
}

istream& operator>> (istream &in, ClienteBanco &cliente)
{
    cout << "Nombre y apellidos: ";
    getline(in, cliente.nombre);  //Ya vale para nombre y apellidos
    cout << "DNI: ";
    in >> cliente.DNI;
    cout << "Saldo: ";
    in >> cliente.saldo;
    return in;
}

ostream& operator<< (ostream &out, const ClienteBanco &cliente)
{
    out << "Nombre: " << cliente.nombre << ".\nDNI: " <<  cliente.DNI <<
".\nSaldo: " << cliente.saldo << " euros\n";
    return out;
}


int main()
```

```
{
    // Declara dos clientes
    ClienteBanco cliente1("Juan Lopez", 12345678, 2500);
    ClienteBanco cliente2("Jose Garcia", 13345666);
    ClienteBanco cliente3;

    cout << "intro datos cliente 3:\n";
    cin >> cliente3;

    cliente3 = cliente3 + 2000; //aniadimos 2000 al saldo de cliente2
    cout << "Hemos aniadido 2000 al cliente 3\n";
    // Imprime la información de los clientes;
    cout << cliente1 <<endl;
    cout << cliente2 <<endl;
    cout << cliente3 <<endl;


    return 0;
}

// sigue access
```

```
//Ya se pueden meter nombres y apellidos con >> sobrecargado
//hemos cambiado el char nombre[25] por un string para usar getline

#include <iostream>
#include <string.h>

using namespace std;

class ClienteBanco
{
    string nombre;
    int DNI;
    double saldo;
public:
    ClienteBanco(const string strNombre="\0", int numDNI=0, double unSaldo=0)
    {
        nombre=strNombre;
        DNI = numDNI;
        saldo = unSaldo;
    }
    friend istream& operator>> (istream &in, ClienteBanco &cliente);
    friend ostream& operator<< (ostream &out, const ClienteBanco &cliente);
};

istream& operator>> (istream &in, ClienteBanco &cliente)
{
    cout << "Nombre y apellidos: ";
    getline(in, cliente.nombre);  //Ya vale para nombre y apellidos
    cout << "DNI: ";
    in >> cliente.DNI;
    cout << "Saldo: ";
    in >> cliente.saldo;

    return in;
}

ostream& operator<< (ostream &out, const ClienteBanco &cliente)
{
    out << "Nombre: " << cliente.nombre << ".\nDNI: " <<  cliente.DNI <<
".\nSaldo: " << cliente.saldo << " euros\n";
    return out;
}


int main()
{
    // Declara tres clientes
    ClienteBanco cliente1("Juan Lopez", 12345678, 2500);
    ClienteBanco cliente2("Jose Garcia", 13345666);
    ClienteBanco cliente3;
    // Pide solo los datos del tercero (los otros inicializados ya)
    cout << "Intro datos cliente 3:\n";
```

```
    cin >> cliente3;

    // Imprime la información de los clientes;
    cout << cliente1 <<endl;
    cout << cliente2 <<endl;
    cout << cliente3 <<endl;

    return 0;
}
```

```
// Incluye excepciones throw-catch
// Si se intenta sacar más dinero del que hay se lanza excepcion
// que se captura para visualizar un mesaje

#include <iostream>
#include <string.h>
#include <exception>

using namespace std;

class ClienteBanco
{
    string nombre;
    int DNI;
    double saldo;
public:
    ClienteBanco(const string strNombre="\0", int numDNI=0, int unSaldo=0)
    {
        nombre=strNombre;
        DNI = numDNI;
        saldo = unSaldo;
    }

    void operator+= (const double &variaSaldo)
    {
        saldo+=variaSaldo;
    }
    void operator-= (const double &variaSaldo)
    {
        if(variaSaldo > saldo)  //no hay saldo suficiente
        {
            throw -1;
        }
        else saldo-=variaSaldo;
    }
    friend istream& operator>> (istream &in, ClienteBanco &cliente);
    friend ostream& operator<< (ostream &out, const ClienteBanco &cliente);
};

istream& operator>> (istream &in, ClienteBanco &cliente)
{
    cout << "Nombre y apellidos: ";
    getline(in, cliente.nombre);  //Ya vale para nombre y apellidos
    cout << "DNI: ";
    in >> cliente.DNI;
    cout << "Saldo: ";
    in >> cliente.saldo;

    return in;
}

ostream& operator<< (ostream &out, const ClienteBanco &cliente)
```

```
{
    out << "Nombre: " << cliente.nombre << ".\nDNI: " <<  cliente.DNI <<
".\nSaldo: " << cliente.saldo << " euros\n";
    return out;
}


int main()
{
    // Declara dos clientes
    ClienteBanco cliente1("Juan Lopez", 12345678, 2500);
    ClienteBanco cliente2("Jose Garcia", 13345666);
    ClienteBanco cliente3("Juan Jose Rodriguez", 17895678, 500);

    cliente2 += 2000; //aniadimos 2000 al saldo de cliente2
    // Si intentamos sacar mas dinero del que hay lanza mensaje de error
    try
    {
        cliente1 -=3000;
    }
    catch(int e)
    {
        cout << "ERROR " << e << ": No hay dinero suficiente en cuenta. "
<<endl;
    }

    // Imprime la información de los clientes;
    cout << cliente1 <<endl;
    cout << cliente2 <<endl;
    cout << cliente3 <<endl;

    return 0;
}
```

```
//Incluye los operadores +=  y  -= para incrementar y decrementar saldo

#include <iostream>
#include <string.h>

using namespace std;

class ClienteBanco
{
    string nombre;
    int DNI;
    double saldo;
public:
    ClienteBanco(const string strNombre="\0", int numDNI=0, double unSaldo=0)
    {
        nombre=strNombre;
        DNI = numDNI;
        saldo = unSaldo;
    }
    void operator+= (const double &variaSaldo)
    {
        saldo+=variaSaldo;
    }
    void operator-= (const double &variaSaldo)
    {
        saldo-=variaSaldo;
    }
    friend istream& operator>> (istream &in, ClienteBanco &cliente);
    friend ostream& operator<< (ostream &out, const ClienteBanco &cliente);
};

istream& operator>> (istream &in, ClienteBanco &cliente)
{
    cout << "Nombre y apellidos: ";
    getline(in, cliente.nombre);  //Ya vale para nombre y apellidos
    cout << "DNI: ";
    in >> cliente.DNI;
    cout << "Saldo: ";
    in >> cliente.saldo;

    return in;
}

ostream& operator<< (ostream &out, const ClienteBanco &cliente)
{
    out << "Nombre: " << cliente.nombre << ".\nDNI: " <<  cliente.DNI <<
".\nSaldo: " << cliente.saldo << " euros\n";
    return out;
}


int main()
```

```
{
    // Declara dos clientes
    ClienteBanco cliente1("Juan Lopez", 12345678, 2500);
    ClienteBanco cliente2("Jose Garcia", 13345666);

    cout << "Vamos a quitar 500 al saldo del primer cliente y sumar 2000 al
segundo\n";
    cliente1 -=500;     //De esta forma tan sencilla quitamos 500 del saldo de
cliente1
    cliente2 += 2000;   //aniadimos 2000 al saldo de cliente2

    // Imprime la información de los clientes;
    cout << cliente1 <<endl;
    cout << cliente2 <<endl;
    return 0;
}
```

```
//incluye sobrecarga de operadores tanto de salida como de entrada

#include <iostream>
#include <string.h>

using namespace std;

class ClienteBanco
{

    char nombre[25];
    int DNI;
    double saldo;
public:
    ClienteBanco(const char strNombre[]="\0", int numDNI=0, double unSaldo=0)
    {
        strncpy(nombre, strNombre, 25);
        DNI = numDNI;
        saldo = unSaldo;
    }
    friend istream& operator>> (istream &in, ClienteBanco &cliente);
//entrada datos
    friend ostream& operator<< (ostream &out, const ClienteBanco &cliente);
//salida
};


istream& operator>> (istream &in, ClienteBanco &cliente)
{
    cout << "Nombre: ";
    in >> cliente.nombre;  //Ojo! solo vale para un nombre sencillo, sin
espacios en blanco
    cout << "DNI: ";
    in >> cliente.DNI;
    cout << "Saldo: ";
    in >> cliente.saldo;

    return in;
}

ostream& operator<< (ostream &out, const ClienteBanco &cliente)
{

    out << "Nombre: " << cliente.nombre << ".\nDNI: " <<  cliente.DNI <<
".\nSaldo: " << cliente.saldo << " euros\n";
    return out;
}


int main()
{
    // Declara tres clientes
```

```
    ClienteBanco cliente1("Juan Lopez", 12345678, 2500);
    ClienteBanco cliente2("Jose Garcia", 13345666);
    ClienteBanco cliente3;
    // Pide solo los datos del tercero (los otros inicializados ya)
    cout << "Intro datos cliente 3:\n";
    cin >> cliente3;

    // Imprime la información de los clientes;
    cout << cliente1 <<endl;
    cout << cliente2 <<endl;
    cout << cliente3 <<endl;

    return 0;
}
```

```
// Ilustra la insercion en deques (STL) con
// push_back() y push_front()

#include <iostream>
#include <deque>
using namespace std;
int main()
{
    deque<int> miLista;
    for (int i=0; i < 10; i++) miLista.push_back(i);  // inserta al final de la
lista
    cout << "Lista tras insercion al final:" << endl;
    for( auto elem : miLista) cout << elem << " " ;
    cout << endl;
    for (int i=0; i < 10; i++) miLista.push_front(i); // inserta al principio
    cout << "Lista tras insercion al principio:" << endl;
    for( auto elem : miLista) cout << elem << " " ;
    cout << endl;
}
```

```
// Ejemplo for_each. Todavía sin for_each. Ver forEach1.cpp

#include <iostream>     // cout
#include <vector>       // vector

using namespace std;

int main ()
{
    vector<int> miVector{10,20,30}; //vector inicializado con tres elementos
    //Luego podemos cargar mas elementos con push_back
    miVector.push_back(40);
    miVector.push_back(50);

    cout << "miVector contiene:";
    for(auto i=0; i<miVector.size(); i++) cout << " " << miVector[i];

    return 0;
}
```

```
// Ejemplo for_each.

#include <iostream>     // cout
#include <vector>       // vector
#include <algorithm>    // for_each

using namespace std;

void imprimeNum (int N)
{
    cout << " " << N;
}

int main ()
{
    vector<int> miVector{10,20,30}; //vector inicializado con tres elementos
    //Luego podemos cargar mas elementos con push_back
    miVector.push_back(40);
    miVector.push_back(50);

    cout << "miVector contiene:";

    for_each (miVector.begin(), miVector.end(), imprimeNum);
    cout << endl;

    return 0;
}
```

```
// Ejemplo for_each con functor
//un 'functor' es un objeto que actúa como una función.
//Básicamente, una clase que define operator().

#include <iostream>     // cout
#include <vector>       // vector
#include <algorithm>    // for_each

using namespace std;

class MiFunctor
{
   public:
     void operator()(int N) { cout << " " << N;}
};


int main ()
{
    MiFunctor impresor;
    vector<int> miVector{10,20,30}; //vector inicializado con tres elementos
    //Luego podemos cargar mas elementos con push_back
    miVector.push_back(40);
    miVector.push_back(50);

    cout << "miVector contiene:";

    for_each (miVector.begin(), miVector.end(), impresor);
    cout << endl;

    return 0;
}
```

```
// Ejemplo for_each con lambda (funcion anonima)

#include <iostream>     // cout
#include <vector>       // vector
#include <algorithm>    // for_each

using namespace std;

int main ()
{
    vector<int> miVector{10,20,30}; //vector inicializado con tres elementos
    //Luego podemos cargar mas elementos con push_back
    miVector.push_back(40);
    miVector.push_back(50);

    cout << "miVector contiene:";

    for_each (miVector.begin(), miVector.end(), [](int N){ cout << " " << N;}
);
    cout << endl;

    return 0;
}
```

forEachImpVector.cpp

```
// Ejemplo for_each con lambda (funcion anonima)

#include <iostream>     // cout
#include <vector>       // vector
#include <algorithm>    // for_each

using namespace std;

int main ()
{
    vector<int> miVector{10,20,30}; //vector inicializado con tres elementos
    //Luego podemos cargar mas elementos con push_back
    miVector.push_back(40);
    miVector.push_back(50);

    cout << "miVector contiene:";

    for_each (miVector.begin(), miVector.end(), [](int N){ cout << " " << N;}
);
    cout << endl;

    return 0;
}
```

forEachSumaElem.cpp

```
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

class Suma
{
public:
    int total;
    Suma(): total{0} { }
    void operator()(int n) { total += n; }

};

int main()
{
    vector<int> miVector{1, 2, 3, 4, 5};

    cout << "antes:";
    for (auto const &elem : miVector) cout << " " << elem;
    cout << endl;

    //incrementa en una unidad cada elemento con una lambda
    for_each(miVector.begin(), miVector.end(), [](int &elem){ elem++; });

    //visualiza el vector con los incrementos
    cout << "despues: ";
    for (auto const &elem : miVector)  cout << " " << elem;
    cout << endl;

    // llama a Suma::operator() para cada número
    Suma s=for_each(miVector.begin(), miVector.end(), Suma());

    cout << "suma: " << s.total << endl;

}
```

forEachMultiplicator.cpp

```
// Emplea un objeto funcion (functor) para multiplicar
// los elementos de un vector


#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

// El functor multiplica un elemento por un factor
template <class Tipo>
class MultiplicaXvalor
{
private:
    Tipo factor;   // El valor por el que multiplicaremos
public:
    // Constructor inicializar el valor por el que multiplicaremos
    MultiplicaXvalor ( const Tipo& valor )
    {
        factor = valor;
    }
    void operator ( ) ( Tipo& elem )
    {
        elem *= factor;
    }
};

int main( )
{
    vector <double> miVector;
    for ( int i = -4 ; i <= 2 ; i++ ) miVector.push_back(  i );

    cout << "vector original miVector = ( " ;
    for(auto elem : miVector) cout << elem << " ";
    cout << ")." << endl;

    // Empleamos for_each para multiplicar cada elemento por un factor
    for_each ( miVector.begin ( ), miVector.end ( ), MultiplicaXvalor<double>(
-2.1 ) );

    cout << "miVector multiplicado por -2.1 = ( ";
    for(auto elem : miVector) cout << elem << " ";
    cout << ")." << endl;

    // La funcion emplea templates y puede usarse con distintos tipos
    for_each (miVector.begin ( ), miVector.end ( ), MultiplicaXvalor<double>
(5.2 ) );

     cout << "miVector multiplicado por 5.2 = ( ";
    for(auto elem : miVector) cout << elem << " ";
    cout << ")." << endl;
```

```
}
```

```
//Calcula la media de los elementos de un vector (cualquier tipo, template)
//Emplea for_each y functor que devuelve un double con la media

#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;


// El functor para calcular promedio
template <class Tipo>
class Media
{
private:
    long numElem;       // numero de elementos
    Tipo suma;          // suma de los elementos
public:
    // Constructor inicializa numElem y suma a cero
    Media ( ) : numElem ( 0 ), suma ( 0 ) {}
    void operator ( ) ( Tipo elem )
    {
        numElem++;      // Incrementa el contador de elementos
        suma += elem;   // Añade elemento a la suma parcial
    }
    operator double ( ) //para que devuelva un double con suma/numElem
    {
        return  static_cast <double> (suma) / numElem;
    }
};


int main( )
{
    vector <double> miVector{2.2, 2.0, 3.0, 4.0, 5.0};

    cout << "vector original miVector = ( " ;
    for(auto elem : miVector) cout << elem << " ";
    cout << ")." << endl;

    double media = for_each ( miVector.begin ( ), miVector.end ( ),
Media<double> ( ) );
    cout << "El promedio de los elementos en el vector es  " << media << "." <<
endl;
}
```

```
#include <iostream>

using namespace std;

int main()
{

    int myarray[]= {10,20,30};

    for (int i=0; i<3; ++i)
        myarray[i]++; //sumamos una unidad a cada elemento

    //visualiza los tres elementos
    for (int i=0; i<3; ++i)
        cout << myarray[i] << '\n';
}
```

```
//recorre vector empleando for range

#include <iostream>

using namespace std;

int main()
{

    int myarray[]= {10,20,30};

    for (int i=0; i<3; ++i)
        myarray[i]++; //sumamos una unidad a cada elemento

    //visualiza los tres elementos con un for RANGE
    for (int elem : myarray)
        cout << elem << '\n';
}
```

```
//recorre vector
//emplea for range y variables auto.

#include <iostream>

using namespace std;

int main()
{

    int myarray[]= {10,20,30};

    //empleamos auto, el compilador asigna el tipo a i
    for (auto i=0; i<3; ++i)
        myarray[i]++; //sumamos una unidad a cada elemento

    //visualiza los tres elementos con un for RANGE y variable auto
    for (auto elem : myarray)
        cout << elem << '\n';
}
```

```
//recorre vector
//emplea for range & y variables auto.

#include <iostream>

using namespace std;

int main()
{

    int myarray[]= {10,20,30};

    //empleamos auto, el compilador asigna el tipo a i
    for (auto& elem : myarray)
        elem++; //aquí tenemos que emplear & porque modificamos los elementos

    //visualiza los tres elementos con un for RANGE y variable auto
    for (auto elem : myarray)
        cout << elem << '\n';
}
```

```
// Ilustra la busqueda en listas enlazadas con find
// y la insercion con insert (STL)


#include <iostream>
#include <list>
#include <algorithm>

using namespace std;

int main()
{
    list<int> miLista;
    // Creamos una lista con 20 elementos del 0 al 20
    for (int n=0; n < 20; n++) miLista.push_back(n);
    // Declaramos un iterador y llamamos a find para que busque el elemento que
vale 17
    auto it = find(miLista.begin(), miLista.end(), 17);
    // it apuntará ahora al elemento que vale 17
    miLista.insert(it, 20); // inserta el valor 20 delante de donde apunta it,
delante del 17
    for( auto elem : miLista) cout << elem << " " ;
    cout << endl;
}
```

```
// Ilustra el empleo de lambdas y functors con forEach

#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

class Suma
{
public:
    int total;
    Suma(): total{0} { }
    void operator()(int n) { total += n; }

};

int main()
{
    vector<int> miVector{1, 2, 3, 4, 5};

    cout << "antes:";
    for (auto const &elem : miVector) cout << " " << elem;
    cout << endl;

    //incrementa en una unidad cada elemento con una lambda
    for_each(miVector.begin(), miVector.end(), [](int &elem){ elem++; });

    //visualiza el vector con los incrementos
    cout << "despues: ";
    for (auto const &elem : miVector)  cout << " " << elem;
    cout << endl;

    // llama a Suma::operator() para cada número
    Suma s=for_each(miVector.begin(), miVector.end(), Suma());

    cout << "suma: " << s.total << endl;

}
```

```
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

class Suma
{
public:
    int total;
    Suma(): total{0} { }
    void operator()(int n) { total += n; }

};

int main()
{
    vector<int> miVector{1, 2, 3, 4, 5};

    cout << "antes:";
    for (auto const &elem : miVector) cout << " " << elem;
    cout << endl;

    //incrementa en una unidad cada elemento con una lambda
    for_each(miVector.begin(), miVector.end(), [](int &elem){ elem++; });

    //visualiza el vector con los incrementos
    cout << "despues: ";
    for (auto const &elem : miVector)  cout << " " << elem;
    cout << endl;

    // llama a Suma::operator() para cada número
    Suma s=for_each(miVector.begin(), miVector.end(), Suma());

    cout << "suma: " << s.total << endl;

}
```

```
//Emplea templated functor para multiplicar un vector por un escalar

#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

// El functor multiplica cada elemento del vector por el factor
template <class Tipo>
class MultiplicaXvalor
{
private:
    Tipo factor;   // El valor por el que multiplicaremos
public:
    // Constructor inicializar el valor por el que multiplicaremos
    MultiplicaXvalor ( const Tipo& valor )
    {
        factor = valor;
    }
    void operator ( ) ( Tipo& elem )
    {
        elem *= factor;
    }
};


int main( )
{

    vector <double> miVectorD{1.0, 2.0, 3.0, 4.0, 5.0};
    vector <int> miVectorI{1, 2, 3, 4, 5};


    cout << "vector double original miVector = ( " ;
    for(auto elem : miVectorD) cout << elem << " ";
    cout << ")." << endl;

    // Empleamos for_each para multiplicar cada elemento por un factor
    for_each ( miVectorD.begin ( ), miVectorD.end ( ),
MultiplicaXvalor<double>( -1.01 ) );

    cout << "miVector multiplicado por -1.01 = ( ";
    for(auto elem : miVectorD) cout << elem << " ";
    cout << ")." << endl;

    // La funcion emplea templates y puede usarse con distintos tipos. Ahora
con int
    for_each (miVectorI.begin ( ), miVectorI.end ( ), MultiplicaXvalor<int> (2)
);

     cout << "miVector multiplicado por 2 = ( ";
```

```
    for(auto elem : miVectorI) cout << elem << " ";
    cout << ")." << endl;


}
```

```
//Calcula la media de los elementos de un vector (cualquier tipo, template)
//Emplea for_each y functor que devuelve un double con la media

#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;


// El functor para calcular promedio
template <class Tipo>
class Media
{
private:
    long numElem;       // numero de elementos
    Tipo suma;          // suma de los elementos
public:
    // Constructor inicializa numElem y suma a cero
    Media ( ) : numElem ( 0 ), suma ( 0 ) {}
    void operator ( ) ( Tipo elem )
    {
        numElem++;      // Incrementa el contador de elementos
        suma += elem;   // Añade elemento a la suma parcial
    }
    operator double ( ) //para que devuelva un double con suma/numElem
    {
        return  static_cast <double> (suma) / numElem;
    }
};


int main( )
{
    vector <double> miVector{2.2, 2.0, 3.0, 4.0, 5.0};

    cout << "vector original miVector = ( " ;
    for(auto elem : miVector) cout << elem << " ";
    cout << ")." << endl;

    double media = for_each ( miVector.begin ( ), miVector.end ( ),
Media<double> ( ) );
    cout << "El promedio de los elementos en el vector es  " << media << "." <<
endl;
}
```

```
// Ilustra como recorrer un vector con iteradores STL

#include <iostream>
#include <vector>

int main()
{
    using namespace std;

    vector<int> vect;
    for (int n=0; n < 6; n++)  vect.push_back(n);

    // vector<int>::const_iterator it; // asi es como se declara un iterador
normalmente

    auto it = vect.begin();     // lo mas facil es usar una variable auto y la
hacemos apuntar al inicio
    while (it != vect.end())    // mientras no alcance el ultimo elemento
    {
        cout << *it << " ";     // visualiza el contenido del elemento apuntado
        it++;                   // itera al elemento siguiente
    }

    cout << endl;
    //otra forma
    for(it=vect.begin(); it!=vect.end(); it++) cout << *it << " ";
    cout << endl;
    //la forma mas facil es usando un bucle a todo el rango
    for (auto elem : vect) cout << elem <<  " ";
    cout << endl;

}
```

```
// Ilustra funcionamiento de lambda

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

int main()
{
    // Creamos un vector v con los num del 0 al 9
    vector<int> v;
    for (int i = 0; i < 10; ++i) v.push_back(i);

    // En el for_each incluimos una lambda que visualiza que numero es par
    // que numero es impar y el numero de impares que hay en el vector

    int contImpares = 0;
    for_each(v.begin(), v.end(), [&contImpares] (int num)
    {
        cout << num;
        if (num % 2 == 0)
        {
            cout << " es par " << endl;
            ++contImpares;
        }
        else
        {
            cout << " es impar " << endl;
        }
    });

    // Print the count of even numbers to the console.
    cout << "Hay " << contImpares
         << " numeros impares en el vector." << endl;
}
```

lambdaHilo.cpp

```
//Arrancar un hilo con una lambda. Cuando el codigo asociado
//al hilo es muy pequeño no hace falta crear una funcion.
//Se puede emplear una expresion lambda que defina lo que el
//hilo debe ejecutar. Reescribiremos el codigo del ejemplo anterior
//empleando una lambda:


#include <thread>
#include <iostream>

using namespace std;

int main()
{
    thread hilo1([](){cout << "Hola mundo " << endl;}); //crea y arranca un
nuevo hilo llamado hilo1
    hilo1.join();       //main espera a que termine hilo1

    return 0;
}
```

```
// Ilustra lambda en for_each donde incrementa los elementos de un vector

#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int main()
{
    vector<int> miVector{1, 2, 3, 4, 5};

    cout << "antes:";
    for (auto const &elem : miVector) cout << " " << elem;
    cout << endl;

    //incrementa en una unidad cada elemento con una lambda
    for_each(miVector.begin(), miVector.end(), [](int &elem){ elem++; });

    //visualiza el vector con los incrementos
    cout << "despues: ";
    for (auto const &elem : miVector)  cout << " " << elem;
    cout << endl;

}
```

```
// Ejemplo for_each con lambda (funcion anonima)

#include <iostream>     // cout
#include <vector>       // vector
#include <algorithm>    // for_each

using namespace std;

int main ()
{
    vector<int> miVector{10,20,30}; //vector inicializado con tres elementos
    //Luego podemos cargar mas elementos con push_back
    miVector.push_back(40);
    miVector.push_back(50);

    cout << "miVector contiene:";

    for_each (miVector.begin(), miVector.end(), [](int N){ cout << " " << N;}
);
    cout << endl;

    return 0;
}
```

```
// Ilustra como localizar el elemento mayor o menor
// en un contenedor STL con max_element o min_element

#include <iostream>
#include <list>
#include <algorithm>

using namespace std;

int main()
{
    // Inicializamos la lista con unos valores
    list<int> miLista{ 3, 27, -4, 9, 12};

    auto minimo = min_element(miLista.begin(), miLista.end());
    auto maximo = max_element(miLista.begin(), miLista.end());
    cout << "El elemento menor es: " << *minimo << endl;
    cout << "El elemento mayor es: " << *maximo << endl;

    cout << endl;
}
```

```
//Ilustra el problema de las condiciones de carrera
//al modificar varios hilos a una variable compartida
//El resultado que proporcionará el programa será generalmente erroneo
//porque no se garantiza la exclusion mutua al acceder a la variable
//compartida contador


#include <iostream>
#include <vector>
#include <thread>
#include <mutex>
#define NUM_HILOS 8         //numero de hilos que se crean en el programa
#define NUM_ITER 10000      //numero de iteraciones que realizará cada hilo

using namespace std;

struct Contador
{
    int valor;
    Contador() : valor(0) {}

    void incrementa()
    {
        ++valor;
    }
};

void itera(Contador* c)
{
    for(auto j = 0; j < NUM_ITER; j++) c->incrementa();
}

int main()
{
    Contador cont;
    vector<thread> misHilos;                        //vector en el que
meteremos varios hilos
    for(auto i = 0; i < NUM_HILOS; ++i)             //metemos NUM_HILOS dentro
del vector
        misHilos.push_back(thread(itera, &cont));   //cada hilo incrementara el
cont NUM_ITER
    //Indicamos a main que espere a que terminen todos los hilos
    for(auto& hilo : misHilos) hilo.join();

    cout << "El valor del contador incrementado simultaneamente por los " <<
NUM_HILOS << " hilos es " <<cont.valor << endl;
    cout << "Debe ser igual a " << NUM_HILOS*NUM_ITER;

    return 0;
}
```

```
//Ilustra el problema de las condiciones de carrera
//al modificar varios hilos a una variable compartida
//El resultado que proporcionará el programa será correcto
//porque ahora se garantiza la exclusion mutua al acceder a la variable
//compartida contador regulando el acceso por un mutex
//Ojo, la ejecución va a ser más LENTA porque estamos secuencializando
//el acceso


#include <iostream>
#include <vector>
#include <thread>
#include <mutex>
#define NUM_HILOS 8         //numero de hilos que se crean en el programa
#define NUM_ITER 10000      //numero de iteraciones que realizará cada hilo

using namespace std;

struct Contador
{
    int valor;
    mutex mtx;              //para garantizar EXCLUSION MUTUA
    Contador() : valor(0) {}

    void incrementa()
    {
        mtx.lock();         //cierra el paso a otro hilo
        ++valor;            //incrementa ahora sin problemas
        mtx.unlock();       //libera el acceso a valor
    }
};

void itera(Contador* c)
{
    for(auto j = 0; j < NUM_ITER; j++) c->incrementa();
}


int main()
{
    Contador cont;
    vector<thread> misHilos;                        //vector en el que
meteremos varios hilos
    for(auto i = 0; i < NUM_HILOS; ++i)             //metemos NUM_HILOS dentro
del vector
        misHilos.push_back(thread(itera, &cont));   //cada hilo incrementara el
cont NUM_ITER
    //Indicamos a main que espere a que terminen todos los hilos
    for(auto& hilo : misHilos) hilo.join();

    cout << "El valor del contador incrementado simultaneamente por los " <<
```

```
NUM_HILOS << " hilos es " <<cont.valor << endl;
    cout << "Debe ser igual a " << NUM_HILOS*NUM_ITER;

    return 0;
}
```

```
//Ilustra el problema de las condiciones de carrera
//al modificar varios hilos a una variable compartida
//El resultado que proporcionará el programa será correcto
//porque ahora se garantiza la exclusion mutua al acceder a la variable
//compartida contador regulando el acceso por un mutex
//Ojo, la ejecución va a ser más LENTA porque estamos secuencializando
//el acceso.
//contiene funcion lambda

#include <iostream>
#include <vector>
#include <thread>
#include <mutex>
#define NUM_HILOS 8         //numero de hilos que se crean en el programa
#define NUM_ITER 10000      //numero de iteraciones que realizará cada hilo

using namespace std;

struct Contador
{
    int valor;
    mutex mtx;              //para garantizar EXCLUSION MUTUA
    Contador() : valor(0) {}

    void incrementa()
    {
        mtx.lock();         //cierra el paso a otro hilo
        ++valor;            //incrementa ahora sin problemas
        mtx.unlock();       //libera el acceso a valor
    }
};

int main()
{
    Contador cont;
    vector<thread> misHilos;                        //vector en el que
meteremos varios hilos
    for(auto i = 0; i < NUM_HILOS; ++i)             //metemos NUM_HILOS dentro
del vector
        misHilos.push_back(thread([&cont](){for(auto j = 0; j < 10000;
j++)cont.incrementa();}));//pasamos codigo de cada hilo mediante una lambda
    //Indicamos a main que espere a que terminen todos los hilos
    for(auto& hilo : misHilos) hilo.join();


    cout << "El valor del contador incrementado simultaneamente por los " <<
NUM_HILOS << " hilos es " <<cont.valor << endl;
    cout << "Debe ser igual a " << NUM_HILOS*NUM_ITER;

    return 0;
}
```

```
// Productor Consumidor. Uso de las variables de condicion

#include <thread>
#include <mutex>
#include <chrono>
#include <iostream>
#include <condition_variable>
#define TAM_BUF 20    //tamaño del buffer limitado
using namespace std;

class BufferLimitado
{
    int* buffer;    //puntero al inicio del buffer
    int tamanio;    //tamaño del buffer
    int out;        //indice al siguiente elemento que saldrá del buffer
    int in;         //indice a la posición en la que entrará el sig elemento
    int cont;       //contador de numero de elementos en buffer
    mutex lock;     //para exclusion mutua
    condition_variable noEstaLleno; //variable de condicion para bloquear
productores si esta el buffer lleno
    condition_variable noEstaVacio; //variable de condicion para bloquear
consumidor si esta el buffer vacio
public:
    BufferLimitado(int tam) //constructor
    {
        tamanio= tam;
        out=in=cont=0;
        buffer = new int[tamanio];
    }

    ~BufferLimitado()   //destructor
    {
        delete[] buffer;
    }

    void mete(int valor)
    {
        unique_lock<mutex> l(lock);
        while(cont==tamanio) noEstaLleno.wait(l); //si esta lleno se bloquea
        buffer[in] = valor;
        in = (in + 1) % tamanio;
        cont++;
        noEstaVacio.notify_one();
    }

    int saca()
    {
        unique_lock<mutex> l(lock);
        while(cont==0) noEstaVacio.wait(l); //si esta lleno se bloquea
        int valor = buffer[out];
        out = (out + 1) % tamanio;
        cont--;
```

```
        noEstaLleno.notify_one();
        return valor;
    }
};

void consumidor(int id, BufferLimitado& buffer)
{
    for(int i = 0; i < 10; ++i)
    {
        int valor = buffer.saca();
        cout << "Consumidor " << id << " saca " << valor << endl;
        this_thread::sleep_for(chrono::milliseconds(300));
    }
}

void productor(int id, BufferLimitado& buffer)
{
    for(int i = 0; i < 15; ++i)
    {
        buffer.mete(i);
        cout << "Productor " << id << " mete " << i << endl;
        this_thread::sleep_for(chrono::milliseconds(100));
    }
}

int main()
{
    BufferLimitado buffer(TAM_BUF);
    //Lanzamos varios consumidores
    thread c0(consumidor, 0, ref(buffer)); //pasa el buffer por referencia para
evitar una copia
    thread c1(consumidor, 1, ref(buffer));
    thread c2(consumidor, 2, ref(buffer));
    //...y varios productores
    thread p0(productor , 0, ref(buffer));
    thread p1(productor , 1, ref(buffer));
    //esperamos a que acaben
    c0.join();
    c1.join();
    c2.join();
    p0.join();
    p1.join();

    return 0;
}
```

```
// ordena un vector con sort
// cambia orden con reverse

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
    vector<int> vect{7, -3, 6, 2, -5, 0, 4};

    // visualiza vector
    for(auto elem:vect) cout << elem << " ";
    cout << endl;

    // ordena
    sort(vect.begin(), vect.end()); // sort the list

    // visualiza vector ya ordenado
    for(auto elem:vect) cout << elem << " ";
    cout << endl;

    // da la vuelta al vector
    reverse(vect.begin(), vect.end()); // reverse the list

    // visualiza vector ya ordenado
    for(auto elem:vect) cout << elem << " ";
    cout << endl;
}
```

```
// Empleo de variables atomic a las que se accede 'atomicamente'
// es decir, sin poderse interrumpir en su escritura

#include <iostream>
#include <thread>
#include <atomic>

using namespace std;

atomic<int> total;

void incrementaTotal()
{
    for (int i = 0; i < 100000; i++) total++;
}

int main()
{
    total=0;
    thread hilo1(incrementaTotal), hilo2(incrementaTotal);
    hilo1.join();
    hilo2.join();
    cout << "total: " << total << "\n";
    return 0;
}
```

```
// incluye mutex para garantizar exclusion mutua en incremento
// de variable compartida con tres hilos.
// Ver programa threadsinMutex.cpp

#include <iostream>
#include <thread>
#include <mutex>

using namespace std;

mutex mu;
int total=0;

void incrementa()
{
    for (int i = 0; i < 100000; i++)
    {
        mu.lock();
        total++;
        mu.unlock();
    }
}

int main()
{
    thread hilo1(incrementa),hilo2(incrementa);
    hilo1.join();
    hilo2.join();
    cout << "total: " << total << "\n";
    return 0;
}
```

threadLambda.cpp

```
//Arrancar un hilo con una lambda. Cuando el codigo asociado
//al hilo es muy pequeño no hace falta crear una funcion.
//Se puede emplear una expresion lambda que defina lo que el
//hilo debe ejecutar. Reescribiremos el codigo del ejemplo anterior
//empleando una lambda:

#include <thread>
#include <iostream>

using namespace std;

int main()
{
    thread hilo1([](){cout << "Hola mundo " << endl;}); //crea y arranca un
nuevo hilo llamado hilo1
    hilo1.join();       //main espera a que termine hilo1

    return 0;
}
```

```
// Ilustra join(). Que hace que main espere la terminacion de ese hilo

#include <thread>
#include <iostream>

using namespace std;
void hola()
{
    cout << "Hola mundo " << endl;
}

int main()
{
    thread hilo1(hola); //crea y arranca un nuevo hilo llamado hilo1
    hilo1.join();       //main espera a que termine hilo1

    return 0;
}
```

```
// thread::hardware_concurrency() proporciona el numero de hilos
// que pueden ejecutarse concurrentemente

#include <iostream>
#include <thread>

using namespace std;
int main()
{
    cout << "Numero de hilos = "
              <<  thread::hardware_concurrency() << endl;
    return 0;
}
```

threaddetach.cpp

```
// Ilustra detach(). Que separa la ejecución del hilo de la ejec.del main

#include <thread>
#include <iostream>

using namespace std;
void hola()
{
    cout << "Hola mundo " << endl;
}

int main()
{
    thread hilo1(hola); //crea y arranca un nuevo hilo llamado hilo1
    hilo1.detach();     //Separa la ejecución del hilo de la ejec.del main
                        //permitiendo continuar la ejecución independientemente
    return 0;
}
```

threadsinMutex.cpp

```
// Ilustra los problemas que aparecen cuando
// dos hilos acceden concurrentemente a variable compartida
// Cada vez obtendremos un resultado distinto  pues no se garantiza la
exclusion mutua

#include <iostream>
#include <thread>

using namespace std;

int total=0;

void incrementaTotal()
{
    for (int i = 0; i < 100000; i++) total++;
}

int main()
{
    thread hilo1(incrementaTotal), hilo2(incrementaTotal);
    hilo1.join();
    hilo2.join();
    // el resultado debería ser 200000 tenemos dos hilos que increm en 100000
cada uno
    cout << "total: " << total << "\n";
    return 0;
}
```

threadpasoParam2.cpp

```
// paso de parametros a hilos

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

void imprime()
{
    cout << "Este es el primer hilo y solo imprime este mensaje en pantalla" <<
endl;
}

void multiplicax2(int n)
{
    int doble;
    doble= 2 * n;
    cout << "Este es el segundo hilo" << endl;
    cout << "El doble de "<< n << " es " << doble << endl;
}

int main()
{
    int a,b;
    thread hilo1(imprime);          // genera un nuevo hilo que llama a la
funcion imprime()
    thread hilo2(multiplicax2,7);   // genera un nuevo hilo que llama a la
funcion multiplicax2(7)

    cout << "Los tres hilos ejecutandose concurrentemente...\n";
    // sincronizamos los hilos para que main espere a que terminen los otros
    hilo1.join();                // espera a que finalice hilo1
    hilo2.join();                // espera a que finalice hilo2

    cout << "Ya han acabado hilo1 e hilo2\n";

    return 0;
}
```

threadpasoParam1.cpp

```
// Ilustra paso de un param a hilo

#include <iostream>
#include <thread>
#include <string>

using namespace std;

void imprime(string msg)
{
    cout << "En funcion hilo. ";
    cout << "El mensaje es: " << msg << endl;
}

int main()
{
    string mensaje = "Informatica Industrial";
    thread hilo(imprime, mensaje);
    cout << "Hilo main. El mensaje es: " << mensaje << endl;
    hilo.join();
    return 0;
}
```

```
// Incluye mutex para la impresion ordenada de mensajes

#include <iostream>
#include <thread>
#include <string>
#include <mutex>

using namespace std;

mutex mu;

//Incluye mutex para una impresion ordenada, sin mezclas
void impresionOrdenada(string msg, int id)
{
        mu.lock();
        cout << msg << ":" << id << endl;
        mu.unlock();
}
//funcion miFuncion imprime negativos
void miFuncion()
{
        for (int i = -100; i < 0; i++)
                impresionOrdenada("hilo funcion thread", i);
}

//funcion main imprime positivos
int main()
{
        thread t(miFuncion);
        for (int i = 100; i > 0; i--)
            impresionOrdenada("hilo funcion main", i);
        t.join();
        return 0;
}
```

vectorInsertBorrar.cpp

```
// Ilustra como trabajar con vector STL
// para insertar y borrar elementos


#include <vector>
#include <iostream>

using namespace std;

int main()
{
    int suma=0;
    vector<int> miVector; // declaramos un vector

    for (int n=0; n < 10; n++)  miVector.push_back(n); // insertamos del 0 al 9
por final

    // visualizamos el contenido
    cout << "Contenido inicial: " << endl;
    for( auto elem : miVector) cout << elem << " ";
    cout << endl;

    // borrado del 6º elemento con erase()
    miVector.erase (miVector.begin()+5);
    cout << "Despues de borrar elemento en la sexta posicion: " << '\n';
    for( auto elem : miVector) cout << elem << " ";
    cout << endl;

    //insercion de un elemento en 6º posicion con insert()
    miVector.insert (miVector.begin()+5,10); // inserto 10 en la 6º posicion
    cout << "Despues de introducir  10 en la sexta posicion " << '\n';
    for( auto elem : miVector) cout << elem << " ";
    cout << endl;

    return 0;
}
```

```
// Ilustra como trabajar con una pila empleando un vector STL
// introducir elemento en pila con push_back
// sacar elemento de pila con pop_back
// leer el elemento que va a salir con .back()


#include <vector>
#include <iostream>

using namespace std;

int main()
{
    int suma=0;
    vector<int> miPila; // implementaremos una pila (LIFO) con un vector

    for (int n=0; n < 10; n++)  miPila.push_back(n); // insertamos del 0 al 9

    // visualizamos el contenido
    cout << "Contenido inicial: " << endl;
    for( auto elem : miPila) cout << elem << " ";
    cout << endl;

    // ahora vamos a ir sacandolos en plan LIFO y sumandolos
    cout << "Vamos sacando elementos en plan LIFO:" << endl;
    while (!miPila.empty())
    {
        suma+=miPila.back();    //suma ultimo elemento del vector
        miPila.pop_back();      //extrae ultimo elemento del vector
        for( auto elem : miPila) cout << elem << " ";
        cout << endl;
    }

    cout << "Los elementos que había en mi pila sumaban " << suma << '\n';

    return 0;

}
```