75.40 Algoritmos y Programación I Curso 4
Tipo de Dato Abstracto

Dr. Mariano Méndez[1]

[1]Facultad De Ingeniería. Universidad de Buenos Aires

23 de agosto de 2022

1. Introducción

Esta sección está dedicada a unos de los conceptos más importantes en lo que respecta al ámbito de la informática y en este caso más precisamente a de la programación, ese concepto es la Abstracción. La Abstracción es el proceso por el cual se despoja a un problema o cosa de la complejidad que no es relevante para la solución de un problema determinado. Los lenguajes de programación de alto nivel como por ejemplo C, C++, Python, Java por nombrar algunos poseen un conjunto de herramientas que hacen extensible al lenguaje para construir nuevas abstracciones según los programadores lo requieran [**?**].

Cuando se habla de abstracción lo que se espera "es un mecanismo que permita la expresión los detalles relevantes y la supresión de los detalles irrelevantes" [**?**]. Siguiendo esta afirmación, en lo que respecta a la programación lo que respecta al uso del qué va a ser realizado por la abstracción es relevante, y por otro lado el cómo va a ser realizado es irrelevante.

Específicamente para el lenguaje de programación C las herramientas que hacen extensible al lenguaje con nuevas abstracciones son:

- funciones y procedimientos
- archivos de encabezado
- tipos de datos primitivos del lenguaje

1.1. La Abstracción

Abstracción proviene del latín abstracto y está vinculado al verbo abstraer que significa separar aisladamente en la mente las características de un objeto o un hecho, dejando de prestar atención al mundo sensible para enfocarse solo en el pensamiento.

De acuerdo con la matemática, la abstracción es el proceso intelectual a través del cual separamos mentalmente las cualidades particulares de varios objetos para fijarnos únicamente en uno o diversas características comunes. Es a través del rigor, que se logra esta operación mental denominada generalización simple.

Una de las definiciones y usos de abstracción se puede ver en el siguiente chiste: Un biólogo, un matemático y un físico discuten del motivo por el cual el guepardo es tan veloz. El Matemático dice: -Pff es trivial la explicación, si tomamos la función guepardo(x) y la integramos entre el hocico y la cola, se explica perfectamento por qué el guepardo se mueve con esa velocidad... -Que disparate. Dice el Biólogo: - Esto se explica por los miles de años de evolución, por la teoría darwinista de la especialización de las especies, la adaptación del aparato muscular, bla bla bla.. El físico que observaba callado, dice: -Si tomamos al guepardo puntual ....

1.2. Los Tipos de Datos

Un tipo de dato define dos cosas importantes:

1. Al conjunto de todos los valores posibles que una variable de ese tipo puede tomar.
2. Las operaciones que las variables de ese tipo de dato pueden utilizar.

Es normal que los lenguajes de programación tengan *tipos primitivos (predefinidos) de datos*. Un tipo de dato es primitivo cuando no debe especificarse información extra para definir una variable de ese tipo o sobre las operaciones que pueden hacerse sobre las variables. Por ejemplo, en el lenguaje C el tipo de dato *int* o entero:

Tipo	Conjunto de Valores	Operador	Operación	Resultado
int	-2^{31} y $(2^{31}-1)$ implementado como complemento a 2	+ (suma)	$5+3$	8
		- (resta)	$5-3$	2
		* (multiplicación)	$5*3$	15
		/ (división entera)	$5/3$	1
		% (resto)	$5\,\%3$	2

1.3. ¿Qué es un Tipo de Dato Abstracto?

Por un lado, cuando un programador utiliza una función, a este debería importarle únicamente en qué es lo que hace la función – por ejemplo, cuando se utiliza printf nadie se pregunta cómo está implementada, sino uno se centra en que esta imprime un valor determinado por pantalla– es decir en lo que esta le provee, es decir, lo importante de la abstracción es el uso que se hace de ella y no como la abstracción se implementa.

Por otro lado las funciones y procedimiento permiten descomponer los problemas en sub-problemas, haciendo que partes de las áreas se ralicen en las distintas funciones que se llaman unas a otras y desde el programa principal. Esta herramienta apunta a capturar el concepto de abstracción pero no es suficiente por sí sola.

Un **Tipo de Dato Abstracto (TDA)**, según Liskov y Zilles, define una clase de objetos abstractos los cuales están completamente caracterizados por las operaciones que pueden realizarse sobre esos objetos. Esto quiere decir que un tipo de dato abstracto puede ser definido describiendo las operaciones características para ese tipo. En otras palabras, un tipo de dato abstracto está definido no sólo por una**estructura de datos** sino también por las **operaciones** que se define sobre esa estructura, es decir que son ambas cosas juntas.

Cuando un programador hace uso de un Tipo de Dato Abstracto (TDA), este se interesa únicamente en el comportamiento que tal objeto exhibe y no en los infinitos detalles de cómo se llegó a su Implementación. Más precisamente la visión de aquel que utiliza un TDA es exclusivamente de **Caja Negra**. El comportamiento de un TDA es capturado por el conjunto operaciones características o primitivas. Cualquier detalle de Implementación sobre el TDA es innecesario para aquel que hace uso del TDA.

Los TDA deben tender a parecerse a un tipo de dato primitivo provisto por el lenguaje de programación. Uno usa variables de tipo entero y no se pregunta como la computadora implementa internamente ese tipo, utiliza sus operaciones características o primitivas como la suma, división entera, resto, multiplicación, resta y resto de la división entera. En el caso de los TDAs el programador (usuario) está haciendo uso del concepto o abstracción que logra el TDA en un determinado nivel, pero no de los detalles de bajo nivel de cómo se realizan (implementación del complemento a 2).

Una importante implicación del uso de un tipo de datos abstracto, es que la mayoría (por no decir todas) las operaciones que se utilizan en un programa de un determinado tda pertenecen al set de operaciones características o primitivas del mismo.

2. El Qué y el Cómo

Las visiones de caja negra y caja blanca, hacen una importante diferencia:

- El **Qué**: cuando se habla del qué se está haciendo referencia al concepto de funcionalidad. La funcionalidad está estrechamente relacionado a la pregunta ¿Qué hace esto? Es el concepto de **caja negra**.
- El **Cómo**: cuando se hace referencia al cómo se está hablando de la forma en que algo está diseñado o implementado. Cuando se habla de diseño o implementación se está haciendo referencia a la estructura interna o a la forma en la cual la funcionalidad de algo es llevada a cabo. Por ello está estrechamente ligado a la pregunta de ¿como lo hago? Concepto de **caja blanca**.

En el estudio e implementación de los tda poder separar correctamente el **qué** del **cómo** es fundamental. Por ejemplo, un string es un tipo de dato muy común en los lenguajes de programación de alto nivel. Cuando se pregunta **qué** es un string, una respuesta acertada seria: es un tipo de dato en el cual se pueden almacenar palabras, frases o textos.

Cuando se pregunta de **cómo** se implementa el tipo de dato string, las respuesta pueden ser muy variadas. Existen diversas implementaciones. Por ejemplo en el lenguaje de programación C un string es una arreglo de caracteres cuyo contenido válido está delimitado por un \0 , ver figura 1. En Pascal un string es un arreglo de caracteres cuya contenido válido está informado en la posición 0, ver figura 2. Otra forma posible de implementar un string seria con un registro de dos campos uno que posea una cadena de caracteres y el otro la longitud, ver figura 1:

H	O	L	A	∅	M	U	N	D	O	\0

Figura 1: Implementación de un String en C

10	H	O	L	A	∅	M	U	N	D	O

Figura 2: Implementación de un String en Pascal

H	O	L	A	∅	M	U	N	D	O
10									

Figura 3: Pila de expedientes

3. Un Ejemplo Paso a Paso

A continuación se desarrollará un ejemplo detallado de un TDA. El ejemplo seleccionado es el de una pila.

3.1. Análisis: ¿Qué es una pila?

Una pila es una estructura que se encuentra normalmente en la vida cotidiana, por ejemplo, una pila:

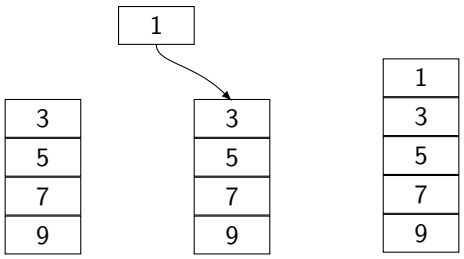
- de libros
- de expedientes
- de sillas
- de dinero
- de platos
- de monedas

Figura 4: Pila de Platos

3.1.1. Operaciones Básicas

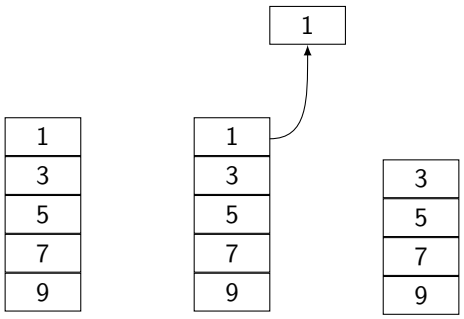
Sobre una pila se pueden realizar un conjunto de operaciones esenciales (básicas o primitivas). Estas operaciones son:

- **Apilar (push)**: poner un elemento en la pila, lo cual implica que este pasa a estar en el tope de la pila.



push(1)

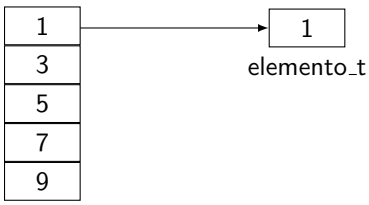
- **Desapilar (pop)** : sacar un elemento de la pila. El elemento que se encuentra en el tope de la pila es removido.



pop()=1

también existe un conjunto de operaciones no esenciales algunas de las cuales no siempre se implementan:

- **Ver Tope (top)**: ver el elemento que esta en el tope de la pila.



tope()

- **Está Vacía (is empty)** : informa si la pila está vacía.

vacia()= falso **vacia()= verdadero**

- **Llena (is full)**: informa si la pila esta llena.
- **Crear (create)**: crear la pila.
- **Destruir (dispose)**: eliminar la pila.

3.2. Implementación: ¿Cómo se implementa pila?

Si bien conceptualmente la funcionalidad o el qué de un tda es único, es decir, un tda bien diseñado debe tener una única funcionalidad. A nivel implementación esto no es para nada así. Existen varias implementaciones a una única funcionalidad. En otras palabras para un **único qué** existen **varios posibles cómos**. para poder ver esto se realizarán dos implementaciones distintas de un tda pila:

1. implementación con memoria estática, utilizando vectores.
2. implementación con memoria dinámica, mediante la utilización de punteros.

Un hecho importante a tener en cuenta es que la funcionalidad del tda está dada por las operaciones que este puede realizar sobre sus datos. Para ello se definen en un archivo cabecera de C en el cual además es necesario

```c
/* File stack.h */
#ifndef STACK_H
#define STACK_H

typedef int ElementType;

struct StackRecord;
typedef struct StackRecord *Stack;

void Pop( Stack S );
void Push( ElementType X, Stack S );

int IsEmpty( Stack S );
int IsFull( Stack S );
Stack CreateStack( int MaxElements );
void DisposeStack( Stack S );
void MakeEmpty( Stack S );

ElementType Top( Stack S );


#endif /* STACK_H */
```

3.2.1. Implemntación con memoria estática: arreglos

```c
#include "stack.h"

#define EMPTY_TOS               -1
#define MIN_STACK_SIZE  5
#define MAX                     1024

struct StackRecord{
        int Capacity;
        int TopOfStack;
        ElementType Array[MAX];
};
```