Clases (Uso)

Algoritmos y Estructuras de Datos II

Objetivos: Materia

Resolver problemas más grandes en complejidad y en escala.
Para ello estudiaremos las siguientes herramientas:

- Especificación con Tipos Abstractos de Datos (TADs).
- Diseño de módulos.
- Análisis de complejidad.
- Estructuras de datos avanzadas.
- Técnicas algorítmicas:
  - "Divide y Reinarás" (Divide and Conquer).
  - Ordenamiento.

Objetivos: Labo

- Ejercitar la descomposición de problemas en partes.
- Aprender a representar estas partes con módulos.
- Escribir estas soluciones en C++.
- Llevar a la práctica los contenidos vistos en el resto de la materia (estructuras de datos y técnicas algorítmicas).

Estructura labo

Clases

- ~~Presentación de temas en aula (2hs).~~
  Clases grabadas anteriormente.
- ~~Resolución de ejercicios en labo (3hs).~~
  Consultas sincrónicas (∼3hs)

Ejercitaciones

- Todas las clases: Guías de ejercicios para resolver en ~~clase~~ casa.
- ~~3 TPs: Diseño, Elección de Estructuras e Implementación.~~
  4 Entregables correlativos grupales (4 participantes max).
- Talleres obligatorios (~~∼3~~ 4).

Guías

- Entregables en guías unificadas (ver campus).

Módulos y Clases

- Problema chico $\rightarrow$ funciones
- Problema mediano $\rightarrow$ TAD
- Problema grande $\rightarrow$ TADs

Módulos y Clases

- Problema chico $\rightarrow$ funciones
- Problema mediano $\rightarrow$ TAD
- Problema grande $\rightarrow$ TADs

Módulo

Encapsulamiento de un problema en un conjunto de operaciones.
Conjunto de operaciones $\rightarrow$ Interfaz

Clase

Implementación de Módulos en un lenguaje de programación.

Clases: definición

```
template <class T>
class vector<T> {
public:
    // Interfaz de la clase
    vector();                   // Constructor.
    void push_back(T elem);     // Agrega un elemento al final.
    T operator[](int idx);      // Devuelve el i-ésimo elemento.
    int size();                 // Devuelve el tamaño del vector.
    T front();                  // Devuelve el primer elemento.
    T back();                   // Devuelve el último elemento.
};
```

Clases: uso

```cpp
#include <vector>

using namespace std;

int main() {
  vector<int> v1;
  cout << v1.size() << endl; // 0
  v1.push_back(1);
  v1.push_back(2);
  cout << v1.size() << endl; // 2
  cout << v1[0] << endl;     // 1
  cout << v1[1] << endl;     // 2

  vector<string> vs = vector<string>();
  vs.push_back("Hola");
  vs.push_back("mundo");
  cout << vs[0] << " " << vs[1] << endl; // "Hola mundo"
};
```

TAD Secuencia

TAD SECUENCIA(α)

parámetros formales

géneros α

observadores básicos

vacía?	:	secu(α)	$\longrightarrow$	bool	
prim	:	secu(α) s	$\longrightarrow$	α	$\{\neg$ vacía?(s)$\}$
fin	:	secu(α) s	$\longrightarrow$	secu(α)	$\{\neg$ vacía?(s)$\}$

generadores

$<>$	:		$\longrightarrow$	secu(α)	
$\bullet \bullet \bullet$	:	$\alpha \times$ secu(α)	$\longrightarrow$	secu(α)	

otras operaciones

$\bullet \circ \bullet$	:	secu(α) $\times$ α	$\longrightarrow$	secu(α)	
$\bullet$ & $\bullet$	:	secu(α) $\times$ secu(α)	$\longrightarrow$	secu(α)	
ult	:	secu(α) s	$\longrightarrow$	α	$\{\neg$ vacía?(s)$\}$
com	:	secu(α) s	$\longrightarrow$	secu(α)	$\{\neg$ vacía?(s)$\}$
long	:	secu(α)	$\longrightarrow$	nat	

Fin TAD

TAD Secuencia

No cuenta con una operación para obtener el i-ésimo elemento.
La agregamos:

$\bullet[\bullet] : \text{secu}(\alpha)\ s \times \text{nat}\ i \longrightarrow \alpha \qquad \{\text{i} < \text{long(s)}\}$

$s[i] \equiv \textbf{if}\ i = 0\ \textbf{then}\ \text{prim(s)}\ \textbf{else}\ \text{fin(s)}[i-1]\ \textbf{fi}$

Problema: especificación

PARES_E_IMPARES(**in** s : secu(nat))
$\rightarrow$ res : tupla(secu(nat), secu(nat))

Pre $\equiv$ {true}

Post $\equiv$ {

(los elementos de la primera secuencia son pares)
$(\forall$ i : Nat$)$ $(i < long(\pi_1(res)) \Rightarrow \pi_1(res)[i]$ mod $2 = 0)$

$\wedge$ (los elementos de la segunda secuencia son impares)
$(\forall$ i : Nat$)$ $(i < long(\pi_2(res)) \Rightarrow \pi_2(res)[i]$ mod $2 = 1)$

$\wedge$ (todos los elementos de la entrada están en la salida)
$(\forall$ i : Nat$)$
$(i < long(s) \Rightarrow pertenece(s[i], \pi_1(res)) \vee pertenece(s[i], \pi_2(res)))$

$\wedge$ (sólo los elementos de la entrada están en la salida)
$(\forall$ i : Nat$)$ $(i < long(\pi_1(res)) \Rightarrow pertenece(\pi_1(res)[i], s)) \wedge$
$(\forall$ i : Nat$)$ $(i < long(\pi_2(res)) \Rightarrow pertenece(\pi_2(res)[i], s))$

}

Descripción: separa los elementos de una secuencia en dos secuencias, una con los elementos pares y otra con los impares.

$$(\forall\ x : \alpha)\ (\forall\ s : \mathsf{secu}(\alpha))$$

$$\mathsf{pertenece}(x,\ s)\ \iff\ (\exists i : \mathsf{nat})\ (i < \mathsf{long}(s) \wedge s[i] = x)$$

Problema: implementación

```
#include <vector>

using namespace std;

pair<vector<int>, vector<int>> pares_e_impares(vector<int> s)
```

Problema: implementación

```cpp
#include <vector>

using namespace std;

pair<vector<int>, vector<int>> pares_e_impares(vector<int> s) {
  vector<int> pares;
  vector<int> impares;
  for (int i = 0; i < s.size(); i++) {
    if (s[i] % 2 == 0) {
       pares.push_back(s[i]);
    } else {
       impares.push_back(s[i]);
    }
  }
  return pair<vector<int>, vector<int>>(pares, impares);
}
```

Interfaz de la clase `vector` según `cppreference`

`https://es.cppreference.com/w/cpp/container/vector`

std::vector

Definido en la cabecera <vector>

```
template<
    class T,
    class Allocator = std::allocator<T>
> class vector;
```

`std::vector` es un contenedor que encapsula secuencia de matrices dinámicas tamaño . Los elementos se almacenan de forma contigua, lo que significa que los elementos no se puede acceder sólo a través de los iteradores, pero también con los desplazamientos en punteros regulares a los elementos. Esto significa que un puntero a un elemento de un vector se puede hacer pasar a cualquier función que espera un puntero a un elemento de una matriz . El almacenamiento del vector se maneja automáticamente, siendo expandido y contraído, según sea necesario. Vectores suelen ocupar más espacio que las matrices estáticas, ya que se asigna más memoria para manejar el crecimiento futuro. De esta manera un vector no necesita reasignar cada vez que se inserta un elemento, pero sólo cuando la memoria adicional se agota. La cantidad total de memoria asignada puede ser consultada mediante la función `capacity()`. Memoria adicional se puede devolver al sistema a través de una llamada a `shrink_to_fit()` . Las reasignaciones suelen ser operaciones costosas en términos de rendimiento. `reserve()` función puede ser utilizada para eliminar las reasignaciones si el número de elementos que se conoce de antemano . La complejidad (eficacia) de las operaciones comunes de vectores es el siguiente:

- Acceso aleatorio - $O(1)$ constante
- Inserción o extracción de elementos al final - $O(1)$ constante amortizado
- La inserción o la eliminación de los elementos - lineal en distancia hasta el extremo de la $O(n)$ vector

`std::vector` cumple los requisitos de `Container`, `AllocatorAwareContainer`, `SequenceContainer` y `ReversibleContainer` .

Interfaz de la clase `vector` según `cppreference`

Las funciones miembro

(constructor)	construye el `vector` (función miembro público)	[editar]
(destructor)	destructs the `vector` (función miembro público)	[editar]
`operator=`	asigna valores para el contenedor (función miembro público)	[editar]
`assign`	asigna valores para el contenedor (función miembro público)	[editar]
`get_allocator`	devuelve el asignador asociado (función miembro público)	[editar]
Elemento acceso		
`at`	acceder al elemento especificado con comprobación de límites (función miembro público)	[editar]
`operator[]`	acceder al elemento especificado (función miembro público)	[editar]
`front`	acceso al primer elemento (función miembro público)	[editar]
`back`	access the last element (función miembro público)	[editar]
`data` (C++11)	dirigir el acceso a la matriz subyacente (función miembro público)	[editar]

Alternativamente:
`http://www.cplusplus.com/reference/vector/vector/`

Interfaz de la clase `vector` según `cppreference`

std::vector::operator[]

This page has been machine-translated from the English version of the wiki using Google Translate.
The translation may contain errors and awkward wording. Hover over text to see the original version. You can help to fix errors and improve the translation. For instructions click here.

```
reference       operator[]( size_type pos );
const_reference operator[]( size_type pos ) const;
```

Devuelve una referencia al elemento en pos ubicación especificada. Sin comprobación de límites se realiza .

Parámetros

pos - la posición del elemento para volver

Valor de retorno

referencia para el elemento solicitado

Complejidad

Constant

Ejemplo

El código siguiente utiliza `operator[]` leer y escribir en un `std::vector<int>`:

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

int main()
{
    std::vector<int> numbers {2, 4, 6, 8};

    std::cout << "Second element: " << numbers[1] << '\n';

    numbers[0] = 5;

    std::cout << "All numbers:";
    for (auto i : numbers) {
        std::cout << ' ' << i;
    }
    std::cout << '\n';
}
```

Salida:

```
Second element: 4
All numbers: 5 4 6 8
```

Conjunto

Colección (finita) de elementos sin distinguir orden ni multiplicidad.
Está caracterizado por el observador $\bullet \in \bullet$ que indica pertenencia.

TAD CONJUNTO(α)

parámetros formales

géneros α

observadores básicos

$\bullet \in \bullet$	:	$\alpha \times \text{conj}(\alpha)$	$\longrightarrow$	bool

generadores

$\emptyset$	:		$\longrightarrow$	conj(α)
Ag	:	$\alpha \times \text{conj}(\alpha)$	$\longrightarrow$	conj(α)

otras operaciones

$\emptyset?$	:	conj(α)	$\longrightarrow$	bool
#	:	conj(α)	$\longrightarrow$	nat
dameUno	:	conj(α) c	$\longrightarrow$	α $\quad \{\neg\emptyset?(c)\}$
sinUno	:	conj(α) c	$\longrightarrow$	conj(α) $\quad \{\neg\emptyset?(c)\}$

Fin TAD

Conjunto (c++)

```
template<class T>
class set<T> {
public:
  set();              // Construye un conjunto vacío.
  void insert(T x); // Inserta un elemento.
  int count(T x);    // Devuelve la cantidad de
                      // apariciones de x (0 ó 1).
  int size();         // Devuelve el tamaño del conjunto
};
```

Problema

$\text{INTERSECCION}(\text{secu}(\alpha)\ a, \text{secu}(\alpha)\ b) \rightarrow res\ : \texttt{secu}(\alpha)$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{(\forall x : \alpha)$
$(pertenece(x, res) \iff (pertenece(x, a) \wedge pertenece(x, b)))\ \}$
Descripción: Devuelve un conjunto con los elementos en común de a y b

Problema

```
vector<int> interseccion(vector<int> a, vector<int> b)
```

Problema

```
vector<int> interseccion(vector<int> a, vector<int> b) {
  set<int> set_a;
  for (int i = 0; i < a.size(); i++) {
      set_a.insert(a[i]);
  }
  vector<int> res;
  for (int i = 0; i < b.size(); i++) {
      if (set_a.count(b[i]) == 1) {
          res.push_back(b[i]);
      }
  }
  return res;
}
```

Diccionario

Tabla que asocia *claves* a *significados*.

TAD DICCIONARIO(κ, σ)

parámetros formales

géneros κ, σ

observadores básicos

def?	:	$\kappa \times \text{dicc}(\kappa, \sigma)$	$\longrightarrow$	bool	
obtener	:	$\kappa\ c \times \text{dicc}(\kappa, \sigma)\ d$	$\longrightarrow$	σ	$\{\text{def?}(c, d)\}$

generadores

vacío	:		$\longrightarrow$	$\text{dicc}(\kappa, \sigma)$
definir	:	$\kappa \times \sigma \times \text{dicc}(\kappa, \sigma)$	$\longrightarrow$	$\text{dicc}(\kappa, \sigma)$

otras operaciones

claves	:	$\text{dicc}(\kappa, \sigma)$	$\longrightarrow$	$\text{conj}(\kappa)$

Fin TAD

Diccionario (c++)

```
template<class K, class S>
class map<K, S> {
public:
  map(); // Crea un diccionario vacío

  S operator[](K clave);
     // Devuelve el valor asociado a la clave.
     // Se puede sobreescribir para definir claves
     // (ver ejemplo).

  int count(K clave);
    // Devuelve 1 si la clave está definida.
    // Devuelve 0 si no.
};
```

Diccionario (ejemplo)

```cpp
#include <map>

using namespace std;

int main() {
  map<char, int> m;
  cout << m.count('a') << endl; // 0
  m['a'] = 5;
  m['c'] = 10;
  cout << m['a'] << endl;       // 5
  cout << m['c'] << endl;       // 10
  m['a'] = 200;
  cout << m['a'] << endl;       // 200
  cout << m.count('a') << endl; // 1
  cout << m.count('b') << endl; // 0
}
```

Para recorrer colecciones

Una *colección* es cualquier tipo de datos que contiene elementos.

¿Cómo recorremos una colección?

- En el lenguaje de especificación:
  - **Arreglo:** tamaño y acceder al i-ésimo (`operator[]`).
  - **Secuencia:** prim y fin.
  - **Conjunto:** dameUno y sinUno.
  - **Diccionario:** claves y obtener.
- En C++:
  - ~~Iteradores~~ (próximamente...)

for-range

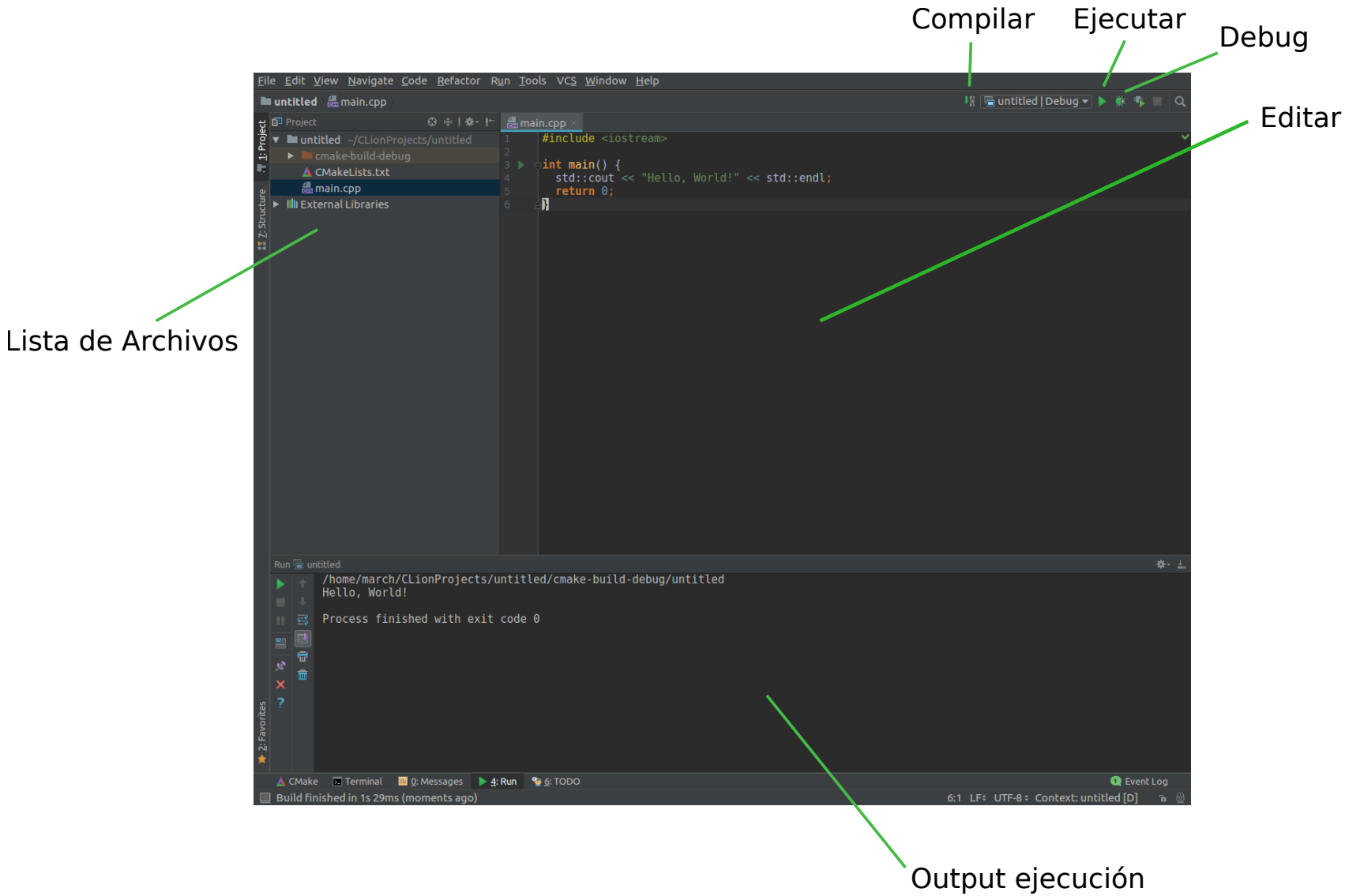
```cpp
#include <vector>
#include <set>
#include <map>
#include <string>

using namespace std;

int main() {
    vector<int> vi = {1, 2, 5, 6, 7};
    for (int n : vi) {
        cout << n << endl;
    }
    set<string> s = {"Bienvenidos/as", "a", "algoritmos", "2"};
    for (string x : s) {
        cout << x << endl;
    }
    map<int, string> m;
    m[2] = "Hola";
    m[1] = "mundo";
    for (pair<int, string> p : m) {
        cout << p.first << " -> " << p.second << endl;
    }
}
```

Ejercitación hoy

Guías de ejercicios para ejercitar el uso de colecciones y módulos.

Ambiente de desarrollo

CLion



CLion en casa

Bajar trial en: `https://www.jetbrains.com/clion/download/`
Crear una cuenta de estudiante con cuenta `@dc.uba.ar`
Licenciarlo con código de activación una vez loggeado con la cuenta en `www.jetbrains.com`

Alternativa

- Editar código: atom, sublime, vim
- Compilar: CMake + Makefile
- Ejecutar: consola
- Debuggear: gdb (valgrind)

CLion en los labos

En los labos el directorio de usuario (/home/{user}) suele llenarse hasta su capacidad (1GB) y las cosas dejan de andar.
Recomendamos los siguientes pasos antes de arrancar:

- Correr `$> du -h -d1 | sort -h` para listar los directorios ordenados crecientemente por tamaño. Si el tamaño final (el del directorio ".") es mayor a 700MB, borrar directorios no importantes. Priorizar los que más ocupan. Los candidatos suelen ser .cache o .CLion201*. .cache tiene los caches de Google Chrome y Mozzila Firefox. .Clion201* tiene las configuraciones de CLion
- Usar `$> rm -r \{directorio\}` para borrar el directorio donde {directorio} se reemplaza con el directorio a borrar. Ej: `$> rm -f .cache`

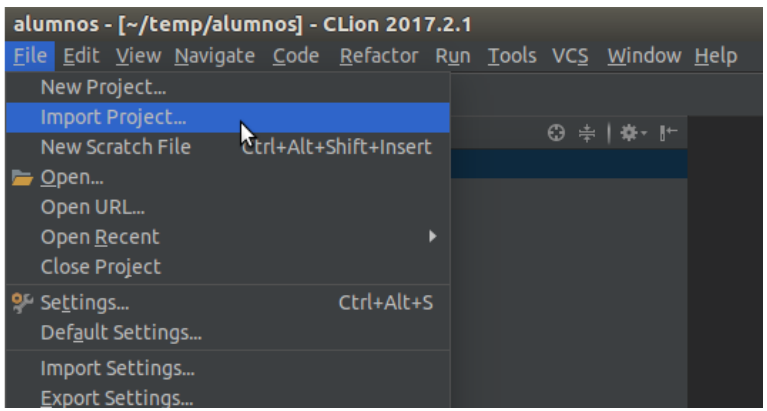
Luego, si al correr CLion pide validarlo, no es necesario usar la licencia propia, sino que se puede usar el servidor de licencias del departamento. Suele ser necesario refrescar la lista de servidores más de una vez para que lo encuentre.

Abrir CLion en los Labos

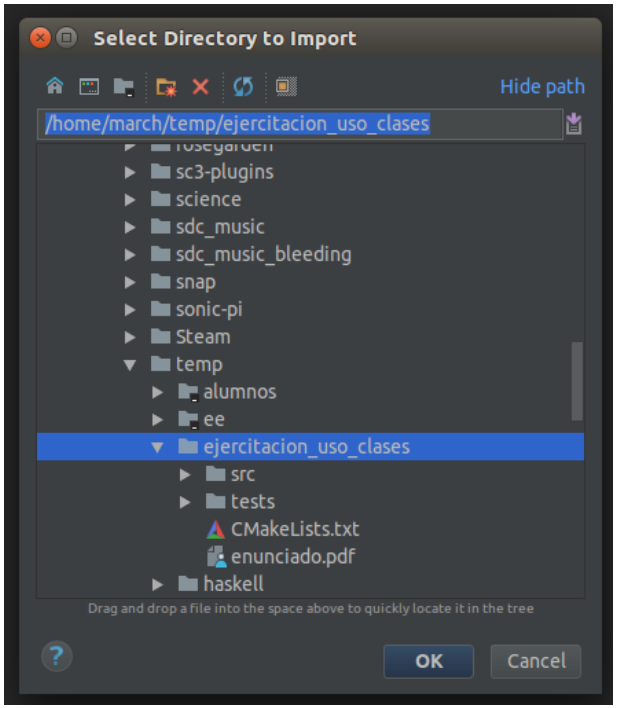
A veces CLion no aparece en la lista de aplicaciones de Ubuntu en los labos. En ese caso, se lo puede abrir corriendo `$> clion.sh` en la terminal (ctrl+alt+t). En caso de faltar actualizar la licencia, se debe usar la opción License Server y hacer Discover server hasta que aparezca el servidor de la facu.

Abrir la ejercitación en CLion

En las ejercitaciones, uno de los directorios va a contener un archivo CMakeLists.txt. Para trabajar en CLion se importa este directorio. Este archivo es el que define como se compila el projecto. En el mismo se pueden declarar varios targets que son los archivos ejecutables finales que vamos a tener después de compilar.

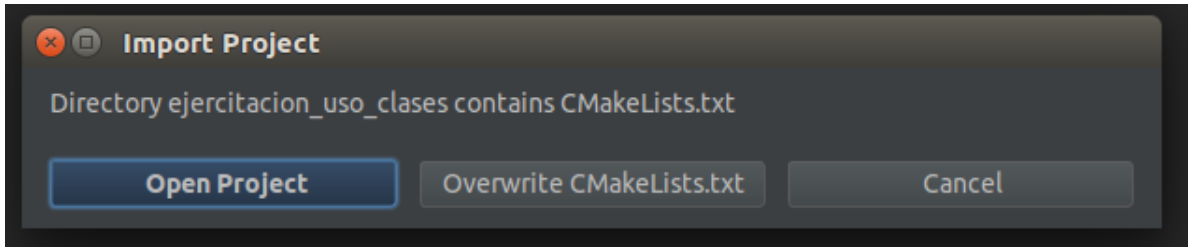


(a) File > Import Project...

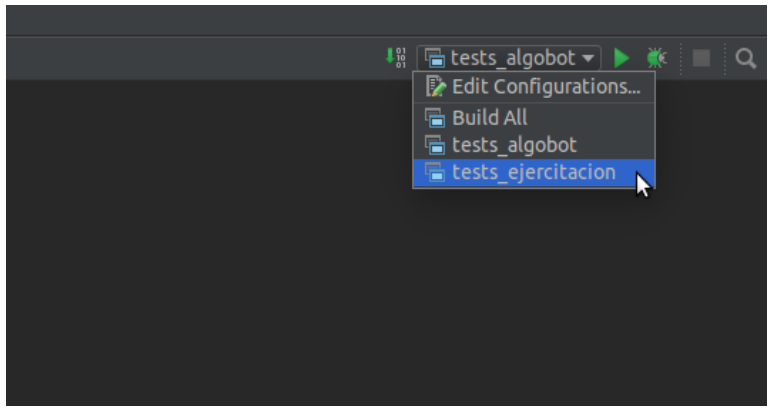


(b) Buscamos la carpeta con el CMakeLists.txt

Abrir la ejercitación en CLion



(c) IMPORTANTE: No sobreescribir el CMakeLists.txt. Elegir Open Project.



(d) Arriba a la derecha aparecerá al lado del ícono de compilar, la lista de targets