Universidad Católica del Norte

ESCUELA DE INGENIERÍA UCN COQUIMBO

Tipos de Datos Abstractos (TDAs) y Estructuras De Datos (EDDs)

Dr. Juan Bekios Calfa

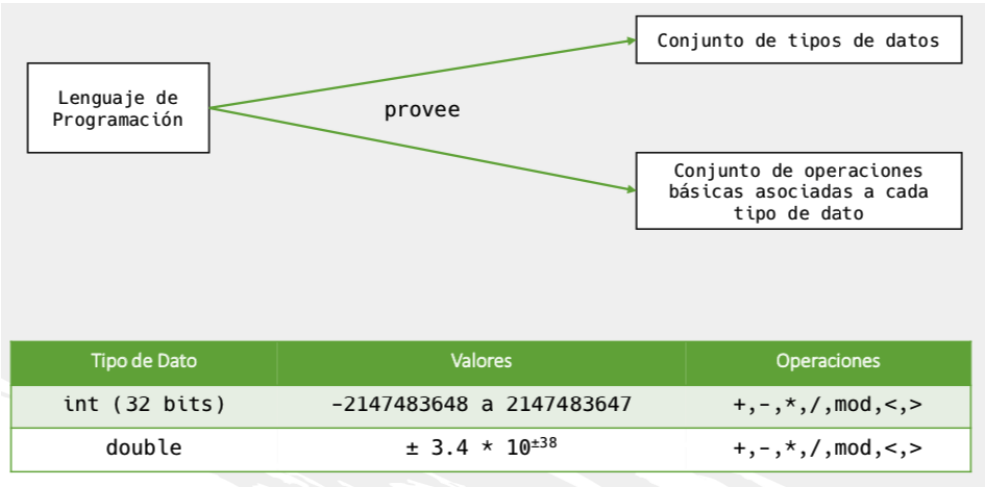
juan.bekios@ucn.cl

http://jbekios.ucn.cl

Datos Primitivos

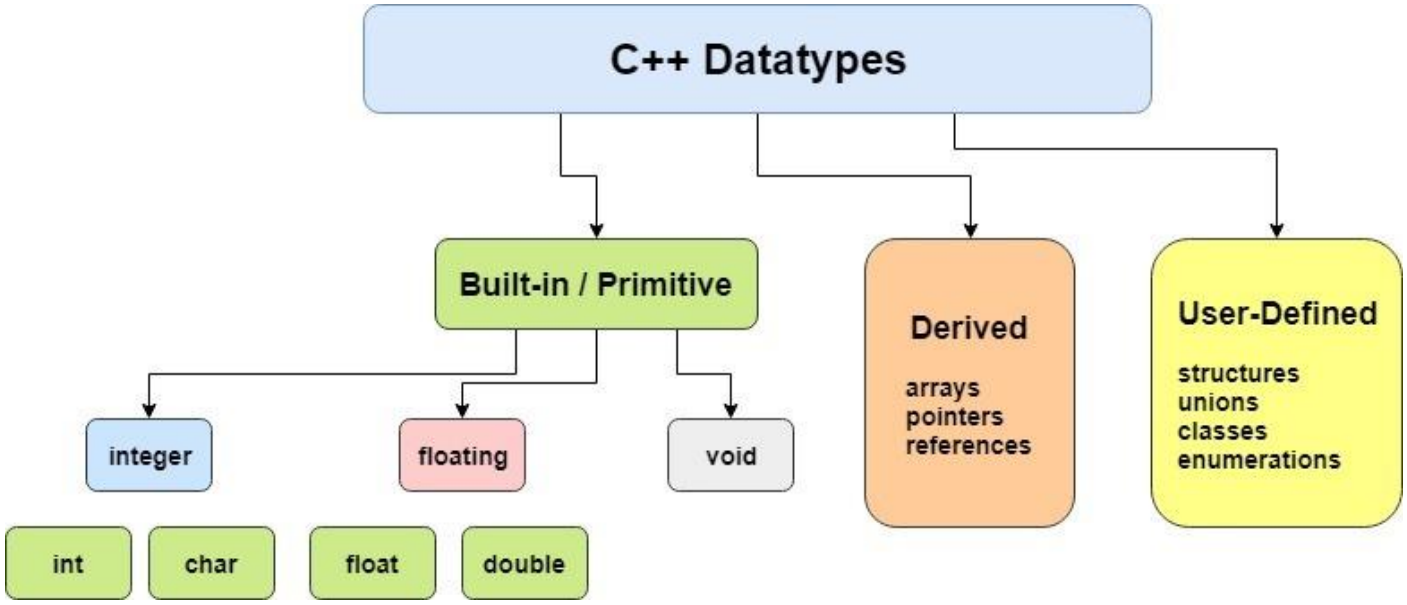
- Un dato representa **un hecho** ya "ocurrido", por ejemplo: edad, nombre, número de teléfono.
- Un dato tiene generalmente asociado un *Tipo* "int", "float", "double", "char", etc.
- Un **lenguaje de programación** contiene un **conjunto de datos primitivos** y **operaciones** sobre ellos.

Datos Primitivos



Tipo de Dato	Valores	Operaciones
int (32 bits)	-2147483648 a 2147483647	+,-,*,/,mod,<,>
double	$\pm$ 3.4 * $10^{\pm 38}$	+,-,*,/,mod,<,>

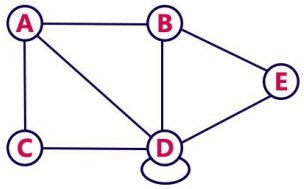
Tipos de datos en C++



https://aktuupdate.com/wp-content/uploads/2016/11/c-datatypes.jpg

Un ejemplo con datos derivados/primitivos: *Matrices Pocos Pobladas*

- Considere que quiere representar un grafo (ej. rutas entre ciudades) con una **matriz de enteros.**
- En la matriz, podemos colocar un 1 cuando dos nodos se conectan y 0 en caso contrario.
- Existe varios espacios vacíos (es decir, 0), por ello, se vuelve ineficiente. De hecho (si la ruta es bilateral), la diagonal inferior es completamente innecesaria.



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	1	1
E	0	1	0	1	0

Los datos primitivos no son siempre eficientes y intuitivos por los conceptos que desean representan

La diagonal inferior es igual a la diagonal superior

Tipos de Datos Abstractos (TDAs)

- La información siempre se vuelve más compleja administrarla y representarla (ej. persona, auto, alumno). A nivel teórico, se han creados varios modelos para representarlos como el modelo entidad-relación (Base de Datos), etc.

- En la teoría de los lenguajes de programación, se está consciente de la complejidad de representar datos, por ello, se creó el **concepto de Tipos de Datos Abstractos** (TDAs)

Tipos de Datos Abstractos (TDAs)

- Cada TDA tiene asociados un **conjunto de operaciones**
- TDAs especifican la **semántica del comportamiento**, pero no necesariamente su implementación
- Las operaciones son definidas en **términos de su semántica**

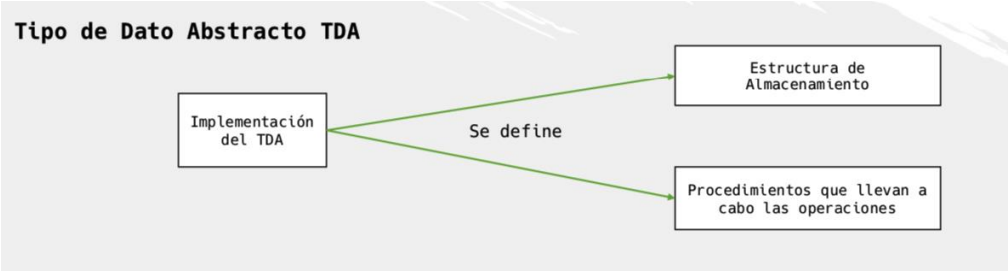
Conjunto: Un TDA Simple

- Un conjunto es un grupo de elementos (del mismo tipo) **no tienen un orden definido** y **no hay repetidos**
- Operaciones son del tipo **Unión**, **Intersección**, **Diferencia**, **Pertenencia**, **Inserción**, **Borrado** y **Creación de vacío**. Por ejemplo:

C = A U B ó C = A.union(B)

Estructura de Datos

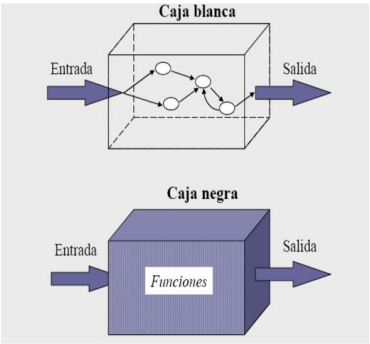
Las estructuras de datos son implementaciones concretas de los TDAs



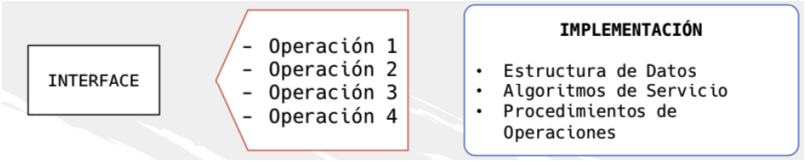
TDA es la definición conceptual sobre un dato no primitivo. No habla de su implementación

Estructura de Datos es una implementación concreta de una estructura de datos

Estructura de Datos

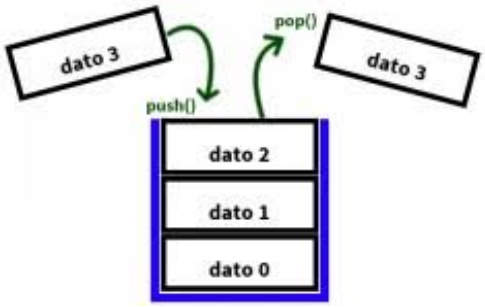


Una **Estructura de Datos** tiene una Interface (API, *Application Programming Interface*) que sigue su TDA e implementación ¿Recuerdan los principios de black-box (y white-box)?



¿Qué operación podría existir en un conjunto que fuese parte de la interface?

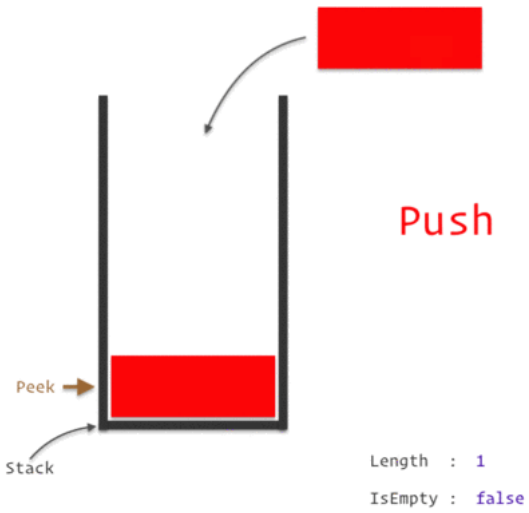
Pila (*Stack*): Un conocido TDA



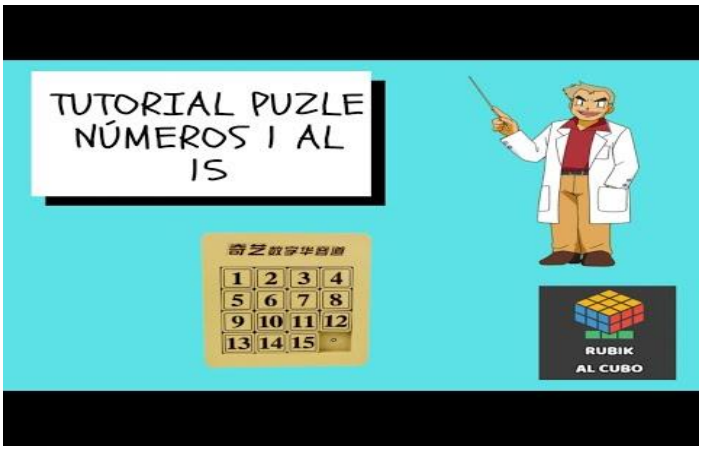
Una Pila es un TDA donde se ingresa y se saca del final, es decir, ***L**ast **I**n **F**irst **O**ut* (LIFO)

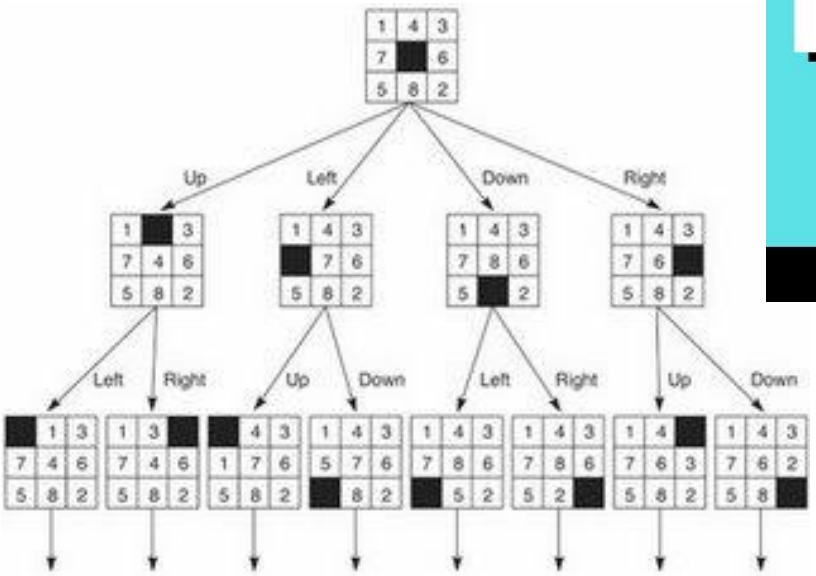
Operaciones: push(x), **pop()**, empty(), size(), create()

Animación de la Pila

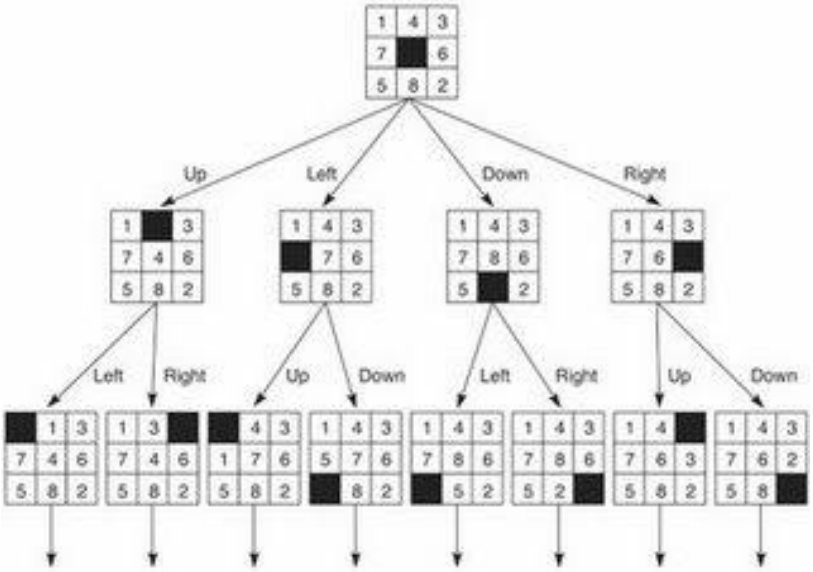


Ejemplo: Pila (*Stack*)





Ejemplo: Pila (*Stack*)



Estructura de Datos: Implementación de una Pila con struct

```
const int MAX = 10;

struct Stack {
  int size;
  int elements[MAX];
};

typedef struct Stack Stack;

Stack* create() {
   return new Stack();
}

void push(Stack *s, int x) {
  s->elements[s->size++] = x;
}

int pop(Stack *s) {
  return s->elements[--(s->size)];
}

int empty(Stack *s) {
  return s->size == 0;
}

int size(Stack *s) {
  return s->size;
}

void show(Stack *s) {
  for (int i = 0; i < s->size; ++i) {
    printf("%d ", s->elements[i]);
  }
  printf("\n");
}
```

```
int main() {
  Stack *s = create();
  push(s,5);
  push(s,2);
  push(s,8);
  push(s,15);
  show(s);

  printf("pop: %d \n", pop(s));

  show(s);

  print ("esta vacia? %d \n", empty(s));

  return 0;
}
```

```
5 2 8 15
pop: 15
5 2 8
esta vacia?0
```

https://repl.it/@PaulLeger/clase61

¿Limitante con esta implementación del TDA PILA?

Implementando create, push, pop, empty, size, show

Variación con tamaño dinámico

```
struct Stack {
  int size;
  int* elements;
};

typedef struct Stack Stack;

Stack* create(int max) {
   Stack *s = new Stack();
   s->elements = new int[max];
   return s;
}
```

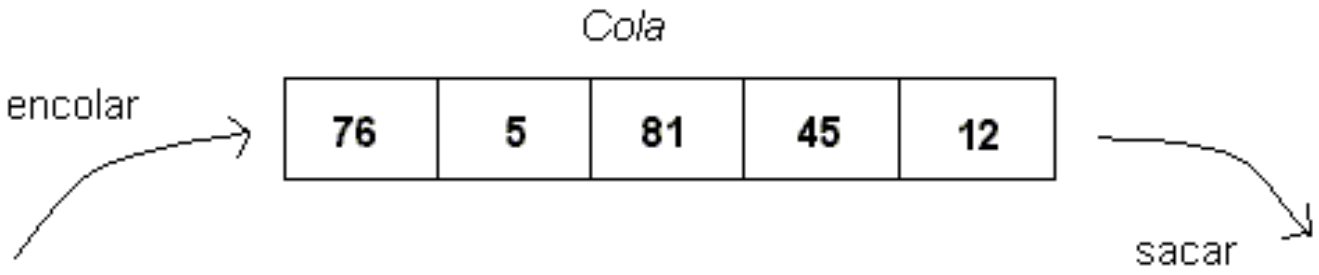
```
void destroy(Stack *s) {
  delete[] s->elements;
}
```

Ahora el tamaño se puede definir al crearse

¿Limitante con esta implementación del TDA-PILA?

https://repl.it/@PaulLeger/clase62

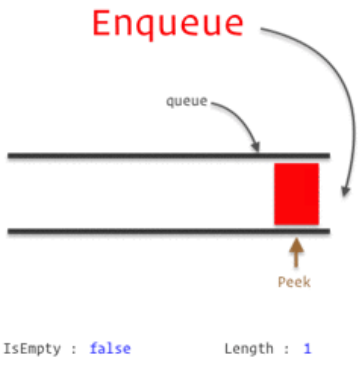
Cola (Queue): Otro conocido TDA



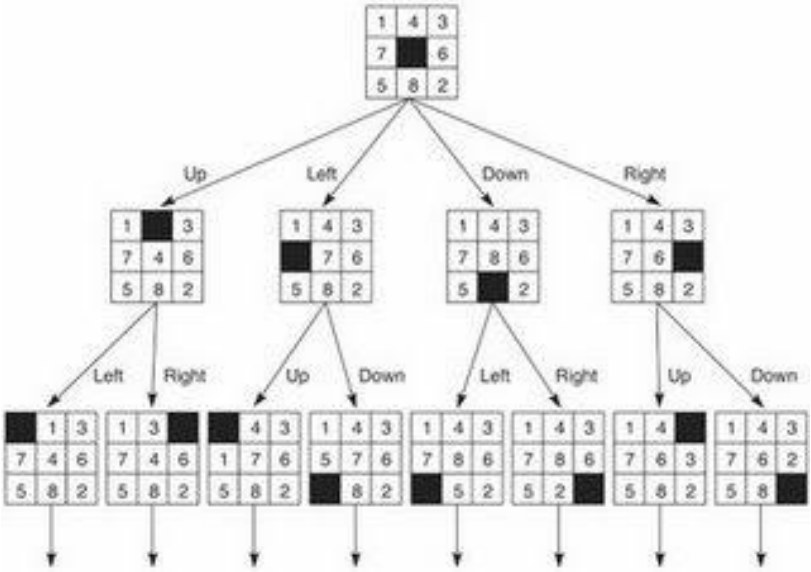
Una Cola es un TDA donde se ingresa al final y se saca al principio, es decir, First In First Out (FIFO)

Operaciones: push(x), **dequeue()**, empty(), size(), create()

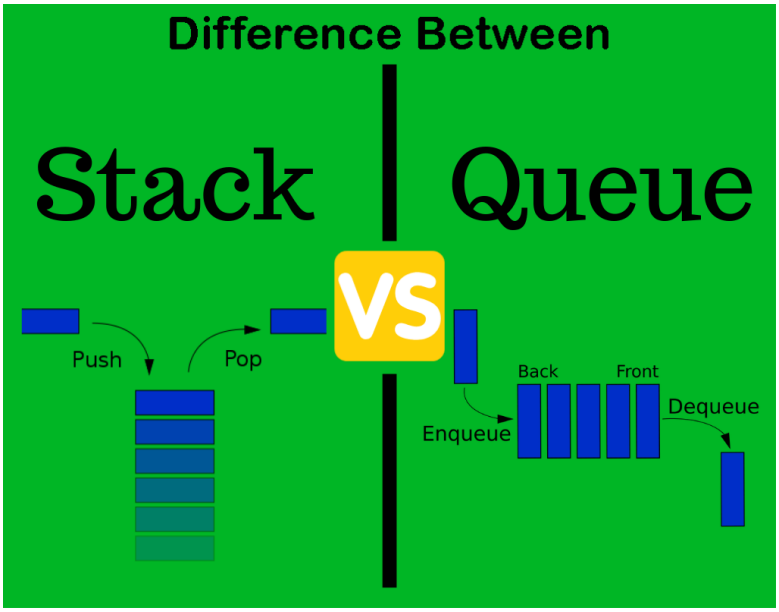
Animación de una cola



Ejemplo: Cola (Queue)



Stack vs *Queue*



Estructura de Datos: Implementación de una Cola con struct y memoria dinámica

```
3  struct Queue {
4    int size;
5    int* elements;
6  };
7
8  typedef struct Queue Queue;
9
10 Queue* create(int max) {
11   Queue *q = new Queue();
12   q->elements = new int[max];
13   return q;
14 }
15
16 void push(Queue *q, int x) {
17   q->elements[q->size++] = x;
18 }
19
20 int dequeue(Queue *q) {
21   int result = q->elements[0];
22   for (int i = 0; i < q->size-1; ++i) {
23     q->elements[i] = q->elements[i+1];
24   }
25   q->size--;
26   return result;
27 }
28
29 int empty(Queue *q) {
30   return q->size == 0;
31 }
32
33 int size(Queue *q) {
34   return q->size;
35 }
36
37 void show(Queue *q) {
38   for (int i = 0; i < q->size; ++i) {
39     printf("%d ", q->elements[i]);
40   }
41   printf("\n");
42 }
43
44 void destroy(Queue *q) {
45   delete[] q->elements;
46 }
```

```
int main() {
  Queue *q = create(10);
  push(q,5);
  push(q,2);
  push(q,8);
  push(q,15);
  show(q);

  printf("dequeue: %d \n", dequeue(q));

  show(q);

  printf("esta vacia? %d \n", empty(q));


  destroy(q);
  delete q;

  return 0;
}
```

https://repl.it/@PaulLeger/clase64

Estructura de Datos: Implementación de una Cola con struct y memoria dinámica

q= create(10)

Push(q, 5)

Push(q, 10)

Push(q , 15)

Dequeue(q)

Ejercicios

- Implemente una pila con tamaño dinámico en orientación a objetos, esta pila debe contener personas (nombre y edad)

https://repl.it/@PaulLeger/pila-oop#main.cpp

- **[Propuesto]** Implemente una cola con tamaño dinámico en orientación a objetos

- **[Propuesto]** Implemente una cola en OOP que contenga una pila en cada elemento