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/UNLASistemasProgramacion/Programacion-de-Computadoras

---

Unidad Extra – Vectores de TDA:

Guía extra diseñada para facilitar el entendimiento de la próxima guía, en esta guía se planteará un ejercicio donde se relacionarán varios TDA. Problemas: implementación y testeo.

Bibliografía citada:

Luis Joyanes Aguilar , Luis Rodríguez Baena y Matilde Fernández Azuela. McGraw-Hill España – Capitulo 12.

Luis Joyanes Aguilar, Andrés Castillo Sanz, y Lucas Sánchez García (2005) - C algoritmos, programación y estructuras de datos - McGraw-Hill España. Capítulo 18 y 19.

Vectores con TDA – Previa de Listas – Guía Teórica

Hasta este momento hemos trabajado con varios TDA, pero todos por separado. ¿Qué ocurre si necesitamos vincularlos? Supongamos que tenemos un TDA Persona, y otro TDA Empleo. ¿Cómo haremos para hacer que una determinada Persona tenga un Empleo? ¿Como haríamos si esta persona tiene dos empleos? Estas preguntas trataremos de responder en esta guía.

Como se lo plantea arriba, tendríamos que darle a una Persona un Empleo, o quizás dos, o quizás tres, ¿Cómo manejar esto?

Caso 1: Una Persona tiene un único Empleo.

```
struct EmpleoEstructura
{
char nombreEmpleo[20];
}
```

```
struct PersonaEstructura
{
int dni;
char nombre;
Empleo e;
```

```
}
```

Pues bien, resulto muy simple, se puso dentro de la estructura un atributo o variable del tipo Empleo.

Caso 2: Una Persona tiene muchos Empleos. Aquí surge el problema que vamos a solucionar elegantemente cuando veamos listas, pero por ahora hagámoslo de una manera mas elemental y menos amplia para comprender el concepto.

Este problema se lo solucionara utilizando un vector o array de Empleo. Es decir, una Persona tendrá dentro de su estructura un vector de Empleo.

```
struct EmpleoEstructura
{
char nombreEmpleo[20];
}

struct PersonaEstructura
{
int dni;
char nombre;
Empleo empleos[20];
}
```

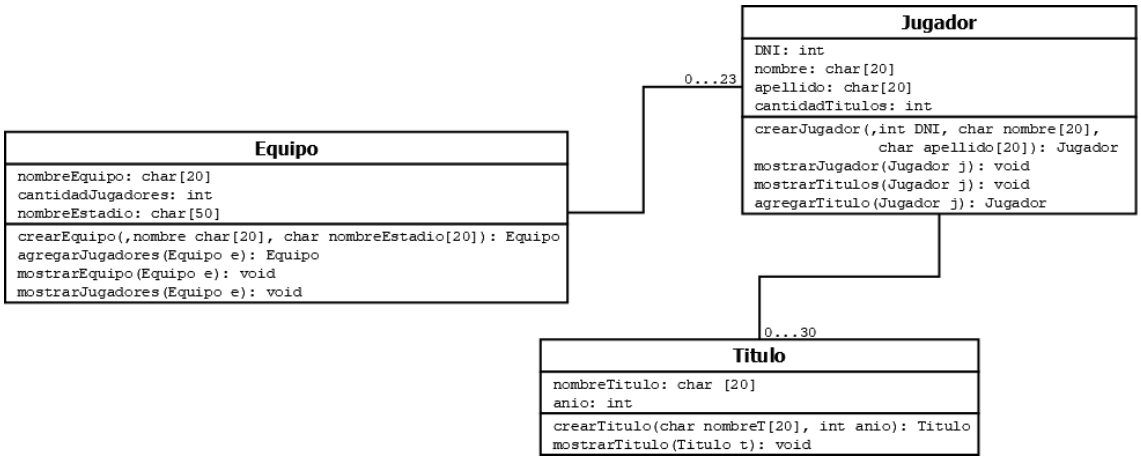
Surge entonces una inevitable pregunta, ¿porque puse un vector de 20 posiciones?, como claramente no sabes de ante mano cuantos empleos tiene cada Persona, debemos poner un numero lo suficientemente grande para asegurarnos que todas las personas podrán cargar sus empleos.

Claramente esta solución consume muchos mas recursos de los necesarios, ya que creamos 20 posiciones de las cuales quizás usemos 1, 2, o hasta ninguna. Es una solución poco eficiente, pero nos servirá de hincapié inicial para poder manejar las listas en un futuro no muy lejano.

Diagramas de TDA:

Para simplificar la comunicación entre programadores y evitar conflictos de arbitrariedad en las decisiones, suelen utilizarse diagramas que definen bien la estructura del cuerpo de un programa a construir. En futuras materias verán un sinfín de diagramas, en nuestro caso trabajaremos con diagramas de TDA.

A continuación, dejo expresado uno con el que trabajaremos e iremos explicando paso a paso como es su lectura.



¿Qué es lo que vemos en ese diagrama?, se puede apreciar que trabajaremos con tres TDA, un TDA Equipo, otro TDA Jugador y uno mas Titulo. En nuestro caso serán los punteros a la estructura que se van a definir, es decir:

typedef struct EquipoEstructura * Equipo;

typedef struct TituloEstructura * Titulo;

typedef struct JugadorEstrucutra * Jugador;

En el rectángulo inferior al nombre del TDA, vemos que están bien definidos cada uno de los atributos o variable del TDA. Por ejemplo, en Equipo tenemos el siguiente cuerpo:

struct EquipoEstrucutura

{

char nombreEquipo[20];

int cantidadJugadores;

char nombreEstadio[20];

}

¿Qué son entonces las líneas que unen cada uno de los TDA?, justamente las líneas que los unen quieren decir que los TDA están unidos o relacionados, en nuestros casos las líneas representaran vectores.

Por lo que nuestra estructura de Equipo quedaría:

```
struct EquipoEstrucutura
{
char nombreEquipo[20];
int cantidadJugadores;
char nombreEstadio[20];
Jugador jugadores[23];
}
```

Aquí se aprecia que el vector tendrá 23 posiciones, justamente eso indica el numero al final de la línea de relación entre los TDA.

¿Y que indica entonces el numero de la izquierda? El cero nos indica que podría no tener ningún Jugador el Equipo, si en vez de 0…23 tuviéramos un 5…23 querría decir que cada equipo debería tener como mínimo 5 jugadores cargados.

Lo ultimo que nos queda por saber es, ¿Qué es el tercer rectángulo de cada TDA?, allí tenemos las funciones o procedimientos que debemos tener en el .h y en el .c. No solo eso, vemos sus argumentos de entrada y su retorno.

Entonces la estructura de nuestro código fuente diagramado quedaría algo así:

equipo.h :

```
1  #ifndef EQUIPO_H_INCLUDED
2  #define EQUIPO_H_INCLUDED
```

```
3
4
5  struct EquipoEstructura;
6  typedef struct EquipoEstructura * Equipo;
7
8  //Pre:
9  //Post:
10  //Axiomas:
11  Equipo crearEquipo (char nombreEquipo[20], char nombreEstadio[20]);
12
13  //Pre:
14  //Post:
15  //Axiomas:
16  Equipo agregarJugador(Equipo e);
17
18  //Pre:
19  //Post:
20  //Axiomas:
21  void mostrarEquipo(Equipo e);
22
23  //Pre:
24  //Post:
25  //Axiomas:
26  void mostrarJugadores(Equipo e);
27
28
29  //Agregar los get y set
30
31
32
33  #endif // EQUIPO_H_INCLUDED
```

jugador.h:

```
1  #ifndef JUGADOR_H_INCLUDED
2  #define JUGADOR_H_INCLUDED
3
4
5  struct JugadorEstructura;
6  typedef struct JugadorEstructura * Jugador;
7
8
9  //Pre:
10  //Post:
11  //Axiomas:
12  Jugador crearJugador(int DNI, char nombre[20], char apellido[20]);
13
14  //Pre:
15  //Post:
16  //Axiomas:
17  void mostrarJugador(Jugador j);
18
19  //Pre:
20  //Post:
21  //Axiomas:
22  void mostrarTitulos(Jugador j);
23
24  //Pre:
25  //Post:
26  //Axiomas:
27  Jugador agregarTitulo(Jugador j);
28
29
30  //deben agregar los get y set
31
32
33
34
```

```
35  #endif // JUGADOR_H_INCLUDED
```

titulo.h:

```
 1  #ifndef TITULO_H_INCLUDED
 2  #define TITULO_H_INCLUDED
 3
 4  struct TituloEstructura;
 5  typedef struct TituloEstructura * Titulo;
 6
 7  //pre:
 8  //Post:
 9  //Axiomas:
10  Titulo crearTitulo(char nombreT[20], int anio);
11
12  //pre:
13  //Post:
14  //Axiomas:
15  void mostrarTitulo(Titulo t);
16
17
18  //agregar los get y set;
19
20
21
22  #endif // TITULO_H_INCLUDED
```

equipo.c:

```
 1  #include <stdio.h>
 2  #include <string.h>
```

```
 3  #include <stdlib.h>
 4
 5  #include "equipo.h"
 6  #include "jugador.h"
 7
 8
 9  struct EquipoEstructura
10  {
11      char nombreEquipo[20];
12      int cantidadJugadores;
13      char nombreEstadio[50];
14      Jugador jugadores[23];
15  };
16
17
18  ////////////////////////////////////////////////////////////////////////////////////////////////
19
20  //Pre:
21  //Post:
22  //Axiomas:
23  Equipo crearEquipo (char nombreEquipo[20], char nombreEstadio[50])
24  {
25
26      Equipo e = malloc(sizeof(struct EquipoEstructura));
27
28      strcpy (e->nombreEquipo, nombreEquipo);
29      strcpy (e->nombreEstadio, nombreEstadio);
30      e->cantidadJugadores = 0;
31
32
33
34      return e;
35  };
36
```

```
37  ////////////////////////////////////////////////////////////////////////////////////////////////
38
39  //Pre:
40  //Post:
41  //Axiomas:
42  Equipo agregarJugador(Equipo e)
43  {
44
45  //Aqui deben pedir los datos del jugador, crear al jugador
46  // y asignarlo al vector de jugadores, una vez hecho, cantidadJugadores mas uno
47
48      return e;
49  };
50
51
52  ////////////////////////////////////////////////////////////////////////////////////////////////
53
54  //Pre:
55  //Post:
56  //Axiomas:
57  void mostrarEquipo(Equipo e)
58  {
59
60      printf("--------------Esto debe mostrar al Equipo------------\n");
61
62  };
63
64
65  ////////////////////////////////////////////////////////////////////////////////////////////////
66
67  //Pre:
68  //Post:
69  //Axiomas:
70  void mostrarJugadores(Equipo e)
```

```
71  {
72
73
74      printf("--------------Jugadores del equipo------------\n");
75  }
```

jugador.c:

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5
6  #include "jugador.h"
7  #include "titulo.h"
8
9
10  struct JugadorEstructura{
11  int DNI;
12  char nombre[20];
13  char apellido[20];
14  int cantidadTitulos;
15  Titulo titulos[30];
16  };
17
18
19  ////////////////////////////////////////////////////////////////////////////////////////////////////
20
21
22  //Pre:
23  //Post:
```

```
24  //Axiomas:
25  Jugador crearJugador(int DNI, char nombre[20], char apellido[20]){
26
27  Jugador j = malloc(sizeof(struct JugadorEstructura));
28
29  strcpy (j->nombre, nombre);
30  strcpy (j->apellido, apellido);
31  j->DNI = DNI;
32
33  j->cantidadTitulos = 0;
34
35  return j;
36
37  };
38
39
40  ////////////////////////////////////////////////////////////////////////////////////////////////
41
42  //Pre:
43  //Post:
44  //Axiomas:
45  void mostrarJugador(Jugador j){
46
47  printf("-----Esto debe mostrar al jugador-----\n");
48  };
49
50  ////////////////////////////////////////////////////////////////////////////////////////////////
51
52  //Pre:
53  //Post:
54  //Axiomas:
55  void mostrarTitulos(Jugador j){
56
57  printf("-----Esto debe mostrar los titulos del jugador-----\n");
```

```
58  };
59
60  ////////////////////////////////////////////////////////////////////////////////////////////
61
62  //Pre:
63  //Post:
64  //Axiomas:
65  Jugador agregarTitulo(Jugador j){
66
67
68  printf("-----Esto debe pedir los datos del titulo\n y agregarlo al Jugador-----\n");
69
70
71  return j;
72  };
73
74
75  //deben agregar los get y set
```

titulo.c:

```
 1  #include <stdio.h>
 2  #include <string.h>
 3  #include <stdlib.h>
 4
 5  #include "titulo.h"
 6
 7
 8  struct TituloEstructura{
 9
10  char nombreT[20];
```

```
11  int anio;
12  };
13
14  //pre:
15  //Post:
16  //Axiomas:
17  Titulo crearTitulo(char nombreT[20], int anio){
18
19  Titulo t = malloc(sizeof(struct TituloEstructura));
20
21  strcpy(t->nombreT, nombreT);
22  t->anio = anio;
23
24
25  return t;
26
27
28  }
29
30  ////////////////////////////////////////////////////////////////////////////////////////////////
31
32  //pre:
33  //Post:
34  //Axiomas:
35  void mostrarTitulo(Titulo t){
36
37  printf("-----Esto debe mostrar los datos del tiitulo\n");
38
39  };
40
41
42  ////////////////////////////////////////////////////////////////////////////////////////////////
43
44
```

```
45  //Agregar los get y set
```

main.c:

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  #include "titulo.h"
 5  #include "jugador.h"
 6  #include "equipo.h"
 7
 8
 9  int main()
10  {
11      printf("---------- Equipo Version 1 --------\n");
12
13      Equipo e = crearEquipo("Velez Sarsfield", "Jose Amalfitani");
14
15      mostrarEquipo(e);
16
17
18      Jugador j = crearJugador(3332, "Nico", "Perez");
19      mostrarJugador(j);
20
21
22      Titulo t = crearTitulo("Libertadores", 2011);
23      mostrarTitulo(t);
24
25
26      return 0;
```

```
27  }
```

Vectores con TDA – Guía Practica

Se trabajará durante todo lo que queda del año con este ejemplo, por lo cual vayan siguiente los pasos de cada guía, así vamos mejorándolo escalonadamente.

1- Compilar y verificar el funcionamiento de los 6 archivos (.h y .c), con un breve testeo en el main.
2- Agregar todos los gets y sets.
3- Agregar las pre, post condiciones y los axiomas en los comentarios.
4- Manejar los errores pertinentes en los set, por ejemplo impedir que el DNI sea negativo, etc.
5- Realizar un menú que permita crear equipos, agregarle jugadores, y agregarles títulos a los jugadores.
6- Mostrar por consola un reporte completo de todo lo que se cargó.

Trabajos próximos, se agregarán los Jugadores a una lista, lo mismo para los Títulos. Luego trabajaremos con Pilas y Colas de jugadores, para por último terminar guardando en un archivo todo lo cargado.