



Tipos de Datos Abstractos (TDA)

- Un TDA es un tipo de dato definido por el programador que se puede manipular de un modo similar a los tipos de datos definidos por el sistema.
- Está formado por un conjunto válido de elementos y un número de operaciones primitivas que se pueden realizar sobre ellos.

Ejemplo:

- Definición del tipo

Numero racional: Conjunto de pares de elementos (a,b) de tipo entero, con $b \neq 0$.

- Operaciones:

CrearRacional:	a, b	= (a,b)
Suma:	(a,b) + (c,d)	= (a*d+b*c , b*d)
Resta:	(a,b) - (c,d)	= (a*d-b*c , b*d)
Producto:	(a,b) * (c,d)	= (a*c , b*d)
División:	(a,b) / (c,d)	= (a*d , b*c)
Numerador:	(a,b)	= a
Denominador:	(a,b)	= b
ValorReal:	(a,b)	= a/b
MCD:	(a,b)	...
Potencia:	(a,b)^c	= (a^c , b^c)
Simplifica:	(a,b)	= (a/mcd(a,b) , b/mcd(a,b))

- Una vez definido se podrán declarar variables de ese tipo y operar con ellas utilizando las operaciones que aporta el tipo.

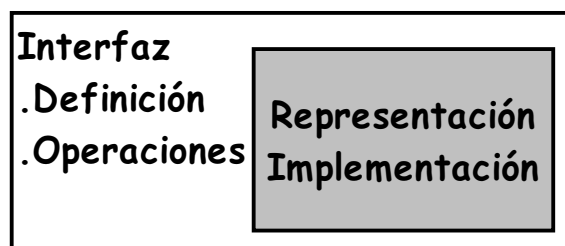
Ejemplo: `TRacional r1,r2, rsuma;`
`CrearRacional(4,7, &r1);`
`CrearRacional(5,8,&r2);`
`Suma(r1, r2, &rsuma);`
`printf("El valor real es %f", ValorReal(rsuma));`



- Un TDA es el elemento básico de la abstracción de datos. Su desarrollo es independiente del lenguaje de programación utilizado, aunque este puede aportar mecanismos que faciliten su realización. Debe verse como una caja negra.
- En un TDA existen dos elementos diferenciados:
 - La Interfaz de utilización
 - La representación

A la hora de utilizar el TDA, la representación debe permanecer oculta. Solo podremos utilizar las operaciones del tipo para trabajar con sus elementos.

TDA



- Para construir un tipo abstracto debemos:
 1. Exponer una definición del tipo.
 2. Definir las operaciones (funciones y procedimientos) que permitan operar con instancias de ese tipo.
 3. Ocultar la representación de los elementos del tipo de modo que *sólo* se pueda actuar sobre ellos con las operaciones proporcionadas.
 4. Poder hacer instancias múltiples del tipo.

◆ Tipos básicos de operaciones en un TDA

- **Constructores:** Crean una nueva instancia del tipo.
- **Transformación:** Cambian el valor de uno o más elementos de una instancia del tipo.
- **Observación:** Nos permiten observar el valor de uno o varios elementos de una instancia sin modificarlos.
- **Iteradores:** Nos permiten procesar todos los componentes en un TDA de forma secuencial.



◆ Implementación

- Una vez definido el TAD se escoge una representación interna utilizando los tipos que proporciona el lenguaje y/o otros TAD ya definidos previamente.
- La representación deberá ocultarse utilizando los mecanismos que nos proporcione el lenguaje. Ocultamiento de Información.
- Normalmente la implementación del tipo se realiza en un módulo aparte que será enlazado al programa principal
- Se necesitará un fichero cabecera que contenga la definición de las operaciones y la declaración del tipo (representación). Con esta información se podrán definir elementos del tipo y acceder a sus operaciones

TComplejo.h

operación1 operación2 operación3 ...	+	Definición del tipo
---	---	------------------------

TComplejo.c

Representación: estructura de datos
--

Implementación de operaciones: Código de operación1 Código de operación2 Código de operación3

Ejemplo: Fichero cabecera

```
struct _TRacional { int a,b};
typedef struct _TRacional TRacional;
```

```
void CreaRacional (int a, int b, TRacional *r );
void SumaRacional(TRacional r1, TRacional r2, TRacional *rsum);
```



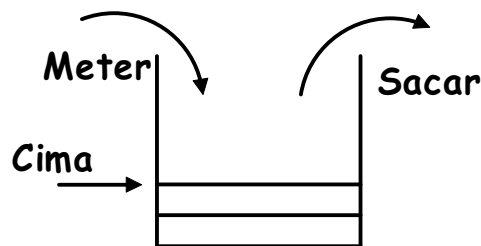
TDA Pila

Definición del Tipo

Es una colección lineal, dinámica y homogénea, en la que los elementos se insertan y se extraen por el mismo extremo. También conocida como estructura LIFO (Last In, First Out).

Operaciones:

CrearPila
Meter
Sacar
DestruirPila
EstaVacia



Representación:

Utilizaremos un array para representar la pila.

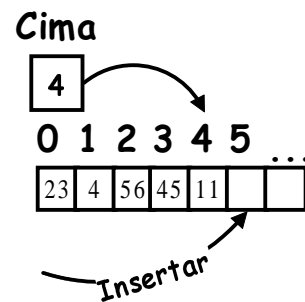
Definiremos un tamaño máximo de array (**MaxElemPila**).

Llevaremos una variable: **cima** que indicará cual es el último elemento ocupado en el array.

#define MaxElemPila

```
struct _TPilaEnteros {
    int elementos[MaxElemPila];
    int cima;
};
typedef struct _TPilaEnteros TPilaEnteros;
```

```
void CreaPila (TPilaEnteros *p) { p->cima = 0; };
int InsertaPila (int nelem, TPilaEnteros *p) {
    if (p->cima==MaxElemPila) {
        return 0; /*No se ha podido insertar*/
    } else {
        p->cima++;
        p->elementos[p->cima]=nelem;
        return 1;
    }
};
```





TDA Lista

Notas

- Es una estructura homogénea, dinámica y de acceso por posición.
- El tipo lista no existe en C por lo que habrá que implementarlo como un TAD.

Definición del tipo:

Una lista es una colección homogénea de elementos con una relación lineal entre ellos. Es decir, cada elemento de la lista (excepto el primero) tiene un único elemento predecesor y cada elemento (excepto el último) tienen un elemento sucesor

Operaciones:

Creación

CreaLista

Transformación

VaciarLista

InsertarElementoLista

BorrarElementoLista

ModificarElementoLista

Observación

LongitudLista

RecuperarElementoLista

Iteradores

PrimeroLista

SiguienteLista

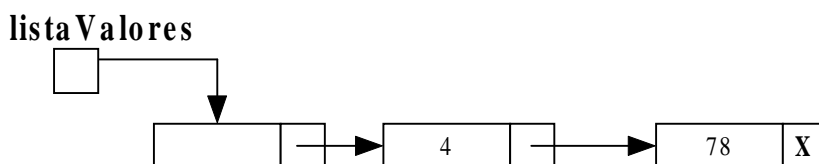
AnteriorLista

FinalLista



◆ Representación Listas

- La representación escogida dependerá de la utilización que se le vaya a dar al tipo. Lo normal es implementarlas como listas de elementos enlazados mediante punteros.



- Los elementos serán estructuras con un uno o varios campos de datos y un campo de tipo puntero que hará referencia al siguiente elemento.

Ejemplo de representación

/*lista de enteros*/

```

struct Tnodo {
    int dato;
    struct Tnodo *siguiente;
};
typedef struct Tnodo *TLista;
  
```

/*Permitira iterar sobre la lista*/

```

typedef struct Tnodo *TPosicion;
  
```

Ejemplo de declaración:

```

TLista listaValores;
TPosicion p;
  
```

◆ Operaciones habituales

CreaLista(L), BorraLista(L)	Crea una nueva lista vacía o la borra
EsVacía(L)	Determina si la lista está vacía
Inserta(x,p,L)	Inserta en la lista L un nodo con el campo X, delante del nodo de dirección p.
Localiza(x,L),Recupera (x,p,L)	Posición donde está el valor x o su valor
Suprime(p,L)	Elimina el nodo p
Anterior(p,L), Siguiente(p,L)	Posición anterior o siguiente del elemento p
Primero(L), Ultimo(L)	Posición del primer o último elemento



◆ Ejemplo de utilización de listas

```
#include <stdio.h>
#include <conio.h>
```

```
#include "listas.h"
```

```
int main () {
    int i,valor;
    TLista lista1;
    TPosicionLista pl1,pl2;
```

```
    clrscr();
```

```
    /*Creamos una lista e insertamos 10 elementos*/
```

```
    CrearLista1(&lista1); /*o también lista1=CrearLista2()*/
```

```
    for (i=1; i<=10; i++) { /*Rellenamos la lista*/
```

```
        InsertarFinalLista(i,lista1);
```

```
    };
```

```
    pl1=PrimeroLista(lista1); /*pl1 será un iterador de la lista*/
```

```
    pl2=FinalLista(lista1);
```

```
/*Recorremos los elementos de la lista del primero al último y los
presentamos en pantalla.*/
```

```
    while (pl1!=pl2) {
```

```
        RecuperaElementoLista(&valor,pl1,lista1);
```

```
        printf("\nEl valor es: %d",valor);
```

```
        pl1=SiguienteLista(pl1,lista1);
```

```
    };
```

```
/*Liberamos la memoria ocupada por la lista*/
```

```
    DestruirLista(&lista1);
```

```
};
```

◆ **FICHERO: listas.h**

```
/*=====
Representación del Tipo
=====*/
struct TNode {
    int info;
    struct TNode *siguiente;
};
typedef struct TNode *TLista;
typedef struct TNode *TposicionLista;

/*=====
Operaciones del Tipo
=====*/
/*Creación*/

int          CrearLista1 (TLista *l);
TLista       CrearLista2 (void);

/*Transformación*/

void InsertarElemento
    (int ndato, TposicionLista p, TLista l);
void InsertarFinalLista (int x, TLista l);
void ModificarElementoLista
    (int ndato, TposicionLista p, TLista l);
void EliminarElemento (TposicionLista p, TLista l);
void VaciarLista (TLista l);
void DestruirLista (TLista *l);

/*Observación*/

int VaciaLista (TLista l);
void RecuperaElementoLista
    (int *ndato, TposicionLista p, TLista l);

/*Iteración*/

TposicionLista PrimeroLista (TLista l);
TposicionLista FinalLista (TLista l);
TposicionLista SiguieteLista (TposicionLista p,
    TLista l);
TposicionLista AnteriorLista (TposicionLista p,
    TLista l);
```

◆ **Fichero: listas.c**

```
#include <stdlib.h>
#include "listas.h"
/*=====
//MODULO: Tipo de Dato Abstracto LISTA DE ENTEROS
//AUTOR/ES: José Ramón Balsas Almagro
//VERSIÓN: 1.2
//ULTIMA REVISIÓN: 30-11-1999
//DESCRIPCIÓN: Implementación del TDA Lista de Enteros
// mediante memoria dinámica utilizando un puntero
// al nodo siguiente. Se tomará un nodo cabecera
// sin información por comodidad en la
// implementación
//UTILIZACIÓN: Definir variables del tipo TLista que
// se inicializarán mediante las operaciones
// CrearLista1 o CrearLista2. Dichas variables se
// deben eliminar cuando no se vayan a utilizar
// utilizando la operación DestruirLista. Las
// variables del tipo TPosicionLista sirven para
// referenciar posiciones concretas de elementos de
// una lista. Se debe incluir el fichero cabecera
// "listas.h" para poder trabajar con el tipo.
//PLATAFORMA: Independiente
=====
Inicializa una variable del tipo pasada por referencia.
Devuelve 1 si todo ha ido bien. 0 en caso contrario.
=====*/
int CrearLista1 (TLista *l) {
    *l=(TLista) malloc (sizeof(struct TNode));
    if (*l==NULL) {
        return (0);
    } else {
        (*l)->siguiente=NULL;
        return (1);
    }
};
/*=====
Devuelve un elemento del tipo como resultado de la función o el
valor NULL en caso contrario
=====*/
TLista CrearLista2 (void) {
    TLista l;
    l=(TLista) malloc (sizeof(struct TNode));
    if (l!=NULL) {l->siguiente=NULL;};
    return (l);
};
```



```
/*=====
Devuelve un valor distinto de 0 si la lista tiene algún
elemento. 0 en caso de que esté vacía.
Precondición: La lista debe estar creada
=====*/

int VaciaLista(TLista l) {
    return(l->siguiente==NULL);
};

/*=====
Devuelve la posición del primer elemento de la lista. Si
no tiene ninguno devolver la posición FINAL.
Precondición: La lista debe estar creada.
=====*/

TPosicionLista PrimeroLista(TLista l) {
    return (l);
};

/*=====
Devuelve la posición siguiente a la última posición.
Esta posición se usar para insertar elementos al final
de la lista.
Precondición: La lista debe estar creada
=====*/

TPosicionLista FinalLista(TLista l) {
    TPosicionLista p;
    p=l;
    while (p->siguiente!=NULL) {
        p=p->siguiente;
    };
    return (p);
};

/*=====
Devuelve la posición del siguiente elemento en la lista a
la posición p.
Precondición: p debe ser una posición válida de la lista
=====*/

TPosicionLista SiguienteLista (TPosicionLista p ,TLista l) {
    if (p!=NULL && p->siguiente!=NULL) {
        /*Comprobamos si tenemos una posición válida*/
        return(p->siguiente);
    };
};
```



```
/*=====
Devuelve la posición anterior a p en la lista l. Si p es
el primer elemento devuelve el valor NULL
Precondición: p debe ser una posición válida de la lista
=====*/
TPosicionLista AnteriorLista (TPosicionLista p, TLista l) {
    TPosicionLista q;
    q=l;
    while (q->siguiente!=p && q!=NULL) {
        q=q->siguiente;
    };
    return (q);
};

/*=====
Modifica el valor del elemento p de la lista con el dato
ndato.
Precondición: p debe ser una posición válida de la lista.
=====*/
void ModificarElementoLista(int ndato, TPosicionLista p, TLista l)
{
    p->siguiente->info = ndato;
};

/*=====
Devuelve el dato almacenado en el elemento de la lista p
Precondición: p debe ser una posición válida de la lista.
=====*/
void RecuperaElementoLista(int *ndato, TPosicionLista p, TLista l) {
    *ndato=p->siguiente->info;
};

/*=====
Crea un nuevo elemento en la lista l en la posición p,
desplazando el existente una posición más adelante. Si p
es la posición final crea un nuevo elemento al final de la lista.
Precondición: p debe ser una posición válida de la lista.
=====*/
void InsertarElemento (int ndato, TPosicionLista p, TLista l) {
    TPosicionLista q;

    q=(TPosicionLista)malloc(sizeof(struct TNode));
    q->info=ndato;
    q->siguiente=p->siguiente;
    p->siguiente=q;
};
```



```
/*=====
Elimina el elemento de la posición p de la lista l.
Precondición: p debe ser una posición válida de la lista.
                p no puede ser la posición final.
=====*/

void EliminarElemento (TPosicionLista p, TLista l) {
    TPosicionLista q;
    q=p->siguiente;
    p->siguiente=q->siguiente;
    free(q);
};

/*=====
Añade un elemento nuevo al final de la lista l
Precondición: La lista l debe estar creada.
=====*/

void InsertarFinalLista(int x, TLista l ) {
    InsertarElemento(x,FinalLista(l),l);
};

/*=====
Borra todos los elementos de la lista l dejándola vacía
Precondición: La lista l debe estar creada.
=====*/

void VaciarLista (TLista l) {
    TPosicionLista p,q;
    p=l->siguiente;
    l->siguiente=NULL;
    while (p!=NULL) {
        q=p->siguiente;
        free(p);
        p=q;
    };
};

/*=====
Borra todos los elementos de la lista l y libera la memoria
ocupada por la misma.
Precondición: La lista l debe estar creada.
=====*/

void DestruirLista (TLista *l) {
    VaciarLista(*l);
    free(*l);
};
```