Biblioteca Algorithm

Jhosimar George Arias Figueroa
jariasf03@gmail.com

Indice

- Ordenamiento
  - SRM 255 – SequenceOfNumbers (Divl I – 250)
  - SRM 496 – AnagramFree (Div II – 250)
  - SRM 227 – ClientsList (Div II – 500 , Div I - 250)
- Permutacion
  - POJ 1731 - Orders
  - SRM 428 – TheLuckyString (Div II – 500,Div I–250)

BIBLIOTECA ALGORITHM C++

Biblioteca Algorithm

- Esta librería es muy útil en concursos de programación ya que simplifica la codificación.
- Los métodos de esta librería, que se realizan sobre contenedores, se aplicarán por medio de iteradores.
- Es necesario agregar la siguiente librería:

```
#include <algorithm>
```

Biblioteca Algorithm

- Entre los principales métodos de esta librería tenemos:
  - **sort:** para ordenar elementos de un arreglo.
  - **next_permutation:** para permutar sobre un arreglo.
  - **reverse:** para invertir el orden de un arreglo.
  - **binary_search:** para saber la existencia de un elemento en un arreglo.
  - **max/min_element:** para obtener el máximo/mínimo elemento de un arreglo.

Ordenamiento

- Definición:
  - Dado un arreglo de elementos desordenado, ordénelos.
- Posibles Algoritmos:
  - $O(n^2)$: Bubble/Selection/Insertion Sort
  - O(n log n): Merge/Quick/Heap Sort
  - Propósitos especiales: Counting/Radix/Bucket Sort

Ordenamiento

- En un concurso solo usa dichos algoritmos si el problema me pide alguna modificación en el código de los mismos, de otro modo debemos usar las librerías para ordenamiento:
  - **sort:** implementación de introsort.
    - Puede ordenar tipos de datos básicos(enteros, doubles,chars), ordenar por varios criterios(comparador).
  - **partial_sort:** implementación de heapsort.
    - Puede ordenar en O(k log n), si solo necesitamos los k-primeros elementos ordenados.
  - **stable_sort.**
    - Si necesitamos tener un ordenamiento ‘estable’, es decir, claves con el mismo valor aparecen en el orden dado en el input.

Ordenamiento

- Para el caso de sort, la declaración sobre arreglos es la siguiente:

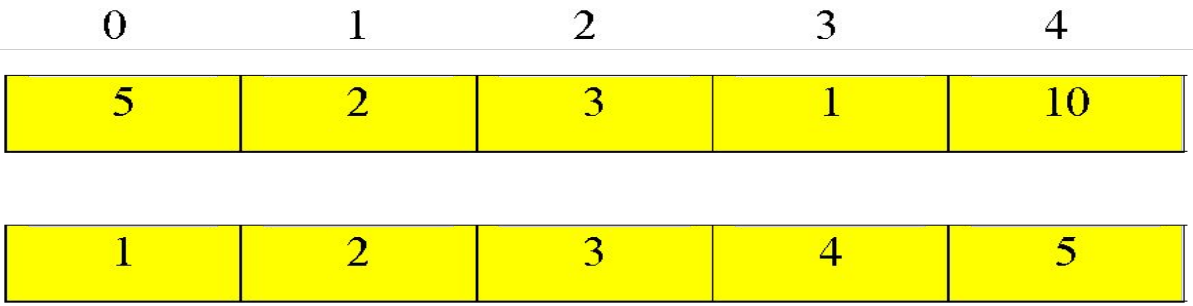
```
//Ordena un arreglo statico  en el rango [ i , j >
sort( arreglo + i , arreglo + j );

//Ordena un arreglo dinamico en el rango [ i , j >
sort( v.begin() + i , v.end() + j );
```

- El ordenamiento tiene un comparador por defecto usando el operador < , es decir, el resultado será en orden ascendente; sin embargo podemos programar un comparador propio

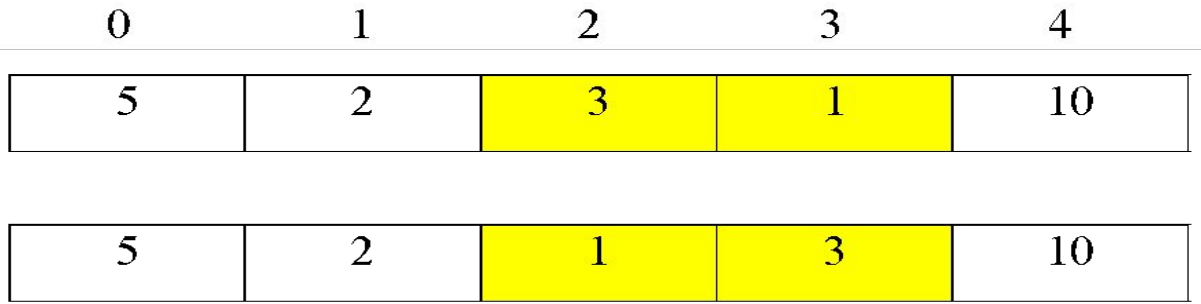
```
//Ordena un arreglo statico  en el rango [ i , j >
//siguiendo el orden dado en el comparador
sort( arreglo + i , arreglo + j , comparador );

//Ordena un arreglo dinamico en el rango [ i , j >
//siguiendo el orden dado en el comparador
sort( v.begin() + i , v.end() + j , comparador );
```

Ordenamiento

sort(0 , 5)



sort(2 , 4)



Ordenamiento

```
//Funcion comparador para ordenamiento
//Recibe 2 elementos del tipo de dato que se esta ordenando
bool comparador( int a , int b ){
    return a > b;   //Ordenamos de mayor a menor
}

int main(){
    int n;
    //Ingreso de numero de elementos
    scanf("%d" , &n );
    int a[ n + 5 ];

    //Ingreso de elementos
    for( int i = 0 ; i < n ; ++i )
        scanf("%d" , &a[ i ] );

    //Ordenamos un arreglo estatico de acuerdo al comparador
    sort( a , a + n , comparador );
    for( int i = 0 ; i < n ; ++i ){
        printf("%d " , a[ i ] );
    }
    printf("\n");
    return 0;
}
```

SRM 255 DIV II - 250

Problem Statement

It is a common mistake to sort numbers as strings. For example, a sorted sequence like {"1", "174", "23", "578", "71", "9"} is not correctly sorted if its elements are interpreted as numbers rather than strings.

You will be given a String[] sequence that is sorted in non-descending order using string comparison. Return this sequence sorted in non-descending order using numerical comparison instead.

Definition

Class:	SequenceOfNumbers
Method:	rearrange
Parameters:	String[]
Returns:	String[]
Method signature:	String[] rearrange(String[] sequence)
(be sure your method is public)	

Constraints

- **sequence** will contain between 2 and 50 elements inclusive.
- Each element of **sequence** will contain between 1 and 9 characters inclusive.
- Each element of **sequence** will consist of only digits ('0'-'9').
- Each element of **sequence** will not start with a '0' digit.
- **sequence** will be ordered lexicographically.

Examples

0)

```
{"1","174","23","578","71","9"}

Returns: {"1", "9", "23", "71", "174", "578" }
```

1)

```
{"172","172","172","23","23"}

Returns: {"23", "23", "172", "172", "172" }
```

2)

```
{"183","2","357","38","446","46","628","734","741","838"}

Returns: {"2", "38", "46", "183", "357", "446", "628", "734", "741", "838" }
```

Solución

- Solución 1: Realizar el ordenamiento mediante un comparador:
  - Si tamaños iguales devolver el menor.
  - Si tamaños diferentes devolver el de menor tamaño.
- Solución 2: Conversión de tipos y ordenar:
  - Posible uso de stringstream.
  - Posible uso de sscanf y sprintf

SRM 496 DIV II - 250

Problem Statement

A string X is an anagram of string Y if X can be obtained by arranging all characters of Y in some order, without removing any characters and without adding new characters. For example, each of the strings "baba", "abab", "aabb" and "abba" is an anagram of "aabb", and strings "aaab", "aab" and "aabc" are not anagrams of "aabb".

A set of strings is anagram-free if it contains no pair of strings which are anagrams of each other. Given a set of strings S, return the size of its largest anagram-free subset. Note that the entire set is considered a subset of itself.

Definition

Class:	AnagramFree
Method:	getMaximumSubset
Parameters:	String[]
Returns:	int
Method signature:	int getMaximumSubset(String[] S)

(be sure your method is public)

Constraints

- S will contain between 1 and 50 elements, inclusive.
- Each element of S will contain between 1 and 50 characters, inclusive.
- Each element of S will consist of lowercase letters ('a'-'z') only.
- All elements of S will be distinct.

Examples

0)

```
{"abcd","abdc","dabc","bacd"}
```

Returns: 1

All of the strings in S are anagrams of each other, so no two of them can simultaneously belong to an anagram-free set.

1)

```
{"abcd","abac","aabc","bacd"}
```

Returns: 2

One of the maximum anagram-free subsets of S is {"abcd","abac"}.

2)

```
{"aa","aaaaa","aaa","a","bbaaaa","aaababaa"}
```

Returns: 6

Note that strings of different length cannot be anagrams of each other.

3)

```
{"creation","sentence","reaction","sneak","star","rats","snake"}
```

Returns: 4

Solución

- Solución 1: Ordenar cada palabra e iterar y contar palabras que no se repitan.
- Solución 2: Ordenar cada palabra y agregarlas a un Set, devolver el tamaño del Set.

SRM 227 DIV II – 500, DIV I- 250

Problem Statement

Your company has just undergone some software upgrades, and you will now be storing all of the names of your clients in a new database. Unfortunately, your existing data is inconsistent, and cannot be imported as it is. You have been tasked with writing a program to cleanse the data.

You are given a String[], **names**, which is a list of the names of all of your existing clients. Some of the names are in "First Last" format, while the rest are in "Last, First" format. You are to return a String[] with all of the names in "First Last" format, sorted by last name, then by first name.

Definition

Class:	ClientsList
Method:	dataCleanup
Parameters:	String[]
Returns:	String[]
Method signature:	String[] dataCleanup(String[] names)
(be sure your method is public)	

Constraints

- **names** will contain between 1 and 50 elements, inclusive.
- Each element of **names** will be of the form "First Last" or "Last, First" (quotes added for clarity).
- Each first and last name will begin with a single capital letter 'A'-'Z', and the remaining letters will be lower case 'a'-'z'.
- Each element of **names** will contain between 3 and 50 characters, inclusive.

Examples

0)

```
{"Joe Smith", "Brown, Sam", "Miller, Judi"}

Returns: { "Sam Brown",  "Judi Miller",  "Joe Smith" }
```

The last names, in order, are Brown, Miller, Smith. We rearrange each name to be in the proper format also.

1)

```
{"Campbell, Phil", "John Campbell", "Young, Warren"}
Returns: { "John Campbell",  "Phil Campbell",  "Warren Young" }
```

Notice here how we sort by first name when the last names are the same.

2)

```
{"Kelly, Anthony", "Kelly Anthony", "Thompson, Jack"}

Returns: { "Kelly Anthony",  "Anthony Kelly",  "Jack Thompson" }
```

Be careful to properly identify first name versus last name!

SRM 227 DIV II – 500, DIV I- 250

2)

```
{"Kelly, Anthony", "Kelly Anthony", "Thompson, Jack"}

Returns: { "Kelly Anthony",  "Anthony Kelly",  "Jack Thompson" }
```

Be careful to properly identify first name versus last name!

3)

```
{"Trevor Alvarez", "Jackson, Walter", "Mandi Stuart",
 "Martin, Michael", "Peters, Tammy", "Richard Belmont",
 "Carl Thomas", "Ashton, Roger", "Jamie Martin"}

Returns:
{ "Trevor Alvarez",
 "Roger Ashton",
 "Richard Belmont",
 "Walter Jackson",
 "Jamie Martin",
 "Michael Martin",
 "Tammy Peters",
 "Mandi Stuart",
 "Carl Thomas" }
```

4)

```
{"Banks, Cody", "Cody Banks", "Tod Wilson"}

Returns: { "Cody Banks",  "Cody Banks",  "Tod Wilson" }
```

Notice that two identical names can appear in the list.

5)

```
{"Mill, Steve", "Miller, Jane"}

Returns: { "Steve Mill",  "Jane Miller" }
```

Notice that when shorter names precede longer names alphabetically, when the shorter name is a substring of the longer.

Explicación del Problema

```
{"Campbell, Phil", "Young, Warren", "John Campbell"}

Returns: { "John Campbell",  "Phil Campbell",  "Warren Young" }
```

- Dados nombres y apellidos:

  “Campbell, Phil” -> “Phil Campbell”

  “Young, Warren” -> “Warren Young”

  “John Campbell” -> “John Campbell”

Explicación del Problema

- Tenemos que ordenar por apellido:

"Phil Campbell"

"John Campbell"

"Warren Young"

- Si poseen igual apellido, ordenamos por nombre:

"John Campbell"

"Phil Campbell"

"Warren Young"

Solución

- Uso de **stringstream** para parsear entrada.
- Uso de **find** para detectar las comas.
- Realizar ordenamiento mediante un comparador.
- Comparar primero por apellido y luego por nombre.
- Posible uso de **pair< T , T >.**

Problemas

- TopCoder Problem Archive – Sorting
- UVA Toolkit - Sorting

Permutaciones

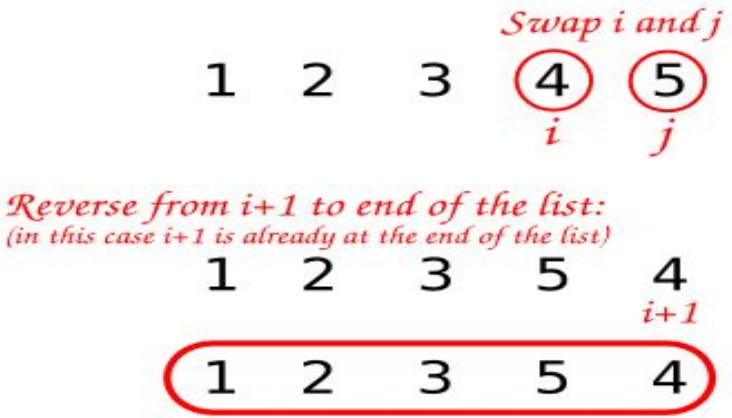
- Definición:
  - Dado un arreglo, mostrar todas las posibles permutaciones de dicho arreglo.
- La complejidad de hallar todas las permutaciones en un arreglo de tamaño n es O(n!).
- Usaremos este algoritmo solo cuando el tamaño sea: n <= 10

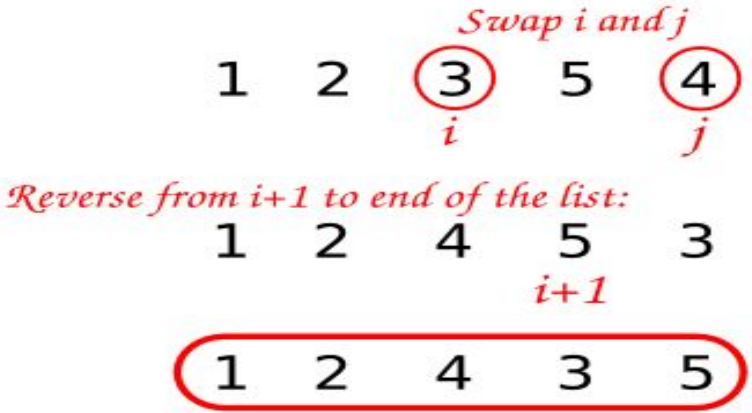
Permutaciones

- La librería algorithm nos otorga 2 métodos para trabajar con permutaciones:
  - **next_permutation:** Reordena los elementos del arreglo obteniendo la siguiente permutación del arreglo.
  - **prev_permutation:** Reordena los elementos del arreglo obteniendo la anterior permutación del arreglo.

Permutaciones

- El algoritmo que se usa para hallar la siguiente permutación es:





Permutaciones

- La declaración del método es de la siguiente manera:

```
//Hallamos siguiente permutacion de arreglo statico  en el rango [ i ,j >
next_permutation( arreglo + i , arreglo + j );

//Hallamos siguiente permutacion de arreglo dinamico en el rango [ i ,j >
next_permutation( v.begin() + i , v.end() + j );
```

- De manera similar se declara para prev_permutation:

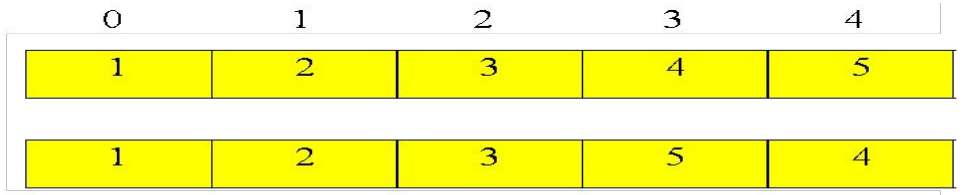
```
//Hallamos anterior permutacion de arreglo statico  en el rango [ i ,j >
prev_permutation( arreglo + i , arreglo + j );

//Hallamos anterior permutacion de arreglo dinamico en el rango [ i ,j >
prev_permutation( v.begin() + i , v.end() + j );
```

Permutaciones

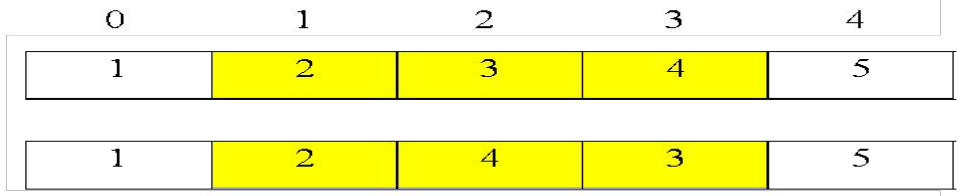
- Para hallar todas las posibles permutaciones usando los 2 métodos:
  - next_permutation: es necesario ordenar el arreglo ingresado en orden ascendente.
  - prev_permutation: es necesario ordenar el arreglo ingresado en orden descendente.

Permutaciones

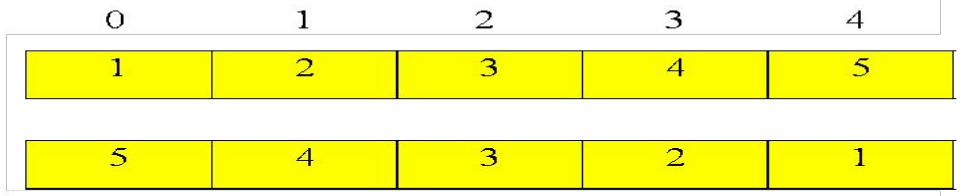
next_permutation(0 , 5)



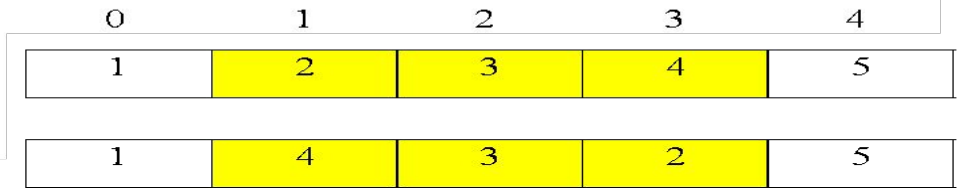
next_permutation(1 , 4)



prev_permutation(0 , 5)

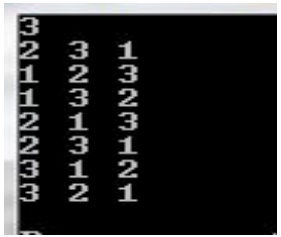


prev_permutation(1 , 4)



Permutaciones

```
int main(){
    int n;
    scanf("%d" , &n );
    int a[ n + 5 ];
    for( int i = 0 ; i < n ; ++i ){
        scanf("%d" , &a[ i ] );
    }
    //Ordenamos el arreglo en orden ascendente
    sort( a , a + n );

    do{
        //Realizamos cualquier tipo de operacion en la
        //permutación actual
        for( int i = 0 ; i < n ; ++i ){
            printf("%d " , a[ i ] );
        }
        printf("\n");

    }while( next_permutation( a , a + n ) );

    return 0;
}
```



Orders

Language: Default

Time Limit: 1000MS **Memory Limit:** 10000K
Total Submissions: 7646 **Accepted:** 4809

Description

The stores manager has sorted all kinds of goods in an alphabetical order of their labels. All the kinds having labels starting with the same letter are stored in the same warehouse (i.e. in the same building) labelled with this letter. During the day the stores manager receives and books the orders of goods which are to be delivered from the store. Each order requires only one kind of goods. The stores manager processes the requests in the order of their booking.

You know in advance all the orders which will have to be processed by the stores manager today, but you do not know their booking order. Compute all possible ways of the visits of warehouses for the stores manager to settle all the demands piece after piece during the day.

Input

Input contains a single line with all labels of the requested goods (in random order). Each kind of goods is represented by the starting letter of its label. Only small letters of the English alphabet are used. The number of orders doesn't exceed 200.

Output

Output will contain all possible orderings in which the stores manager may visit his warehouses. Every warehouse is represented by a single small letter of the English alphabet -- the starting letter of the label of the goods. Each ordering of warehouses is written in the output file only once on a separate line and all the lines containing orderings have to be sorted in an alphabetical order (see the example). No output will exceed 2 megabytes.

Sample Input

```
bbjd
```

Sample Output

```
bbdj
bbjd
bdbj
bdjb
bjbd
bjdb
dbbj
dbjb
djbb
jbbd
jbdb
jdbb
```

SRM 428 DIV2 – 500 , DIV1 - 250

Problem Statement

John and Brus are studying string theory at the university. According to Brus, a string is called lucky if no two consecutive characters are equal. John is analyzing a String s, and he wants to know how many distinct lucky strings can be generated by reordering the letters in s. If s is a lucky string in its original ordering, it should also be considered in the count.

Definition

Class:	TheLuckyString
Method:	count
Parameters:	String
Returns:	int
Method signature:	int count(String s)

(be sure your method is public)

Constraints

- s will contain between 1 and 10 characters, inclusive.
- Each character of s will be a lowercase letter ('a' - 'z').

Examples

0)

"ab"

Returns: 2

Two lucky strings - "ab" and "ba".

1)

"aaab"

Returns: 0

It's impossible to construct a lucky string.

2)

"aabbbaa"

Returns: 1

"abababa" is the only lucky string that can be generated.

3)

"abcdefghij"

Returns: 3628800

Explicación del Problema

- **Input:** aaab
- Distintos strings generados al reordenar letras:

  aaab

  aaba

  abaa

  baaa

- El input es llamado Lucky si no posee caracteres consecutivos repetidos.

Explicación del Problema

- Revisamos caracteres consecutivos.

aaab

aaba

abaa

baaa

- Ningun ordenamiento es Lucky por lo tanto retornamos 0.

Explicación del Problema

- **Input:** abac
- Tenemos los posibles ordenamientos:

aabc	baac
aacb	baca
abac	bcaa
abca	caab
acab	caba
acba	cbaa

Explicación del Problema

- Revisamos caracteres consecutivos:

aabc	baac
aacb	baca
abac	bcaa
abca	caab
acab	caba
acba	cbaa

- Para este caso tenemos 6 cadenas Lucky.

Solución

- El tamaño maximo de la cadena es 10, por lo tanto es posible generar todas las posibles permutaciones de la cadena ingresada.
- Revisar en cada permutación si la cadena ingresada es Lucky.

PROBLEMAS

- SRM 533 – CasketOfStarEasy (Div II – 500)
- POJ 1256 - Anagram
- UVA 1209 - Wordfish
- UVA 11742 – Social Constraints

Macros

- Incluir todas las cabeceras que utilices.

```
#include <stdio.h>
#include <algorithm>
#include <cstring>
#include <stdlib.h>
#include <cctype>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <functional>
#include <numeric>
#include <utility>
#include <deque>
#include <stack>
#include <bitset>
#include <map>
#include <set>
#include <string>
#include <vector>
#include <queue>
#include <limits>
#include <fstream>
#include <list>
#include <sstream>
#include <iostream>
#include <iomanip>
using namespace std;
```

- O usar solo la siguiente librería en GCC:

#include<bits/stdc++.h>

Macros

- Usar accesos rápidos para tipos de datos comunes.

```
typedef long long                       ll;
typedef long double                     ld;
typedef vector< int >                   vi;
typedef vector< vi >                    vvi;
typedef pair< int , int >               pii;
typedef set< int >                      si;
typedef pair< string , string >         pss;
typedef vector< pair< int , int > >     vpii;
```

```
#define REP(i, a, b) for (int i = int(a); i <= int(b); ++i )
#define REPN(i, n) REP (i, 1, int(n))
#define REPD(i, a, b) for (int i = int(a); i >= int(b); --i )
#define TR(c, it) for (vpii::iterator it = (c).begin(); it != (c).end(); ++it )
//#define TR(c, it) for (typeof((c).begin()) it = (c).begin(); it != (c).end(); ++it )
// solo en UNIX
```

Macros

- Más accesos rápidos

```
#define sz(a) int((a).size())
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define present ( c , x ) ( ( c ).find( x ) != ( c ).end() )
#define show(x)  cerr<<#x<<" "<<x<<endl;
```

- Revisar la siguiente página:
  - http://gcc.gnu.org/onlinedocs/cpp/Macros.html