```
//ejemplo de templates y sobrecarga
#include<iostream>
using namespace std;
#include<vector>

//funcion template que devuelve el mayor elemento de un
//vector de ... lo que sea
template <class T> T maximo(const vector<T> &a)
{
  T max=a[0];
  for (int i=1;i<a.size();i++)
    if (a[i]>max)
      max=a[i];
  return max;
}

//funcion template que calcula la suma de dos vectores de T
template <class T> void suma(const vector<T> &a,
                             const vector<T> &b,
                                   vector<T> &c)
{
  c=vector<T>(a.size());
  for (int i=0;i<a.size();i++)
    c[i]=a[i]+b[i];
}

//version sobrecargada de la funcion template suma con dos
//parametros, devolviendo el resultado con return
template <class T> vector<T> suma(const vector<T> &a,
                                  const vector<T> &b)
{
  vector<T> c(a.size());
  for (int i=0;i<a.size();i++)
    c[i]=a[i]+b[i];
  return c;
}

//funcion template para leer un vector por el teclado
//el vector ya tiene especificado el tamaÃ±o
template <class T> void lee_vector(vector<T> &a)
{
  for (int i=0;i<a.size();i++)
    {
      cout<<'['<<i<<"]=";
      cin>>a[i];
    }
}

//sobrecarga de la funcion template para leer un vector por
//el teclado, especificando el tamaÃ±o
template <class T> void lee_vector(vector<T> &a, int tam)
{
  a=vector<T>(tam);
  for (int i=0;i<a.size();i++)
    {
      cout<<'['<<i<<"]=";
      cin>>a[i];
    }
}

//funcion template para mostrar un vector por la pantalla
template <class T> void muestra_vector(const vector<T> &a)
{
  for (int i=0;i<a.size();i++)
    cout<<a[i]<<' ';
}

int main()
{
  vector<int> a(5),b,c;
  //llamadas a los templates con vector de int
  lee_vector(a);
  cout<<maximo(a)<<endl;
  lee_vector(b,a.size());
  suma(a,b,c);
  muestra_vector(c);
  cout<<endl;

  vector<float>x,y;
  //llamadas a los templates con vector de float
  lee_vector(x,3);
  lee_vector(y,3);
  muestra_vector(suma(x,y));
  cout<<endl<<maximo(x)<<endl;
}

//expurgar los elementos repetidos de un vector, version template
#include<vector>
#include<iostream>
using namespace std;

template <class T> vector<T>  unicos(const vector<T>&a)
{
  //se marcan las repeticiones con false
  //en un vector de booleanos auxiliar
  vector<bool> b(a.size(),true);
  for (int i=0;i<a.size()-1;i++)
    for (int j=i+1;j<a.size();j++)
      if (a[i]==a[j])
        b[j]=false;

  vector<T> unicos;

  //se recorren en paralelo el vector original
  //y el de booleanos, si el booleano es true
  //el elemento original es unico y se anade a unicos
  for (int i=0;i<a.size();i++)
    if (b[i])
      unicos.push_back(a[i]);

  return unicos;
}

int main()
{
  vector<int>a(10);
  for (int i=0;i<a.size();i++)
    cin>>a[i];
  vector<int>b;
  //llamada con un vector de int
  b=unicos(a);
  for (int i=0;i<b.size();i++)
    cout<<b[i]<<' ';
  cout<<endl;

  vector<char>x(10);
  for (int i=0;i<x.size();i++)
    cin>>x[i];
  vector<char>y;
  //llamada con un vector de char
  y=unicos(x);
  for (int i=0;i<y.size();i++)
    cout<<y[i]<<' ';
}
```

```
//ejemplo de argumentos procedurales y templates
//ordenacion por el metodo de la burbuja
//por cierto, uno de los PEORES metodos que hay
#include<iostream>
using namespace std;
#include<vector>
//ordena un vector de T segun criterio
template <class T> void ordena(vector<T> &a,
                               bool (*criterio)(const T &,const T &) )
{
  //si el vector tiene al menos tamaño 2
  if (a.size()>1)
    //para cada elemento
    for (int i=0;i<a.size()-1;i++)
      //comparandolo con los siguientes
      for (int j=i+1;j<a.size();j++)
        //si el primero no cumple el criterio
        //comparandolo con el segundo
        if(!(*criterio)(a[i],a[j]))
          {
            T tmp=a[i];
            a[i]=a[j];
            a[j]=tmp;
          }
}

//produce una copia ordenada utilizando
//la funcion anterior
template <class T> void copia_ordenada(const vector<T> &a, vector<T> &b,
                                       bool (*criterio)(const T &,const T &) )
{
  b=a;
  ordena(b,(*criterio));
}

//definicion del criterio <
template <class  T> bool menor(const T &x,const  T &y)
{
  return x<y;
}

//definicion del criterio >
template <class  T> bool mayor(const T &x,const  T&y)
{
  return x>y;
}

//operador <<
template <class T> ostream& operator<<(ostream &o,const  vector<T> &x)
{
  for (int i=0;i<x.size();i++)
    o<<x[i]<<' ';
  o<<endl;
  return o;
}

//operador >>
template <class T> istream& operator>>(istream &in,vector<T> &x)
{
  for (int i=0;i<x.size();i++)
    in>>x[i];
  return in;
}

int main()
{
  vector<int> x(5),y;
  cin>>x;
  //en y se guarda una copia ordenada de mayor a menor de x
  copia_ordenada(x,y,mayor);
  cout<<y;
  //en y se guarda una copia ordenada de menor a mayor de x
  copia_ordenada(x,y,menor);
  cout<<y;
  //x no cambia
  cout<<x;
  //ahora x si va  cambiar
  ordena(x,mayor);
  cout<<x;

  //ahora viene lo bueno, como mayor y menor son templates,
  //todo funciona mientras se trate de tipos para los que estan
  //definidos el < y el >. Por ejemplo matrices (vectores de vectores)
  //recordar que se hace una ordenacion lexicografica
  //similar a la ordencion alfabetica de nombres y apellidos

  vector<vector<char> > a(5,vector<char>(2));

  //como el >> y el << son templates, no hace falta redefinirlos
  //para matrices
  cin>>a;
  ordena(a,mayor);
  cout<<a;
}
```