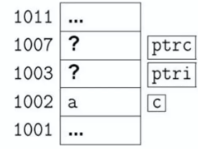
**Gestión de Memoria dinámica con TDA**

           En la memoria dinámica durante la ejecución del programa el tamaño de la estructura puede cambiar, las estructuras de datos dinámicas se generan a partir de un dato conocido como referencia (dirección de memoria).

**1. Se dividen en dos grandes grupos**

1. Lineales: Pilas, Colas, Listas enlazadas.
2. No Lineales: Arboles y Grafos.

**Punteros:** Es una variable que contiene una posición de memoria, y por tanto se dice que apunta a esa posición de memoria.



**Figura 5** *Punteros*

**Declaración:** Declaración <tipo> \*<identificador> <tipo> Tipo de dato del objeto referenciado por el puntero <identificador> Identificador de la variable de tipo puntero.

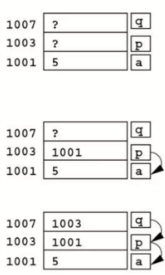
**Dirección:** Operador & &<id> devuelve La dirección de memoria donde comienza la variable <id>.

El operador & se utiliza para asignar valores a datos de tipo puntero

**Indirección:** Operador \* \* devuelve el contenido del referenciado por el puntero. objeto El operador \* se usa para acceder a los objetos a los que apunta un puntero

**Asignación:** Operador = A un puntero se le puede asignar una dirección de memoria concreta, la dirección de una variable o el contenido de otro puntero.

**Punteros a punteros:** Es un puntero que contiene la dirección de memoria de otro puntero.



**Figura 6** *Asignación de puntero a puntero*

           Para acceder al valor de la variable a podemos escribir a (forma habitual)

\*p (a través del puntero p)

\*\*q (a través del puntero a puntero q)

q contiene la dirección de p, que contiene la dirección de a

**Operador New:** Sirve para reservar memoria Este operador permite crear un objeto de cualquier tipo, incluyendo tipos definidos por el usuario, devuelve un adecuado) al objeto creado.

**Operador Delete:** Se usa para liberar la memoria dinámica reservada con new. La expresión será normalmente un puntero, el operador delete[] se usa para liberar memoria de arrays dinámicos. Es importante liberar siempre usando delete la memoria reservada con new.

**5. Implementación de TDA**

La implementación de un TDA puede variar dependiendo del lenguaje de programación y del contexto en el que se utilice. Sin embargo, hay algunas pautas generales que se pueden seguir:

* Definir una interfaz clara que especifique las operaciones que se pueden realizar sobre el TDA.
* Implementar las operaciones utilizando las estructuras de datos y algoritmos adecuados.
* Ocultar los detalles de implementación detrás de la interfaz, utilizando encapsulamiento u otros mecanismos de protección de datos.

Siguiendo con el ejemplo de las fechas, podemos indicar que las operaciones válidas sobre una fecha son, entre otras:

Crear (dia, mes, año: natural): fecha

Incrementar (fechaInicio: fecha; numDias: entero): fecha

Distancia (fechaInicio, fin: fecha): entero

ObtenerMes (f: fecha): natural

***Fecha.h***

**#ifndef FECHA\_H**

**#define FECHA\_H**

**class Fecha {**

**private:**

**int dia, mes, anio;**

**public:**

**Fecha(int d, int m, int a);**

**Fecha incrementar(int numDias);**

**int distancia(Fecha fin); // cambiar la variable periodo de tiempo.**

**int obtenerMes();**

**private:**

**int diasEnMes(int m, int a);**

**int diasDesde1900(int d, int m, int a);**

**};**

***Fecha.cpp***

**#endif**

**#include "Fecha.h"**

**Fecha::Fecha(int d, int m, int a) {**

**dia = d;**

**mes = m;**

**anio = a;**

**}**

**Fecha Fecha::incrementar(int numDias) {**

**dia += numDias;**

**if (dia > diasEnMes(mes, anio)) {**

**dia -= diasEnMes(mes, anio);**

**mes++;**

**if (mes > 12) {**

**mes = 1;**

**anio++;**

**}**

**}**

**return Fecha(dia, mes, anio);**

**}**

**int Fecha::distancia(Fecha fin) {**

**int diasInicio = diasDesde1900(dia, mes, anio);**

**int diasFin = diasDesde1900(fin.dia, fin.mes, fin.anio);**

**return diasFin - diasInicio;**

**}**

**int Fecha::obtenerMes() {**

**return mes;**

**}**

**int Fecha::diasEnMes(int m, int a) {**

**if (m == 2) {**

**if (a % 4 == 0 && (a % 100 != 0 || a % 400 == 0)) {**

**return 29;**

**} else {**

**return 28;**

**}**

**} else if (m == 4 || m == 6 || m == 9 || m == 11) {**

**return 30;**

**} else {**

**return 31;**

**}**

**}**

**int Fecha::diasDesde1900(int d, int m, int a) {**

**int dias = 0;**

**for (int i = 1900; i < a; i++) {**

**if (i % 4 == 0 && (i % 100 != 0 || i % 400 == 0)) {**

**dias += 366;**

**} else {**

**dias += 365;**

**}**

**}**

**for (int i = 1; i < m; i++) {**

**dias += diasEnMes(i, a);**

**}**

**dias += d - 1;**

**return dias;**

}

**6. Conclusiones**

En resumen, los Tipos de Datos Abstractos (TDA) son una herramienta poderosa para la programación y el diseño de software. Al encapsular datos y operaciones relacionadas en una sola entidad, los TDA promueven el modularidad, el reúso de código y la abstracción, facilitando el desarrollo de sistemas complejos.

**7. Referencias**

[1] Abstraction and Specification in Program Development, Barbara Liskov and John Guttag.

[2] Data Structures and Algorithms in Java, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser.