ESCUELA DE INGENIERÍA DE SISTEMAS
DEPARTAMENTO DE COMPUTACIÓN
PROGRAMACIÓN 2

PRÁCTICA DE LABORATORIO 3

Tipo Abstrato de Dato

Contenido

1. Introducción

Esta práctica introduce el concepto de Tipo Abstracto de Dato para que el estudiante pueda comprender, construir y utilizar esta poderosa herramienta de programación.

Antes de entrar en los detalles de los TADs entendamos antes algunos conceptos básicos. Con la aparición de los lenguajes de programación en los años 60 se definieron los tipos de datos para adicionar un mayor nivel de expresividad a los programas. La mayoría de los lenguajes de programación incluyen tipos de datos predefinidos que pueden ser utilizados por el programador para realizar distintas actividades dentro del desarrollo de un programa, por ejemplo, enteros, punto flotantes, caracteres, etc.

Dato: En el computador un dato siempre se resume a un conjunto de bits. Un conjunto de reglas sobre esos bits forman un tipo de dato. Por ejemplo: un dato de tipo entero o tipo caracter.

Valor: Es un elemento perteneciente a un conjunto. Este conjunto representa un tipo de dato y todos sus elementos (valores) deben tener las mismas propiedades.

El conjunto define las operaciones que se pueden aplicar a sus valores. Por ejemplo: El valor 2 es un elemento del conjunto siguiente:

$$N = \{ 1, 2, 3 ..... \}$$

Tipo de Dato
Es la agrupación de un conjunto de valores sobre el cual se puede realizar un conjunto de operaciones. Ejemplo, El tipo de dato fecha podría estar representado por los atributos DIA, MES y AÑO, siendo los tres de tipo entero. Las operaciones aplicables sobre el tipo de dato fecha podrían ser:

- Mostrar fecha
- Incrementar día, mes o año
- Intervalo entre dos fechas

Tipo Abstracto de Dato: un TAD es un conjunto de valores sobre los cuales se aplica un conjunto dado de operaciones que cumplen determinadas propiedades. En realidad, los TAD ya existe en los lenguajes de programación estructurados bajo la forma de los tipos predefinidos, que se pueden considerar como tipos abstractos de datos sencillos.

Si consideramos por ejemplo el tipo de datos de los enteros que ofrece el lenguaje C podemos definirlo determinando los siguientes elementos:

- Cuáles son sus valores: los números enteros dentro del intérvalo [minint, maxint];

- Cuáles son sus operaciones: la suma, la resta, el producto, y el cociente y el resto de la división

- Cuáles son las propiedades que cumplen estas operaciones: hay muchas; por ejemplo:

$$a+b = b+a,$$
$$a*0 = 0,$$
etc.

2. Trabajo Práctico

En esta sección el estudiante será guiado en la confección de un TAD. Se tomará como ejemplo el TAD Matriz. Al mismo tiempo se le mostrará al estudiante como organizar el código fuente de tal manera de cumplir con un estilo de programación estándar para desarrollos de dimensión considerable.

Definición del TAD Matriz
Como se mencionó en la sección anterior nosotros debemos determinar 3 elementos para diseñar un TAD. Para esto consideremos matrices cuyos

elementos sean números reales. Así, lo primero que debemos hacer es definir el rango de valores de cada elemento de la matriz. Luego, tenemos que considerar también las dimensiones de la matriz, es decir, el número de filas y columnas que esta tendrá.

Rango de Valores

El número de filas y columnas de una matriz son números enteros. Si definimos estos dos atributos utilizando el tipo de dato entero sin signo entonces podemos tener como rango de valores lo siguiente:

filas: [0 – 65535]
columnas: [0 – 65535]

Eso nos daria una matriz de tamaño máximo de 4.294.836.225 elementos. Si multiplicamos esa cantidad por el tamaño de un punto flotante (4Bytes) entonces necesitamos aproximandamente 16 MB para almacenar en memoria RAM una matriz de ese tamaño.

Respecto al rango de valores de los números reales que podemos representar con la notación punto flotante se recomienda leer el artículo de David Goldberg[1]

Operaciones

Existen muchas operaciones que podemos realizar utilizando matrices, entre ellas tenemos:

- Suma
- Multiplicación
- Matriz Traspuesta
- Determinante
- Multiplicación por un escalar, etc.

Por sencillez y para hacer este trabajo práctico realizable en un corto tiempo definiremos sólo algunas de las operaciones que se pueden hacer con matrices; primero un conjunto de operaciones que nos permiten la gestión del TAD y luego una operación real: inicializarMatriz, crearMatriz, mostrarMatriz, sumarMatrices.

Propiedades

Debemos definir las propiedades de cada operación para construir el TAD. Como hemos tomado la suma de matrices como ejemplo, podemos mencionar algunas de las propiedades de esta operación:

---

1 *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, by David Goldberg, published in the March, 1991 issue of Computing Surveys. Copyright 1991, Association for Computing Machinery, Inc.
http://docs.sun.com/source/806-3568/ncg_goldberg.html

- Asociativa: Dadas las matrices $m \times n$ A, B y C

  $A + (B + C) = (A + B) + C$

- Conmutativa: Dadas las matrices $m \times n$ A y B

  $A + B = B + A$

- Elemento Neutro:

  $A + 0 = 0 + A = A$

- Matriz Opuesta: donde $-A = [-a_{ij}]$

  $A + (-A) = 0$

El estudiante debe encargarse de completar al menos 3 operaciones más y definir las propiedades correspondientes.

2.1. Implantación

El código del TAD será organizado en tres archivos:

- **Matriz.h**: Archivo cabecera contentivo de las estructuras de datos, variables globales y prototipos de las funciones
- **Matriz.c**: Archivo contentivo de la implantación de las funciones del TAD.
- **principal.c**: Archivo con la función principal (mail) donde se harán algunas pruebas del TAD.

Se utilizará un archivo Makefile para compilar todo el código.

1. Edite el archivo **Matriz.h** y agregue el siguiente código:

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

//Estructura que nos permite definir nuestro propio
//tipo de dato Matriz

typedef struct {
        int filas;
        int columnas;
        double **valores;
} Matriz;

//Prototipos de las funciones (operaciones del TAD Matriz)
```

```
Matriz crearMatriz(int, int);
void inicializarMatriz(Matriz *, double);
void imprimirMatriz(Matriz);
Matriz sumarMatrices(Matriz, Matriz);
```

2. Edite el archivo **Matriz.c** y agregue las siguientes líneas:

```
#include "Matriz.h"

Matriz crearMatriz(int f, int c){
        Matriz m;
        int i,j;

        m.filas = f;
        m.columnas = c;

        m.valores = (double **)malloc(sizeof(double)*f*c);
        for(i=0; i<f; i++){
            m.valores[i] = (double *)malloc(sizeof(double)*c);
        }
        return m;
}

void inicializarMatriz(Matriz *m, double v){
        int i,j;
        for(i=0; i<m->filas; i++)
                for(j=0; j<m->columnas; j++)
                        m->valores[i][j] = v;
}

void imprimirMatriz(Matriz m){
        int i,j;
        for(i=0; i<m.filas; i++){
                for(j=0; j<m.columnas; j++)
                        printf("%.2f ", m.valores[i][j]);
                printf("\n");
        }
}

Matriz sumarMatrices(Matriz a, Matriz b){
        Matriz c;
        int i,j;
        c = crearMatriz(a.filas, a.columnas);
        for(i=0; i<a.filas; i++)
                for(j=0; j<a.columnas; j++)
                        c.valores[i][j] = a.valores[i][j] +
                                          b.valores[i][j];
```

```
        return c;
}
```

3. Edite el archivo principal.c y coloque las siguientes instrucciones:

```
/* Para correr este programa debe ejecutar el
 * siguiente comando:
 *
 *       ./matriz f c v1 v2
 *
 *
 * donde:
 * f: es el numero de filas de las dos matrices
 * c: es el numero de columnas de las dos matrices
 * v1: es el valor de inicialización de la matriz 1
 * v2: es el valor de inicialización de la matriz 2
 */

#include <stdio.h>
#include "Matriz.h"

int main(int argc, char *argv[]){
        Matriz a, b, c;
        int filas, columnas;
        int valor1, valor2;

        filas = atoi(argv[1]);
        columnas = atoi(argv[2]);
        valor1 = atoi(argv[3]);
        valor2 = atoi(argv[4]);

        a = crearMatriz(filas, columnas);
        inicializarMatriz(&a, valor1);
        imprimirMatriz(a);

        b = crearMatriz(filas, columnas);
        inicializarMatriz(&b, valor2);
        imprimirMatriz(b);

        c = sumarMatrices(a, b);
        imprimirMatriz(c);

        return 0;
}
```

Las intrucciones para correr este programa están en el comentario ubicado al principio del archivo **principal.c**

4. Por último edite un archivo de nombre **Makefile** y coloque las siguientes líneas dentro de él.

```
CC  =   gcc
CFLAGS  = -g
OTHERFLAGS = -c

main:  principal.o Matriz.o
        $(CC) $(CFLAGS)  principal.o Matriz.o -o matriz

principal.o: principal.c
        $(CC)  $(CFLAGS) $(OTHERFLAGS) principal.c

Matriz.o: Matriz.c
        $(CC)  $(CFLAGS) $(OTHERFLAGS) Matriz.c
```

5. Para compilar ejecute el comando

   make

3. Ejercicio

1. La operación de suma de matrices no es versátil en el sentido que sólo suma matrices cuadradas con iguales dimensiones. Arregle el código de tal función para que pueda sumar matrices de cualquier dimensión, es decir, se pueda realizar operaciones como:

   $A(m \times n) + B(r \times s)$

2. Especifique las propiedades de la multiplicación de matrices e implante esa operación.

3. Implante la operación de matriz inversa