Licenciatura en Sistemas
Programación de computadoras

Equipo docente: Jorge Golfieri, Natalia Romero, Romina Masilla y Nicolás Perez
Mails: jgolfieri@hotmail.com , nataliab_romero@yahoo.com.ar , romina.e.mansilla@gmail.com, nperez_dcao_smn@outlook.com
Facebook: https://www.facebook.com/groups/171510736842353
Git: ***http://***github.com/UNLASistemasPrograma cion/Programacion-de-Computadoras

---

Unidad 7:

Tipos de Datos Abstractos (TDA). Definición e implementación en C. Aplicaciones.

Bibliografía citada:

Luis Joyanes Aguilar, Andrés Castillo Sanz, y Lucas Sánchez García (2005) - C algoritmos, programación y estructuras de datos - McGraw-Hill España. Capítulo 17.

TDA – Guía Teórica

Concepto de abstracción:

Bueno para arrancar vamos a definir el concepto de Abstracción, este concepto no es muy fácil de entender al principio, pero con tiempo y dedicación, uno logra captarlo.

El concepto de la Abstracción o como comúnmente se denomina Mecanismo de Abstracción, en formas generales es la consideración de ciertas partes seleccionadas de un todo complejo, ignorando las partes sobrantes.

Bien hasta acá no aportamos mucho a la cuestión, para eso, vamos a aplicarlo a lo nuestro, es decir a un nivel más Ingenieril. La Abstracción me otorga la posibilidad de considerar una resolución de un cierto problema en el cual, no tenemos idea de cómo está resuelto, o demostrado o etc., es decir, que ignoramos todo lo que está por debajo del nivel que nos interesa. En este caso a nosotros nos interesa solo como se utiliza, lo cual un nivel inferior a lo que nos interesa es el “como esta implementado, o desarrollado, o demostrado, etc” por ende lo descartamos.

Es más podemos citar ejemplos en nuestra vida cotidiana en la cual, aplicamos el mecanismo de Abstracción constantemente sin quizás darnos cuenta que lo hacemos.

Como por ejemplo:

Al utilizar la televisión, es decir no nos interesa el cómo funciona, solo sabemos usarla. Al utilizar el auto, solo sabemos conducirlo, pero quizás no sabemos su complejo funcionamiento interno. Al programar en ciertos lenguajes, utilizamos funciones/procedimientos ya existentes sin saber cómo fueron construidos.

¿Qué es un TDA?:

Con mucha frecuencia se utilizan los términos *TDA* y *Abstracción de Datos* de manera equivalente, y esto es debido a la similitud e interdependencia de ambos. Sin embargo, es importante definir por separado los dos conceptos.

La abstracción de datos consiste en ocultar las características de un objeto y obviarlas, de manera que solamente utilizamos el nombre del objeto en nuestro programa. Esto es similar a una situación de la vida cotidiana. Cuando yo digo la palabra "perro", usted no necesita que yo le diga lo que hace el perro. Usted ya sabe la forma que tiene un perro y también sabe que los perros ladran. De manera que yo abstraigo todas las características de todos los perros en un solo término, al cual llamo "perro". A esto se le llama 'Abstracción' y es un concepto muy útil en la programación, ya que un usuario no necesita mencionar todas las características y funciones de un objeto cada vez que este se utiliza, sino que son declaradas por separado en el programa y simplemente se utiliza el término abstracto ("perro") para mencionarlo.

En el ejemplo anterior, "perro" es un Tipo de Dato Abstracto y todo el proceso de definirlo, implementarlo y mencionarlo es a lo que llamamos Abstracción de Datos.

Supongamos ahora un pequeño programa saca el área de un rectángulo de las dimensiones que un usuario decida. Pensemos también que el usuario probablemente quiera saber el área de varios rectángulos. Sería muy tedioso para el programador

definir la multiplicación de ‘base’ por ‘altura’ varias veces en el programa, además que limitaría al usuario a sacar un número determinado de áreas. Por ello, el programador puede crear una función denominada ‘Área’, la cual va a ser llamada el número de veces que sean necesitadas por el usuario y así el programador se evita mucho trabajo, el programa resulta más rápido, más eficiente y de menor longitud. Para lograr esto, se crea el método Área de una manera separada de la interfaz gráfica presentada al usuario y se estipula ahí la operación a realizar, devolviendo el valor de la multiplicación. En el método principal solamente se llama a la función Área y el programa hace el resto.

Al hecho de guardar todas las características y habilidades de un objeto por separado se le llama Encapsulamiento y es también un concepto importante para entender la estructuración de datos. Es frecuente que el Encapsulamiento sea usado como un sinónimo del Ocultación de información.

Importancia de los TDA:

Cuando se usa en un programa de computación, un TDA es representado por su interfaz, la cual sirve como cubierta a la correspondiente implementación. La idea es que los usuarios de un TDA tengan que preocuparse solo por la interfaz, pero no por la implementación, ya que esta puede ir cambiando con el tiempo y, si no existiera encapsulación, afectar a los programas que usan el dato. Esto se basa en el concepto de Ocultación de información, una protección para el programa de decisiones de diseño que son objeto de cambio.

La solidez de un TDA reposa en la idea de que la implementación está escondida al usuario. Solo la interfaz es pública. Esto significa que el TDA puede ser implementado de diferentes formas, pero mientras se mantenga consistente con la interfaz, los programas que lo usan no se ven afectados.

¿Cómo usar la abstracción para crear un TDA? :

La construcción de un TDA, cambia bastante a diferencia de su utilización al momento de aplicar el mecanismo de abstracción. Para eso vamos primero a recordar lo siguiente:

Durante esta materia ustedes solo creaban software cuyos clientes eran personas comunes, es decir, personas comunes y corrientes que no saben sobre programación, por lo tanto, creaban programas en los cuales, les daban opciones al usuario que ingrese datos o que haga algo o etc, y el usuario seguía todos los pasos, como pueden apreciar, ustedes construían software, de tal manera que lograban que el usuario se abstraiga del como estaba programado (para el usuario "creado") y lo utilice sin preocupación, ya que daba fe, que ese programa funcionaba correctamente.

Bueno, ahora cuando uno cree un TDA, lo que está haciendo es exactamente lo mismo. Uno puede estar sorprendido con esta afirmación, ya que le resulta medio raro que un TDA sea un software que lo use un cliente. Pues bien, un TDA es un software, cuyo destinatario o usuario, es en definitiva otro programador, así es, otra persona que sabe programar al igual que ustedes. Por ende al igual que un programa común y corriente, al momento de crear un TDA, ustedes tienen que entregarle al usuario, la documentación correspondiente al mismo, junto con los archivos necesarios para que dicho TDA funcione.

Pero al momento de construir un TDA, se tiene que tener un buen diseño previo del mismo. Para eso vamos a ver el concepto de diseño de un TDA.

Interfaz VS la implementación:

Para separar la interfaz del TDA, es decir aquellas funciones y tipos que el programador "usuario" utilizará, de los detalles de implementación, vamos a utilizar los archivos headers de C.

Es decir, los .h.

- **Interfaz:** declarada en el archivo .h
- **Implementación**: declarada en el .c

Y el programa de prueba o cliente deberá simplemente incluir el .h

Veamos un ejemplo simple donde queremos modelar los **números racionales** es decir los que se expresan como el cociente de dos números como 1/4, 2/3, etc.

¿Qué es lo primero que hay que hacer?

Definir la interfaz es decir el .h, en nuestro caso podría ser fracción.h

Por ejemplo podría quedarnos algo así:

```
struct FraccionStruct;
typedef struct FraccionStruct* Fraccion;
```

Luego definimos la implementación, es decir las operaciones que podremos realizar con esta nueva estructura.

Las cuales podrían ser:

```
Fraccion crear(int n, int d);
int numerador(Fraccion x);
int denominador(Fraccion x);
Fraccion sumar(Fraccion x, Fraccion y);
Fraccion restar(Fraccion x, Fraccion y);
Fraccion multiplicar(Fraccion x, Fraccion y);
Fraccion dividir(Fraccion x, Fraccion y);
int iguales(Fraccion x, Fraccion y);
```

```
Fraccion simplificar(Fraccion fraccion);
void imprimir(Fraccion x);
```

Relación con el main:

Sin importarnos los detalles de cómo se va a implementar esto, ya estaríamos en condiciones de hacer una aplicación de ejemplo que use estas fracciones. Por ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include "fraccion.h"

int main(void) {
    Fraccion dosQuintos = crear(2, 5);
    imprimir(dosQuintos);

    Fraccion suma = sumar(dosQuintos, crear(2,5));
    imprimir(suma);
    Fraccion sumaSimplificada = simplificar(suma);
    imprimir(sumaSimplificada);

    return 0;
}
```

Pero bien, claramente todo lo escrito arriba no haría nada puesto que no le asignamos un comoprtamiento a cada una de las funciones o procedimientos descriptos en el .h, asi que ahora es lo que hay que realizar, vamos a crear el archivo fracción.c:

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

```
#include "fraccion.h"

struct FraccionStruct {
    int numerador;
    int denominador;
};

Fraccion crear(int n, int d) {
    struct FraccionStruct* fraccion = malloc(sizeof(struct FraccionStruct));
    fraccion->numerador = n;
    fraccion->denominador = d;
    return fraccion;
}

int numerador(Fraccion x) {
    return x->numerador;
}

int denominador(Fraccion x) {
    return x->denominador;
}

Fraccion sumar(Fraccion x, Fraccion y) {
    int numerador = x->numerador * y->denominador + x->denominador * y-
>numerador;
    int denominador = x->denominador * y->denominador;
    return crear(numerador, denominador);
}

Fraccion restar(Fraccion x, Fraccion y) {
    int numerador = x->numerador * y->denominador - x->denominador * y-
>numerador;
    int denominador = x->denominador * y->denominador;
    return crear(numerador, denominador);
```

```
}

Fraccion multiplicar(Fraccion x, Fraccion y) {
   int numerador = x->numerador * y->numerador;
   int denominador = x->denominador * y->denominador;
   return crear(numerador, denominador);
}

Fraccion dividir(Fraccion x, Fraccion y) {
   int numerador = x->numerador * y->denominador;
   int denominador = x->denominador * y->numerador;
   return crear(numerador, denominador);
}

int iguales(Fraccion x, Fraccion y) {
   return x->numerador * y->denominador == x->denominador * y->numerador;
}

int maximoComunDenominador(int mayor, int menor) {
   if (menor == 0) {
      return mayor;
   }
   if (menor > mayor) {
      return maximoComunDenominador(menor, mayor);
   }
   int resto = mayor % menor;
   return maximoComunDenominador(menor, resto);
}

Fraccion simplificar(Fraccion fraccion) {
   int mcd = maximoComunDenominador(fraccion->numerador, fraccion-
>denominador);
   int numerador = fraccion->numerador / mcd;
   int denominador = fraccion->denominador / mcd;
```

```
    return crear(numerador, denominador);
}

void imprimir(Fraccion x) {
    printf("%d/%d\n", x->numerador, x->denominador);
}
```

Reflexiones sobre los TDA:

El TDA, tiene que modelar solo un tema, es decir que el TDA tiene que resolver solo un problema y no varios a la vez, en este caso, supongamos que tenemos el TDA fracción, dicho TDA, tiene que modelar y resolver las operaciones con fracciones y no dedicarse a otra cosa.

Las funciones/procedimientos de los TDA, se denominan "Primitivas" y dichas Primitivas, tienen que contener códigos cortos, no tienen que ser extremadamente largas, una Primitiva con pocas líneas de código, es una Primitiva bien diseñada.

Todo TDA tiene que tener una primitiva para su "construcción" y (en algunos casos) para su "destrucción".

Las primitivas del TDA tienen que permitirle al usuario manipularlo dándole la mínima información sobre la implementación del TDA, esto se denomina Ocultamiento de la Información con el cual al usuario solo le Informamos de cómo se compone y que herramientas tiene a la alcance para manipularlo. El ocultamiento de la implementación de las primitivas, es lo que se denomina como Encapsulamiento en el cual ocultamos toda línea de código existente en dicho TDA.

Ahora que ya entendimos más o menos el funcionamiento y el uso de los TDA, vayamos a enfocarnos un poco más en los componentes de dichos TDAs.

Un TDA se compone de:

Tipo de dato, axiomas, pre condiciones, post condiciones.

Donde cuando hablamos de tipo de dato nos referimos a:

Los atributos que lo componen, es decir sus variables.

Cuando hablamos de axiomas nos referimos a:

Los axiomas son proposiciones lógicas que deben cumplirse siempre. Son aceptadas sin requerir una demostración previa. Un ejemplo seria decir, edad_persona tiene que ser mayor a 0.

Cuando hablamos de operaciones, nos referimos a:

Las operaciones son también llamadas primitivas y es el único vínculo entre el usuario y el TDA. No existe ninguna otra forma de interactuar con ellos.
Las operaciones características de los TDA son los constructores, destructores, gets y sets. Donde pasaremos a prestarles más atención en los próximos ejemplos.

Las pre y post condiciones ya las hemos visto y usado en las primeras clases:

Las precondiciones son proposiciones lógicas que deben cumplirse antes de la ejecución de una primitiva. Reflejan la realidad de que no todos los estados posibles son válidos para el TDA (no puedo estados posibles son válidos para el TDA (no puedo pedir el primer carácter a un String vacío)).
Las post condiciones son proposiciones lógicas que resultan verdaderas luego de la ejecución de una primitiva.

TDA – Guía Practica

Ejercicio 1:

Los números enteros, con la suma, la resta, la multiplicación y la división como operaciones básicas ¿es un tipo abstracto de datos? Razona la respuesta.

Ejercicio 2:

Supongamos el tipo abstracto de datos 'Numero complejo', las operaciones que conocemos sobre él: suma, resta, multiplicación, división y conjugado de complejos.

Ejercicio 3:

Supongamos el tipo abstracto 'Matriz de números enteros'. ¿Qué operaciones básicas sería conveniente definir?

Implementa con clases de C el tipo abstracto 'Matriz de números enteros', y las operaciones que has definido para él.

Ejercicio 4

Supongamos el tipo abstracto de datos 'Cadena_Caracteres', que tiene definida la siguiente operación:

Concatenar (Cadena_Caracteres, Cadena_Caracteres). ¿Qué estructura de datos debemos usar para representar el problema?

Ejercicio 5:

Supongamos el tipo abstracto de datos 'Fecha' que contiene día, mes y año. Las operaciones definidas sobre este tipo serán:

Crear (Entero, Entero, Entero) que retorna una Fecha.

Que a partir de un cierto día, mes y año crea una fecha correcta. Si los valores de día, mes y/o año fueran incompatibles mostrará error.

Incrementar (Fecha, Entero) que retorna una Fecha.

Que incrementa la Fecha en un cierto valor de días.

a.- Definir al menos dos representaciones válidas del TAD 'Fecha' en C.

b.- Implementa correctamente como una clase en C las operaciones definidas sobre el TAD 'Fecha'.

Ejercicio 6:

¿Por qué es interesante la utilización de tipos abstractos de datos en programación?