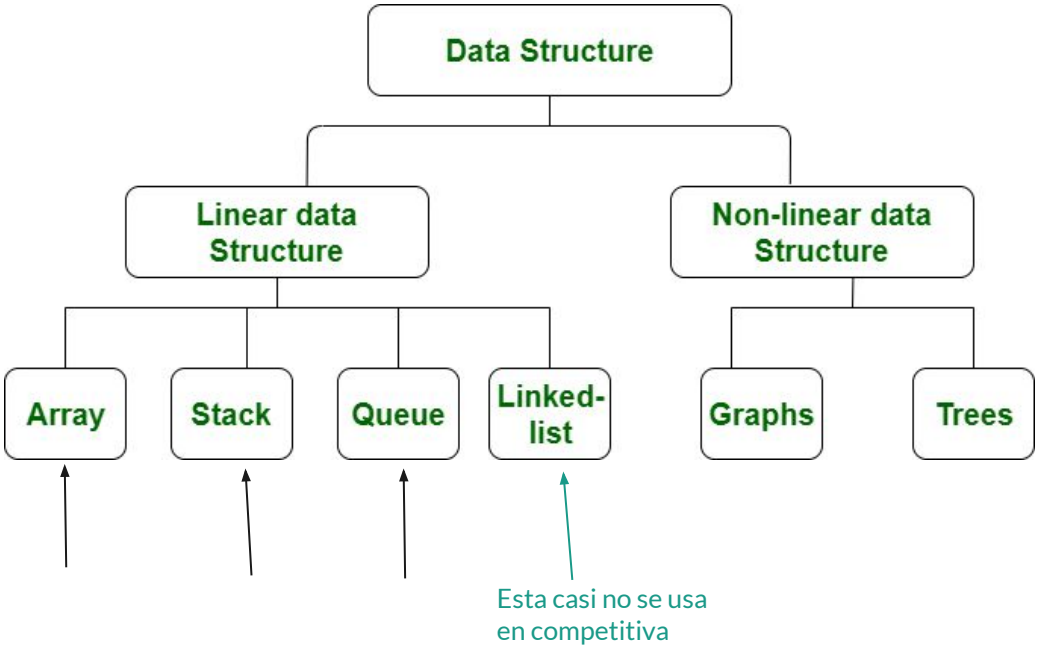
Clase 2 - C++ STL

Estructuras de Datos I

Preparación GPC-UPC
Adaptado de las diapositivas de Rodolfo Mercado.
Complementado por Oscar Burga.
Tutor: Oscar Burga

Por ver hoy

- Concepto de estructura de datos
- Estructuras de datos lineales (contenedores de secuencia)
- Stacks / Pilas
- Queues / Colas
- Deques / Colas Dobles (?)



Repaso: Iteradores

- Básicamente son punteros
- La gran mayoría de funciones y contenedores STL realizan sus operaciones mediante iteradores.
- De igual manera, muchas funciones STL reciben y retornan iteradores en vez de elementos.

Iterador al primer elemento del contenedor:	`contenedor.begin();`
Iterador a la primera posición fuera del contenedor:	`contenedor.end();`
Moverse por el contenedor:	`iterator++, iterator--;`
Acceder al valor apuntado por el iterador:	`*(iterator);`

Repaso: Funciones Principales - Vector y String

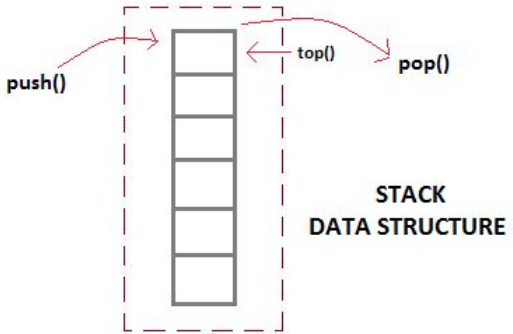
- **vec.size():** Retorna la cantidad de elementos en el contenedor. O(1)
- **vec.push_back(elem):** Agrega un elemento *elem* al final. O(1)
- **vec.pop_back()**: Elimina el elemento del final **(CUIDADO SI EL VECTOR ESTÁ VACÍO).** O(1)
- **vec.front():** Obtiene el primer elemento **(POR REFERENCIA)**. O(1)
- **vec.back():** Obtiene el último elemento **(POR REFERENCIA).** O(1)

Recuerdan que en la diapositiva de la clase pasada les puse "Luego verán que pop_back() también puede ser extremadamente útil"?

Bueno, ese momento ha llegado :)

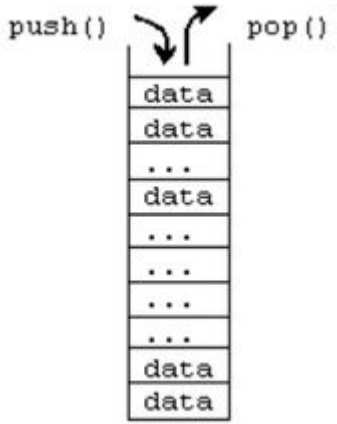
Estructuras de Datos

- Forma particular de almacenar y organizar datos.
- Útiles para manejar grandes cantidades de datos eficientemente.
- Realizar operaciones específicas de manera eficiente.
- Dos tipos generales:
  - Estructuras de datos lineales: Vector, Stack, etc. (Hoy)
  - Estructuras de datos logarítmicas: Map, Set, etc. (próximamente).



Contenedor: Stack (pila)

```
stack<double> p;
```



- Estructura de datos tipo LIFO (Last In - First Out).
- Last In - First Out: Último elemento colocado es el primero en salir.
- Comportamiento como el push_back() y pop_back() (se llaman solo push() y pop())
- Insertar y remover del final en O(1).
- No es iterable :(Las pilas no tienen iteradores, solo se puede acceder al último elemento con el método top().

Otros contenedores como vector y deque soportan las operaciones fundamentales de una pila, por lo que se les puede usar para simular una sin problemas.

Stack: Paréntesis Balanceados

Dado una cadena compuesta de paréntesis, es decir ‘(‘ y ‘)’ , se dice que los paréntesis están balanceados si cada paréntesis de apertura tiene un correspondiente paréntesis de cierre y los pares de paréntesis están correctamente anidados.

Ejemplos:

( ) si

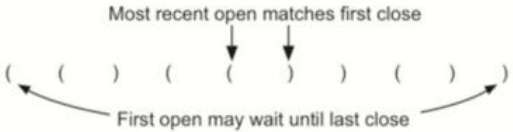
(( ) ( )) si

(( )) (no

Stack: Paréntesis balanceados

Pista:

Si procesamos los paréntesis de izquierda a derecha:



- El más reciente de apertura debe hacer match con el más próximo de cierre.
- El primer paréntesis de apertura puede esperar hasta el último de cierre.
- Todos los prefijos de la cadena de paréntesis deben tener mayor o igual cantidad de paréntesis de apertura que de cierre.

Determinar si una cadena de paréntesis es balanceada en O(n)

Hora de mostrar las habilidades de paint

Contenedor: Queue (cola)

```
queue<int> q;
```

- Estructura de datos tipo FIFO.
- First In - First Out: Primer elemento colocado es el primero en salir
- Comportamiento como push_back() (se llama solo push())
- Nuevo comportamiento: pop_front() ?!? (se llama solo pop())
- Insertar al final y remover del inicio en O(1).
- No es iterable :(Las colas no tienen iteradores, solo se puede acceder al elemento del inicio con el método front().

Las colas, aunque son la base de algunos algoritmos y técnicas importantes, aunque por sí solas no suelen aparecer muy seguido. Muchas veces, cuando necesitamos soportar este tipo de operaciones, necesitamos **insertar y remover** por ambos lados eficientemente...

Contenedor: Deque

```
deque<int> dq;
```

- Lo mejor de ambos mundos, y más.
- Permite acceso aleatorio y cambio de tamaño en tiempo de ejecución (como vector).
- Permite insertar y eliminar un elemento al inicio y al final, ambas en tiempo constante.
- Permite simular pilas y colas.
- Es iterable!!! (tiene iteradores, puedes usar las funciones STL como si fuera un vector).
- Métodos nuevos:
  - dq.push_front(elem): Insertar el elemento *elem* al inicio O(1)
  - dq.push_back(elem): Insertar el elemento *elem* al final O(1)
  - dq.pop_front(): Eliminar el elemento en la posición inicial O(1)
  - dq.pop_back(): Eliminar el elemento en la última posición O(1)

```
deque< int > dq ; // declarar deque
dq.size(); // tamaño del deque O(1)
dq[ i ]; // accesos O(1)
dq.push_back( x ); // agregar un elemento al final  O(1)
dq.push_front( x ); // agregar un elemento al inicio O(1)
dq.pop_back(); // eliminar el último elemento  O(1)
dq.pop_front(); // eliminar el primer elemento O(1)
dq.front(); // obtener primer elemento O(1)
dq.back(); // obtener el último elemento O(1)

// insertar elemento x en posición pos O(n)
deque<int>::iterator it;
it = dq.begin() + pos ;
dq.insert ( it , x);

// eliminar elemento en posicion pos O(n)
dq.erase (v.begin() + pos );
```

Problemitas de práctica

Tercer Contest Semanal