ESTRUCTURAS DE DATOS

UNIVERSIDAD EAFIT

Tipos de datos

short	$-2^{15} <= x < 2^{15}$
int	$-2^{31} <= x < 2^{31}$
long long	$-2^{63} <= x < 2^{63}$
char	ASCII
bool	{0,1} o {false, true}
double	$1.7*10^{-308} <= x <= 1.7*10^{308}$
__int128_t (con ciertos compiladores)	$-2^{127} <= x < 2^{127}$

Estructuras libreria estandar

- Arreglos estáticos.
- Arreglos dinámicos. (vector)
- Pila.
- Cola.
- set - multiset - unordered_set.
- map-unordered_map (hashtable)

Arreglos

```cpp
int N = 5;
int arreglo[N]; // definir un arreglo de N enteros

arreglo[0] = 2; // asignar un valor en una posicion

arreglo[1] = arreglo[0] * 3; // acceder a una posicion

int target = 6;
for(int i = 0; i < N; i++){ // buscar el numero target en el arreglo
  if(arreglo[i]==target){
    cout<< "Encontre el numero en la posicion "<< i << "\n";
  }
}
```

Insertar un elemento	No se puede por ser estático
Cambiar el valor de una posición existente	O(1)
Preguntar por el valor de una posición	O(1)
Chequear si existe un valor en el arreglo	O(N)
Borrar un elemento	No se puede por ser estático.

Vectores

```cpp
vector<int> vec(N); // definir un vector de N enteros
vec[1] = 30; // asignar un valor en una posicion


vec[3] = 2*vec[1] + 4; // acceder a una posicion

vec.push_back(3); // insertar al final

vec.insert(vec.begin() + 2, 8);  // insertar en la posicion 2 el numero 8

vec.pop_back(); // eliminar del final

vec.erase(vec.begin() + 3); // eliminar la posicion 3
target = 8;
for(int i = 0; i < vec.size(); i++){ // buscar el numero target en el arreglo
  if(vec[i] == target){
    cout<< "Encontre el numero en la posicion "<< i << "\n";
  }
}
vec.clear(); // eliminar todos los elementos O(vec.size())
```

Insertar un elemento al final del arreglo	O(1) aprox.
Insertar un elemento en cualquier posición	O(N)
Cambiar el valor de una posición existente	O(1)
Preguntar por el valor de una posición	O(1)
Chequear si existe un valor en el arreglo	O(N)
Borrar un elemento del final del arreglo	O(1)
Borrar un elemento en cualquier posición	O(N)

String

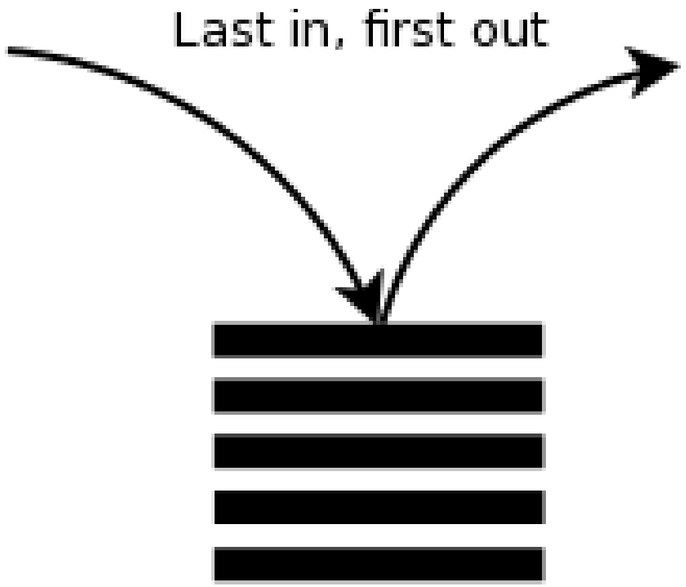
```
string cadena = "hola"; // definir un string

cadena.push_back('2'); // insertar al final

int posicion_inicial = 2;
int longitud_substring = 2;
cadena.substr(posicion_inicial, longitud_substring); // devuelve "la"
cout<< cadena.back() << "\n"; // imprime '2'
```

Funciones utiles de arreglos estaticos y dinamicos

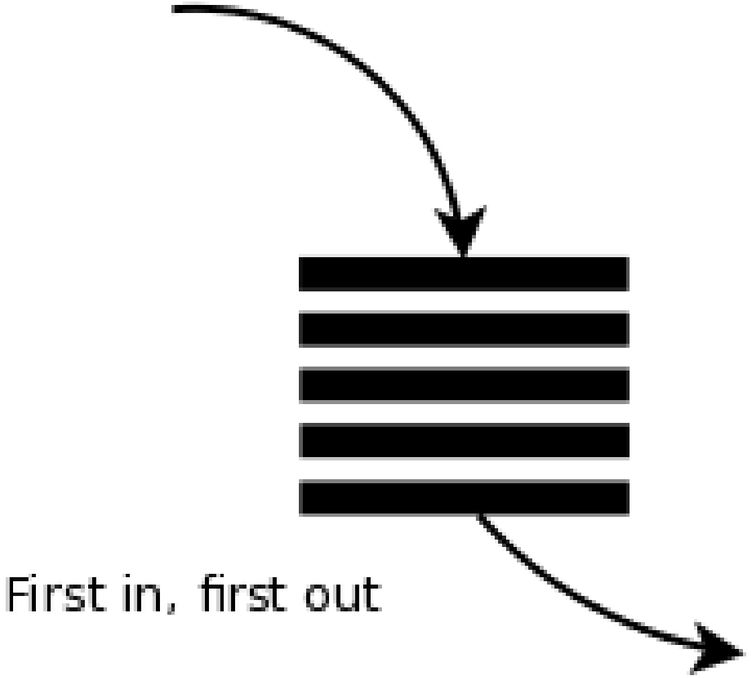
```cpp
// Operaciones utiles en arreglos dinamicos y estaticos

// ordenar los elementos de menor a mayor
sort(arreglo, arreglo + N);
sort(vec.begin(), vec.end());

// reversar los elementos
reverse(arreglo, arreglo + N);
reverse(vec.begin(), vec.end());

int min_pos = min_element(vec.begin(), vec.end())-vec.begin(); // posicion minimo elemento
int max_pos = max_element(arreglo, arreglo + N) - arreglo; // posicion maximo elemento

// solo usar si el vector esta ordenado
int pos = lower_bound(vec.begin(), vec.end(), x) - vec.begin();
// me devuelve la posicion del elemento mayor o igual a x en el vector
// en algunos casos puede retornar la posicion vec.size(), indicando que no hay ningun
// que cumpla que sea mayor o igual a x.
```

Stack - Queue

Stack:



Queue:



Pila

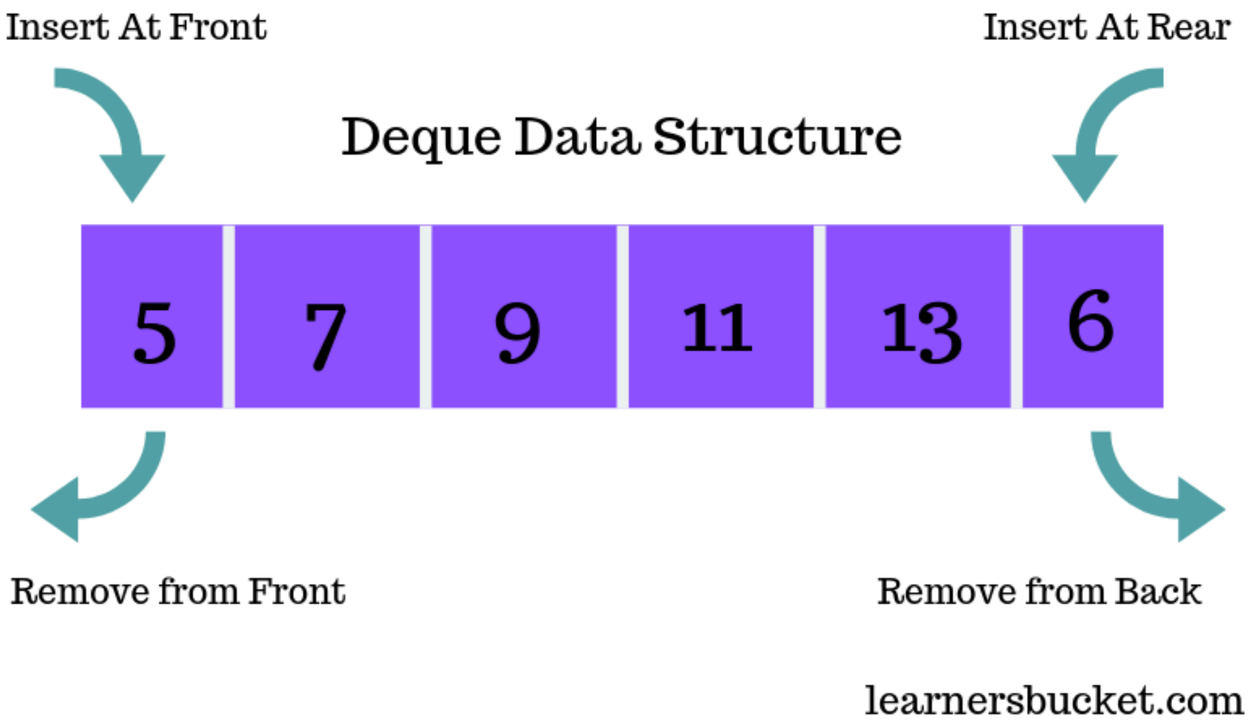
```
stack<int> pila; // declarar la estructura de datos

pila.push(2); // agregar elementos  pila = [2 ]
pila.push(6); // agregar elementos pila = [2, 6]
pila.push(1); // agregar elementos pila = [2, 6, 1]

pila.top(); // retorna elemento que esta arriba

pila.pop(); // remover el elemento que esta arriba

pila.empty(); // verdadero o falso

pila.size(); // retorna el numero de elementos en la pila, 3 en este caso
```

Cola

```
queue<char> cola; // declarar la estructura de datos

cola.push('a'); // agregar elementos  cola = ['a' ]
cola.push('x'); // agregar elementos cola = ['a', 'x']
cola.push('c'); // agregar elementos cola = ['a', 'x', 'c']

cola.front(); // retorna elemento que esta al frente 'a'

cola.pop(); // remover el elemento que esta al frente

cola.empty(); // verdadero o falso

cola.size(); // retorna el numero de elementos en la cola, 3 en este caso
```

Operación	Pila	Cola
Insertar un elemento al final	O(1)	O(1)
Sacar un elemento del final	O(1)	X
Sacar un elemento del inicio	X	O(1)

Deque


Insert At Front
Insert At Rear
Deque Data Structure
5
7
9
11
13
6
Remove from Front
Remove from Back
learnersbucket.com

Deque

```
deque<char> deq; // declarar la estructura de datos

deq.push_back('a'); // agregar elementos  deq = ['a' ]
deq.push_back('x'); // agregar elementos deq = ['a', 'x']
deq.push_front('c'); // agregar elementos deq = ['c', 'a', 'x']

deq.front(); // retorna elemento que esta al frente 'c'
deq.back(); // retorna elemento que esta atras 'x'

deq.pop_front(); // remover el elemento que esta al frente
deq.pop_back(); // remover el elemento que esta atras

deq.empty(); // verdadero o falso

deq.size(); // retorna el numero de elementos en la deq, 3 en este caso
```

Priority queue

Priority queue

```cpp
priority_queue<int> pq; // declarar la estructura de datos

pq.push(4); // agregar elementos  pq = [4 ]
pq.push(3); // agregar elementos pq = [4, 3]
pq.push(7); // agregar elementos pq = [7, 4, 3]

pq.top(); // retorna elemento maximo 7

pq.pop(); // remover el elemento maximo

pq.empty(); // verdadero o falso

pq.size(); // retorna el numero de elementos en la pq
```

Set - Multiset - Ordered Set

- **Set:** estructura ordenada que tiene en su interior en todo momento elementos únicos.

- **Multiset:** estructura ordenada parecida al set pero que permite elementos duplicados.

- **Unordered_set**: estructura no ordenada que tiene en su interior en todo momento elementos únicos.

Set

```cpp
set<int> s; // declarar la estructura de datos

s.insert(4); // agregar elementos s = [4 ]
s.insert(4); // agregar elementos s = [4 ]
s.insert(3); // agregar elementos s = [3, 4]
s.insert(7); // agregar elementos s = [3, 4, 7]

(*s.begin()); // retorna el elemento minimo 3

(*s.rbegin()); // retorna el elemento maximo 7

s.erase(3); // remover el elemento que le entrego

s.count(3); // retorna 1 o 0 dependiendo si el elemento esta o no

s.size(); // retorna el numero de elementos en el set
```

Set

```
// retorna el numero >= al que le paso
s.lower_bound(5); // retorna 7
s.lower_bound(7); // retorna 7

// retorna el numero > al que le paso
s.upper_bound(5); // retorna 7
s.upper_bound(7); // retorna s.end() apuntando al final del set
```

MultiSet

```cpp
multiset<int> s; // declarar la estructura de datos

s.insert(4); // agregar elementos s = [4 ]
s.insert(4); // agregar elementos s = [4, 4]
s.insert(3); // agregar elementos s = [3, 4, 4]
s.insert(7); // agregar elementos s = [3, 4, 4, 7]

(*s.begin()); // retorna el elemento minimo 3

(*s.rbegin()); // retorna el elemento maximo 7

s.erase(4); // elimina todas las ocurrencias del numero 4
s.erase(s.find(4)); // elimina una de las ocurrencias del numero 4

s.count(3); // retorna la cantidad de veces que aparece el numero

s.size(); // retorna el numero de elementos en el set
```

Unordered_Set

```cpp
unordered_set<int> s; // declarar la estructura de datos

s.insert(4); // agregar elementos s = [4 ]
s.insert(4); // agregar elementos s = [4]
s.insert(3); // agregar elementos s = [3, 4]
s.insert(7); // agregar elementos s = [3, 4, 7]

s.erase(4); // elimina el numero 4

s.count(3); // retorna la cantidad de veces que aparece el numero 3

s.size(); // retorna el numero de elementos en el set
```

Iterar sobre los elementos

```
for(int x : s){
  cout << "El elemento " << x << " esta en la estructura \n";
}
```

Operación	set	multiset	unordered_set
Insertar un elemento	O(log2(N))	O(log2(N))	O(1)
Eliminar un elemento	O(log2(N))	O(log2(N))	O(1)
Chequear la existencia de un valor	O(log2(N))	O(log2(N))	O(1)

Map - Unordered map

- **map:** estructura ordenada que permite guardar información almacenar parejas en el formato: clave, valor.

- **unordered_map:** igual que el map pero no ordenado.

Map

```cpp
map<string, int> edades;
edades["Andrea"] = 23;
edades["Pedro"] = 29;
edades["Henry"] = 20;

for(pair<string, int> elemento : edades){ // iterar sobre el mapa
  cout << elemento.first << " tiene " << elemento.second << " años\n";
}

edades.count("Henry"); // devuelve si aparece el elemento en el mapa

edades.begin(); // puntero a la primera posicion
```

Unordered_Map

```cpp
unordered_map<string, int> edades;
edades["Andrea"] = 23;
edades["Pedro"] = 29;
edades["Henry"] = 20;

for(pair<string, int> elemento : edades){ // iterar sobre el mapa
  cout << elemento.first << " tiene " << elemento.second << " años\n";
}

edades.count("Henry"); // devuelve si aparece el elemento en el mapa
```

Operación	**map**	**unordered_map**
Insertar un elemento	O(log2(N))	O(1)
Eliminar un elemento	O(log2(N))	O(1)
Chequear la existencia de una clave	O(log2(N))	O(1)
Cambiar el valor de una clave	O(log2(N))	O(1)