Tema – Estructuras de Datos

Dr. Carlos Segura González (carlos.segura@cimat.mx)

Objetivos

- Recordar las estructuras de datos básicas que se vieron en cursos anteriores
- Conocer el uso de las estructuras de datos principales que están implementadas en la STL y algunas estructuras de bajo nivel utilizables para operaciones más complejas
- Conocer estructuras de datos avanzadas que no se impartieron en cursos anteriores.
- Ser capaces de hacer uso de dichas estructuras en la resolución de problemas

C++ STL (Standard Template Library)

- Parte de la C++ Standard library, que ofrece estructuras de datos y algoritmos que se pueden usar con dichas estructuras.

Algoritmos

sort
set_union
set_intersection
min
max
random_shuffle
next_permutation
Etc.

Estructuras de datos
(Containers)

vector
list, forward_list
deque

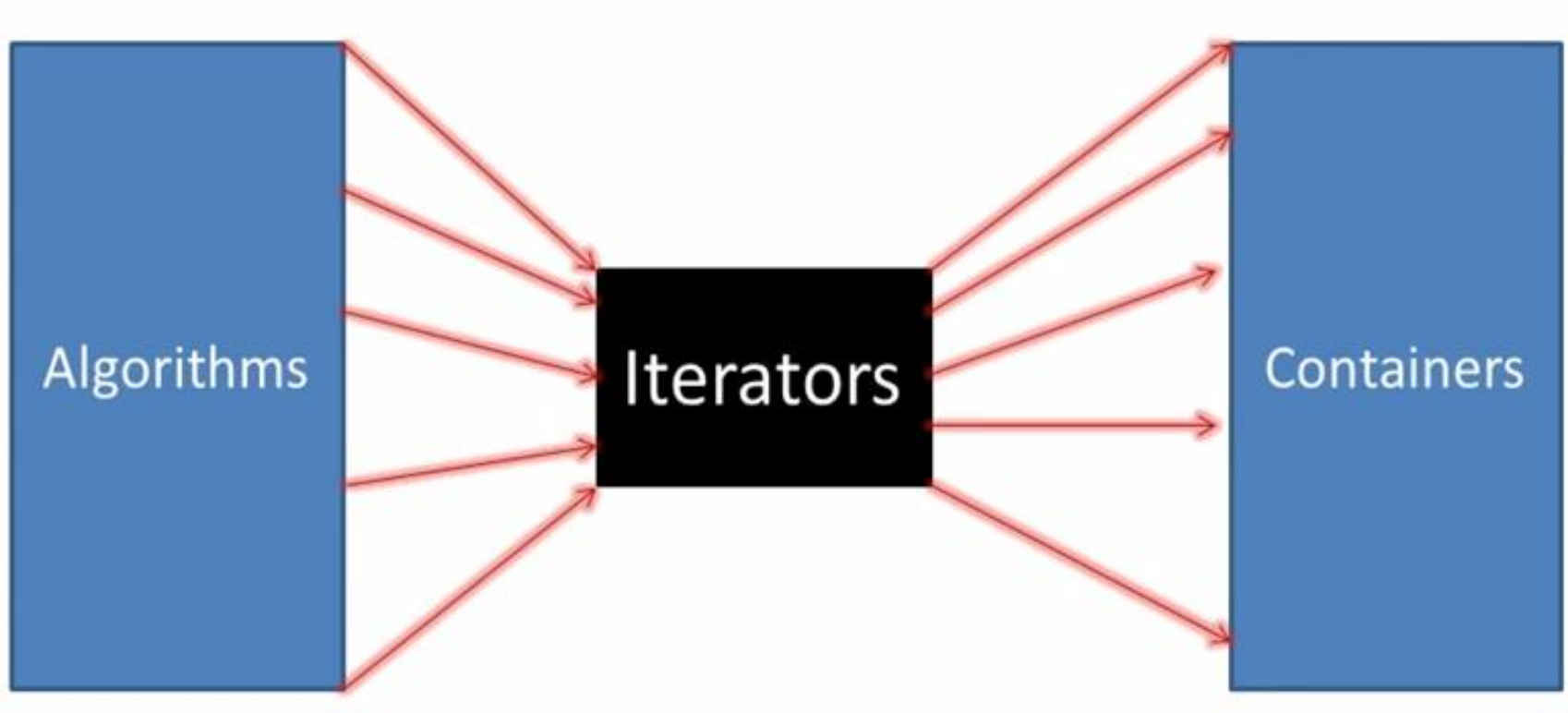
queue
stack
priority_queue

set
multiset
map
multimap

unordered variants

C++ STL

- Web recomendada: www.cppreference.com



Estructuras básicas

- Existen 3 tipos de datos básicos en la STL, que son usados internamente en la implementación de otros tipos de datos, como contenedores de sus datos:
  - Vector
  - List
  - Deque

- Todas las estructuras de datos en la STL están definidas como plantillas, con lo que podemos usar cualquier estructura de la STL con cualquier clase siempre que cumpla una serie de requisitos que dependen de la estructura.
- En el caso de vector, list y deque no hay ningún requisito especial.

Vector

- Es una plantilla que permite disponer de arreglos (contenedor secuencial) que pueden cambiar de tamaño.
- Métodos importantes:
  - size() -> devuelve el número de elementos
  - push_back(const value_type& val) -> insertar al final
  - pop_back() -> quitar último elemento
  - operator[] -> acceder elemento
  - clear() -> eliminar todos los elementos
  - reserve(size_type n) -> reservar para al menos n objetos
  - shrink_to_fit() -> liberar memoria no usada
  - insert(iterator position, const value_type &val) -> insertar antes de position
  - resize(size_type n, value_type val = value_type()) -> cambiar el tamaño
  - begin() -> iterador al primer elemento
  - end() -> iterador tras el último elemento
  - Múltiples constructores
- Fallo típico: size() devuelve un unsigned, cuidado con size()-1

Vector

```cpp
// constructing vectors
#include <iostream>
#include <vector>

int main ()
{
  // constructors used in the same order as described above:
  std::vector<int> first;                                // empty vector of ints
  std::vector<int> second (4,100);                       // four ints with value 100
  std::vector<int> third (second.begin(),second.end());  // iterating through second
  std::vector<int> fourth (third);                       // a copy of third

  // the iterator constructor can also be used to construct from arrays:
  int myints[] = {16,2,77,29};
  std::vector<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );

  std::cout << "The contents of fifth are:";
  for (std::vector<int>::iterator it = fifth.begin(); it != fifth.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Output:

```
The contents of fifth are: 16 2 77 29
```