Algoritmos y estructuras de Datos
P2PU

# Punteros (Continuación)

## Semana 4

Dictado por Marco González Núñez
21 de Febrero de 2011

# Algoritmos y estructuras de Datos
# P2PU

## Introducción

A continuación veremos la forma de crear estructuras y punteros en código C, que es lo que yo les puedo enseñar, ya que no he manejado otro lenguaje para este tema.

En esta semana veremos la creación, modificación, eliminación y búsqueda, además integraremos punteros doblemente enlazados y circulares.

Asi que comencemos...

**nota:** Todos los códigos de esta clase son echos por mi, y no estan compilados, si encuentran errores haganmelo saber. (tendrán 1 punto en la tarea por cada error que encuentren, y 2 si solucionan el error)

# Algoritmos y estructuras de Datos
# P2PU

## Creación de una estructura

Para crear una estructura, se define con la siguiente sintaxis:

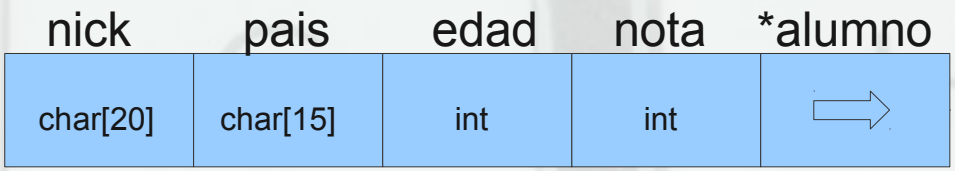
```
struct nombre_de_la_estructura {
        campos de estructura;
};
```

Ejemplo:

```
struct alumno{                      //definimos el nombre de la estructura
        char nick[20];              //definimos un arreglo char de tamaño 20
        char pais[15];              //definimos un arreglo char de tamaño 15
        int edad;                   //definimos un entero
        int nota                    //definimos un entero
        struct alumno *siguiente; //definimos un puntero de tipo struct persona llamado
                                    siguiente, esto significa que apuntará a otra estructura de las
                                    mismas características
};
```

# Algoritmos y estructuras de Datos
# P2PU

Es decir graficamente nos queda algo como esto:




Aun no hemos creado ningún dato, solo hemos definido el "*molde*", de la estructura que usaremos, que llamamos "*alumno*", el cual está compuesta por: char[20], char[15], int, int, *alumno.

# Algoritmos y estructuras de Datos
# P2PU

Ahora crearemos 3 punteros que nos ayudará a identificar la cabeza de la lista y otros dos que nos ayudará a desplazarnos por ésta, que llamaremos p y q.

```
struct alumno *head, *p, *q;
```

Con esto ya estamos preparados para crear las funciones para trabajar con nuestra lista.

Hasta el momento tenemos el siguiente código:

```
struct alumno{                      //definimos el nombre de la estructura
        char nick[20];              //definimos un arreglo char de tamaño 20
        char pais[15];              //definimos un arreglo char de tamaño 15
        int edad;                   //definimos un entero
        int nota                    //definimos un entero
        struct alumno *siguiente; //definimos un puntero de tipo struct persona llamado
                                    siguiente, esto significa que apuntará a otra estructura de las
                                    mismas características
};

struct alumno *head, *p, *q;
```

# Algoritmos y estructuras de Datos
# P2PU

## Agregar una nueva estructura a la lista

Como ya tenemos nuestro molde de estructura, haremos la función para agregar una estructura, el código es el siguiente:

```
void agregar(char nick[20], char pais[15], int edad, int nota) //recibimos los parámetros
{
	if(head==null) //Caso donde no existe ningun elemento en la lista
	{
		p = (struct alumno *) malloc(sizeof(struct alumno)); //creo una estructura
		strcpy(p->nick,nick); //copia el dato en la estructura (usar libreria stdlib.h)
		strcpy(p->pais,pais); //copia el dato en la estructura (usar libreria stdlib.h)
		p->edad=edad; //copia el dato edad
		p->nota=nota; //copia el dato nota
		p->siguiente=null; //el puntero lo apunta a null, diciendo que es el último elemento
		q=p; //el puntero q apunta a la estructura creada
		head=p; //el puntero head apunta a p.
	}else // Por lo menos hay un elemento en la lista
	{
		p = (struct alumno *) malloc(sizeof(struct alumno)); //creo una estructura
		strcpy(p->nick,nick); //copia el dato en la estructura (usar libreria stdlib.h)
		strcpy(p->pais,pais); //copia el dato en la estructura (usar libreria stdlib.h)
		p->edad=edad; //copia el dato edad
		p->nota=nota; //copia el dato nota
		q->siguiente=p;
		p->siguiente=null; //el puntero lo apunta a null, diciendo que es el último elemento
		q=p; //el puntero q apunta a la estructura creada
	}
}
```

# Algoritmos y estructuras de Datos
# P2PU

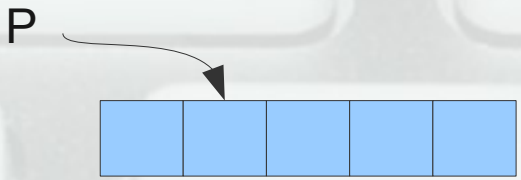
Muy bien, ahora que tenemos el código lo analizaremos paso a paso:

Primero que todo hay dos opciones como se dieron cuenta, el primero es para saber si existe algún elemento en la lista, esto nos sirve exclusivamente para insertar el head (**if**(head==null)).

**Paso1:**

**Código:** p = (struct alumno *) malloc(sizeof(struct alumno)); //creo una estructura
**Explicación:** Lo que hacemos aquí es reservar memoria y crear una estructura de tipo alumno, o sea creamos lo siguiente, con la sintaxis ya conocida, char[20],char[15],int,int,*alumno:

Luego lo igualamos a p, esto quiere decir que el puntero “p” apunta a esta nueva estructura creada:



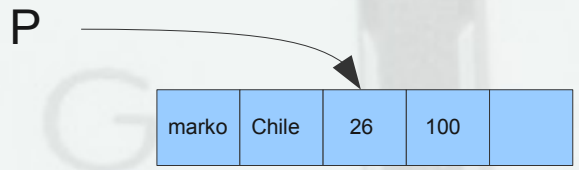

No nos compliquemos, tan solo reservamos memoria, creamos una estructura y le decimos al puntero p que lo apunte, aun no hacemos nada más.

# Algoritmos y estructuras de Datos
# P2PU

**Paso2:**

**Código:** strcpy(p->nick,nick); //copia el dato en la estructura (usar libreria stdlib.h)
strcpy(p->pais,pais); //copia el dato en la estructura (usar libreria stdlib.h)
p->edad=edad; //copia el dato edad
p->nota=nota; //copia el dato nota

**Explicación:** Copiamos todo los datos pasados en nuestra función agregar, a la estructura resultando algo como esto, por ejemplo si llamamos a agregar("marko","Chile",26,100), nos queda algo asi:




**Paso3:**

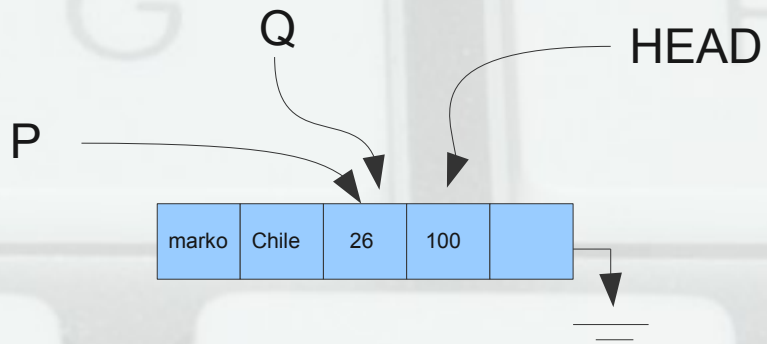
**Código:** p->siguiente=null; //el puntero lo apunta a null, diciendo que es el último elemento
**Explicación:** Decimos que la estructura que apunta P, apunte con su puntero *siguiente a null, o sea:

# Algoritmos y estructuras de Datos
# P2PU

**Paso4:**

**Código:** q=p; //el puntero q apunta a la estructura creada
head=p; //el puntero head apunta a p.

**Explicación:** Finalmente apuntamos Q y HEAD a la estructura donde está apuntando P



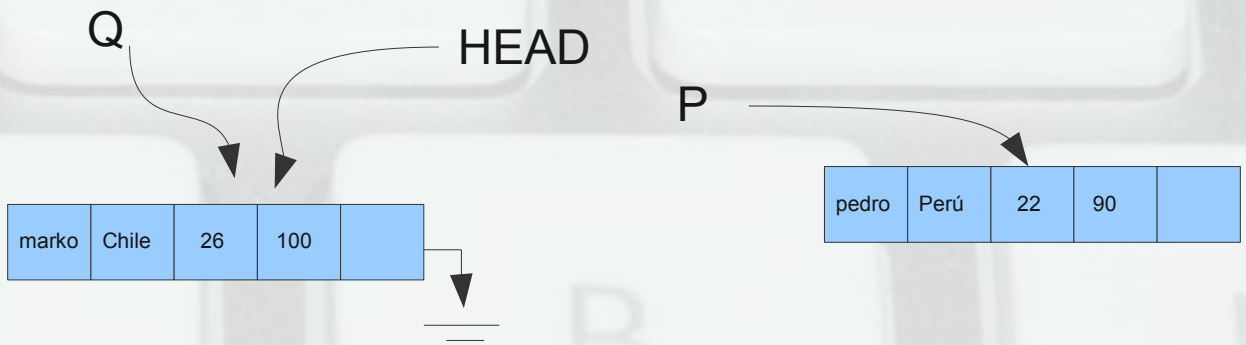
# Algoritmos y estructuras de Datos
# P2PU

Ahora crearemos una estructura cuando hay por lo menos un elemento en la lista:

**Paso1:**

**Código:**
```
p = (struct alumno *) malloc(sizeof(struct alumno)); //creo una estructura
strcpy(p->nick,nick); //copia el dato en la estructura (usar libreria stdlib.h)
strcpy(p->pais,pais); //copia el dato en la estructura (usar libreria stdlib.h)
p->edad=edad; //copia el dato edad
p->nota=nota; //copia el dato nota
```

**Explicación:** Creamos la estructura P y le ingresamos los datos, por ejemplo llamamos a agregar("pedro","Perú",22,90):

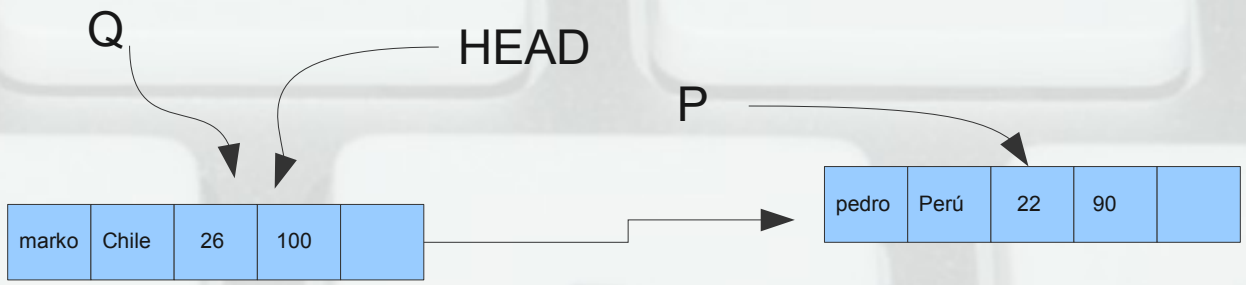


# Algoritmos y estructuras de Datos
# P2PU

**Paso2:**

**Código:**q->siguiente=p;

**Explicación:** Decimos que lo que esté apuntando Q (la estructura con el usuario “marko”) apunte ahora con su puntero *siguiente a P (la estructura con el usuario “pedro”), entonces nos queda lo siguiente:

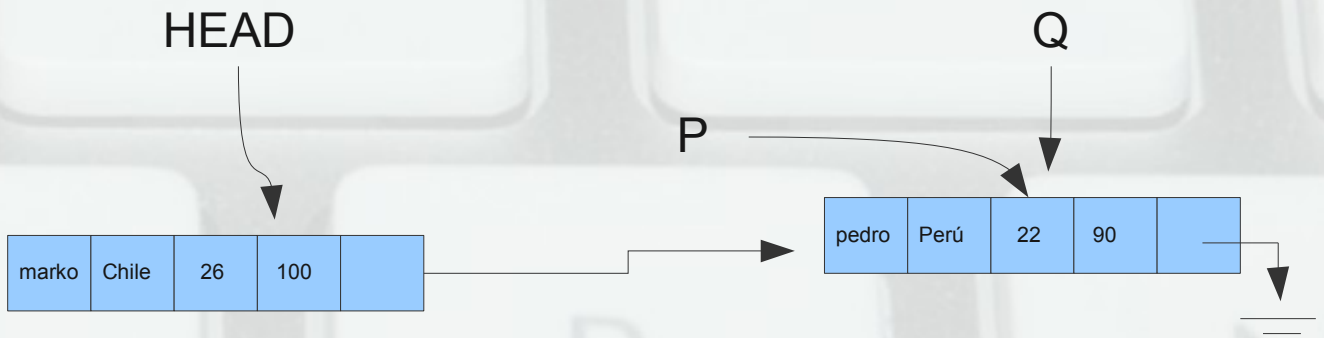


# Algoritmos y estructuras de Datos
# P2PU

**Paso3:**

**Código:**p->siguiente=null; //el puntero lo apunta a null, diciendo que es el último elemento
q=p; //el puntero q apunta a la estructura creada

**Explicación:** Decimos que lo que esté apuntando P (la estructura con el usuario “pedro”) apunte ahora con su puntero a **null**, y luego que Q es igual a P, o sea Q (no el puntero de Q) apuntará a lo que apunta P.



# Algoritmos y estructuras de Datos
# P2PU

### Recorrer una lista

Ahora que ya sabemos como crear una lista podemos hacer lo que queramos con ella, buscar, modificar o eliminar datos de nuestra lista, para esto primero debemos saber como recorrerla, para esto usamos el siguiente codigo:

Primero debemos crear un puntero de forma global:

```
struct alumno *temp;
```

Luego para recorrerla usaremos un for:

```
for(temp=head;temp->siguiente!=null;temp=temp->siguiente)
{
}
```

**Condición inicial:** temp=head, o sea el puntero temp apuntará a la cabeza de la lista.
**Condición final:** temp->siguiente!=null, es decir se ejecuta mientras no sea el ultimo
**Iteración:** temp=temp->siguiente**,** para cada iteración avanza una estructura en la lista.

Con esto podremos crear las funciones de buscar, eliminar y modificar.

# Algoritmos y estructuras de Datos
# P2PU

## Buscar, eliminar y modificar

Ahora como ya sabemos recorrer una lista implementaremos la función buscar, por ejemplo imprimiremos en pantalla el pais de algún alumno en particular:

```
struct alumno *temp;

void buscar_pais(char nombre[20])
{
	for(temp=head;temp->siguiente!=null;temp=temp->siguiente)
	{
		if(temp->nombre==nombre)
			printf("El pais de orígen de %s es %s",nombre,temp->nombre);
			break;
	}
}
```

Si queremos modificar algo es similar, por ejemplo queremos ubirle la nota a pedrito de un 90 a un 100:

```
struct alumno *temp;

void modificar_nota(char nombre[20], int nota)
{
	for(temp=head;temp->siguiente!=null;temp=temp->siguiente)
	{
		if(temp->nombre==nombre)
			temp->nota=nota;
			break;
	}
}
```

# Algoritmos y estructuras de Datos
# P2PU

Y para eliminar un alumno por que no hizo las tareas de nuestro curso, usamos el siguiente código, tuve que usar un while:

```
struct alumno *temp,*temp2;

void eliminar_alumno(char nombre[20])
{
	temp=head;
	temp2=head;
	while(temp!=null){
		if(temp2==head){
			if(head->nombre==nombre){
				head=head->siguiente;
				free(temp);
				temp=head;
				temp2=head;
			}else{
				temp2=head->siguiente;
			}

		}else{
			if(temp2->nombre==nombre){
				temp->siguiente=temp2->siguiente;
				free(temp2);
			}else{
				temp = temp->siguiente;
				temp2 = temp2->siguiente;
			}
		}
	}
}
```

# Algoritmos y estructuras de Datos
# P2PU

Por favor guarden el código de eliminar, que para ser sincero me costó un poco hacerlo... :p

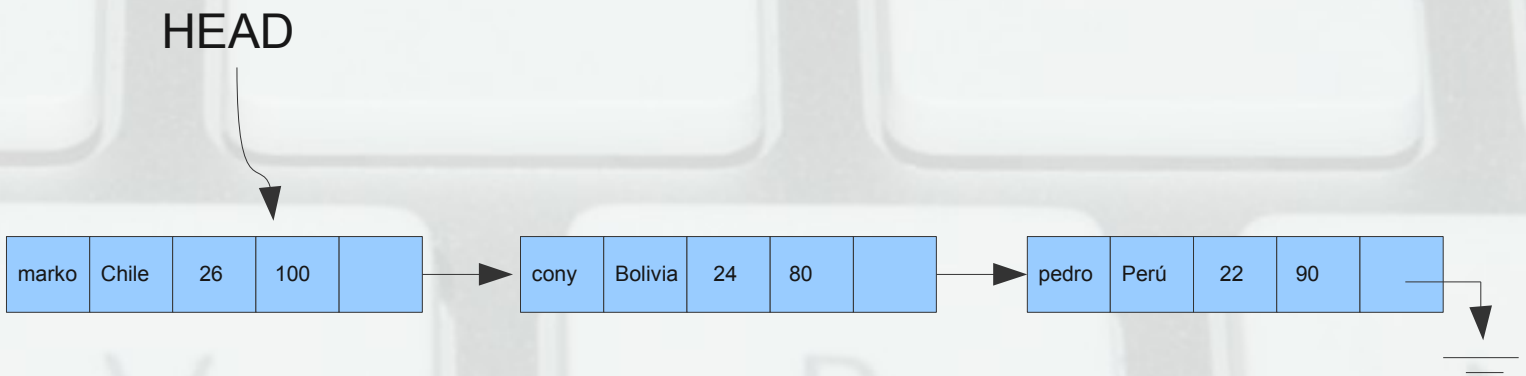
Ahora lo explicaré con lujo de detalles, ustedes pueden implementarlo como quieran en todo caso:

Usé dos punteros, temp y temp2 y un while en vez del for anterior, no se si se puede hacer de otra forma más simplificado, mmm creo que lo pondré en una de las discusiones para que investiguen por mi... en fin seguimos con lo nuestro.

**struct** alumno *temp,*temp2;

void eliminar_alumno(char nombre[20]) // este es el nombre de la función y pasamos como parámetro el nombre del alumno a eliminar.

Hagamos cuenta de que tenemos la siguiente lista:




Eliminaremos los dos casos cuando está en el HEAD y cuando no

# Algoritmos y estructuras de Datos
# P2PU

temp=head; //Primero hacemos que temp se pare en nuestro HEAD, para comenzar obviamente del principio
temp2=head; //Y hacemos que temp2 se pare en nuestro HEAD.
while(temp!=null) //Hacemos un while con la condición de que el temp sea distinto de null, y haremos avanzar el temp

Primero eliminaremos a “marko”, en este caso como podemos apreciar coincide con HEAD, por lo tanto entraremos a la primera condición (en las figuras eliminaré las flechas de los temp y el head para que caiga todo):

```
if(temp==head) // preguntamos si el temp está parado en head, es asi, asi que entramos
{
        if(head->nombre==nombre) //comparamos el nombre del HEAD con nuestra entrada en la función
        {
                if(head->nombre==nombre){
                        head=head->siguiente; (figura 1)
                        free(temp);(figura 2)
                        temp=head; (figura 3)
                        temp2=head; (figura 3)
                }else
```

**Figura 1**

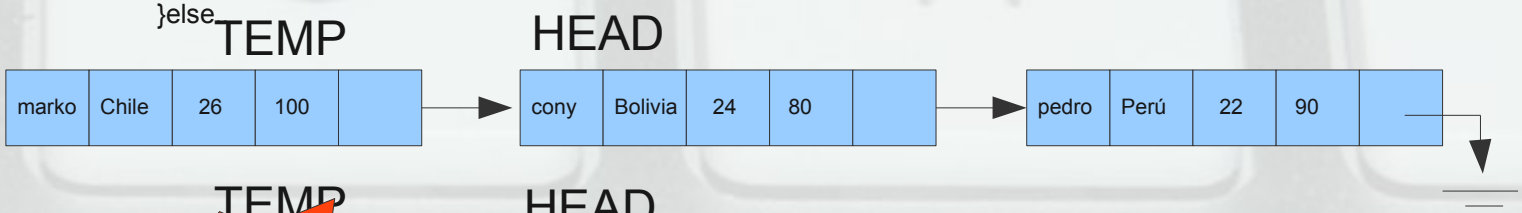



**Figura 2**

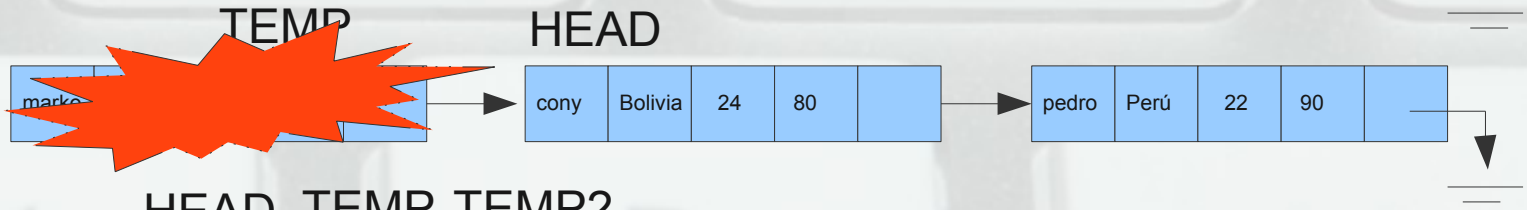
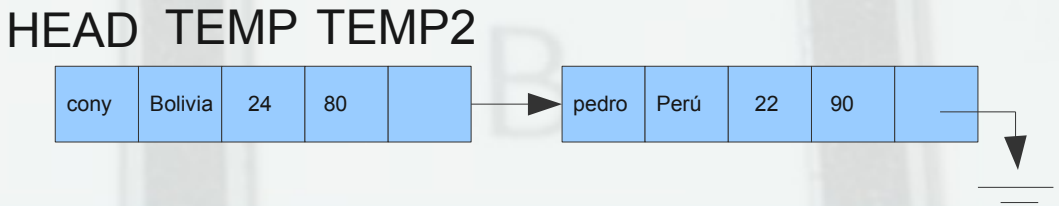
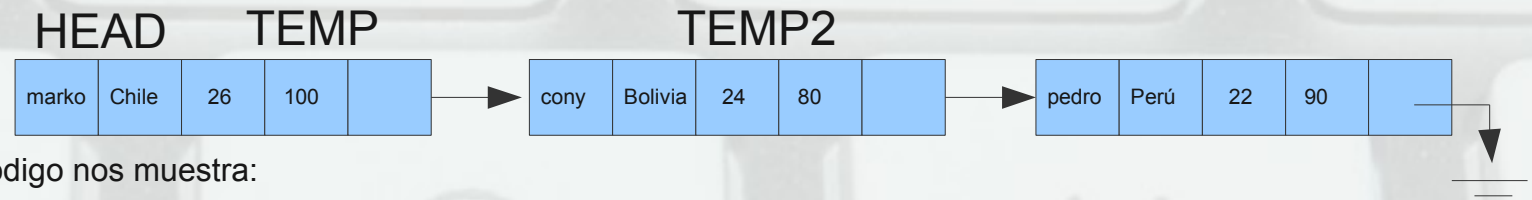



**Figura 3**



# Algoritmos y estructuras de Datos
# P2PU

Y asi eliminamos a marko, ahora eliminaremos en vez de a marko a cony con el siguiente código, ya que al preguntar la primera vez si *temp == head* entra aquí, pero pasa a *temp2=head->siguiente*, ya que no es el que buscamos, por ende tenemos lo siguiente:




El código nos muestra:

```
if(temp2->nombre==nombre){ //si el nombre de temp2 es igual a nuestro dato, en este caso lo es
        temp->siguiente=temp2->siguiente; //decimos que el siguiente del temp apunte al siguiente del temp2 (figura4)
        free(temp2); // Eliminamos temp2 (figura5)
}else{...
```

**Figura 4**

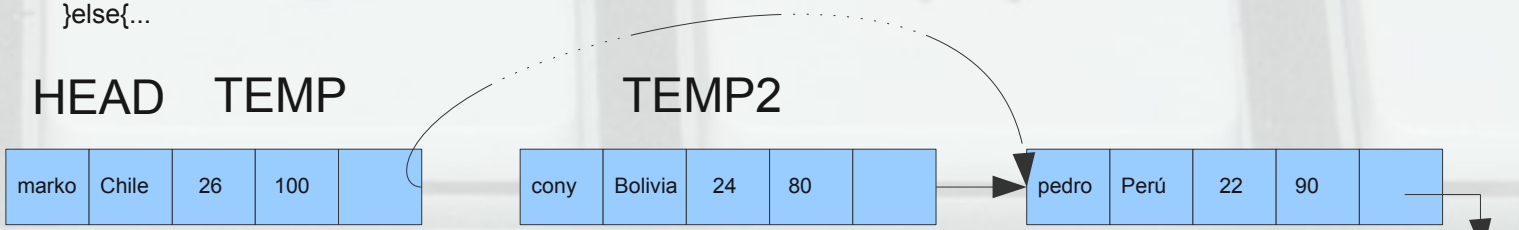
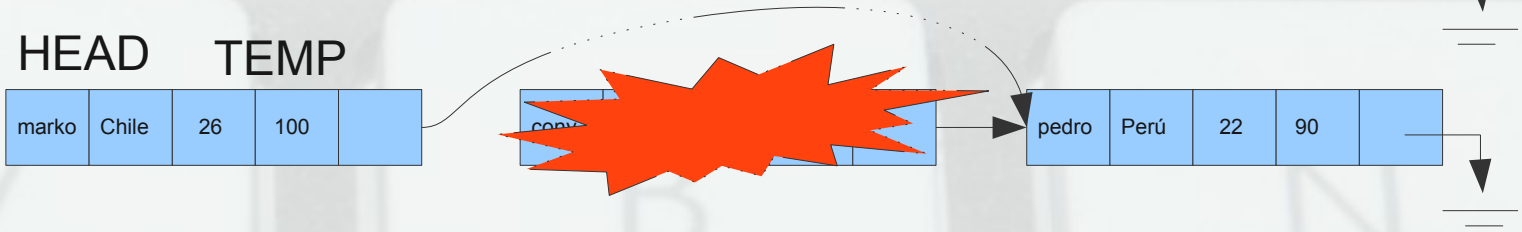



**Figura 5**



# Algoritmos y estructuras de Datos
# P2PU

## Listas doblemente enlazadas

La diferencia con las listas normales, es que las doblemente enlazadas tienen 2 punteros, uno hacia adelante y otro hacia atras, **entonces será mucho más fácil trabajar con ellas**.

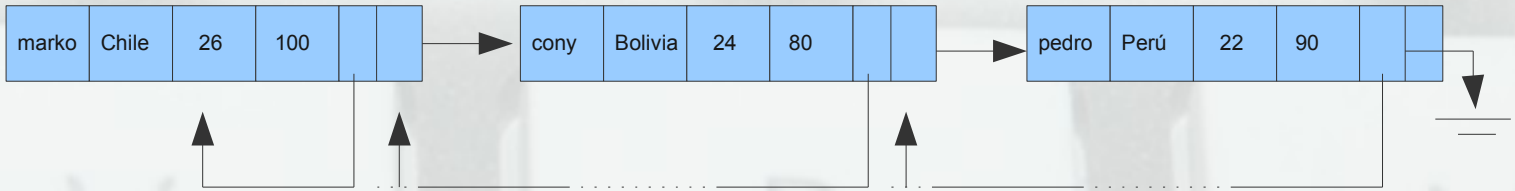
La estructura sería de la siguiente forma:

```
struct alumno{                  //definimos el nombre de la estructura
        char nick[20];          //definimos un arreglo char de tamaño 20
        char pais[15];          //definimos un arreglo char de tamaño 15
        int edad;                       //definimos un entero
        int nota                //definimos un entero
        struct alumno *siguiente; //definimos un puntero siguiente
        struct alumno *anterior; //definimos un puntero siguiente

};
```

Y se vería algo asi:

HEAD



# Algoritmos y estructuras de Datos
# P2PU

## Listas circulares

La única diferencia de las listas circulares es que la última estructura en vez de apuntar a null apunta a head, el código anteriormente escrito en esta clase es una pauta muy clara para la codificación de este tipo de listas, no lo daré porque es parte de la tarea, la lista circular entonces se vería algo como esto:

# Algoritmos y estructuras de Datos
# P2PU

**Tarea:** Un aeropuerto ha contratado a los alumnos del curso de algoritmos y estructura de datos P2PU, ya que han demostrado ser muy inteligentes, para crear un sistema que permita ver la llegada y salida de su flota, para esto su profesor (o sea yo) ha capturado los siguientes requisitos:

**Requerimientos Generales:** Se debe crear un sistema en el lenguaje C en el cual cree una cola doblemente enlazada y circular, que este vacía, de los aviones que saldrán en la única pista del aeropuerto, además el sistema debe cumplir con los siguientes requisitos:

Mostrar un menú principal que indique las acciones que se pueden realizar, estas son: Insertar un nuevo avión en la cola (debe permitir ingresar todos los datos), eliminar un avión de la cola (cancela el vuelo, insertando el código del vuelo), modificar información del vuelo (salta a un submenú preguntando cual dato del vuelo, se quiere modificar, insertando el código del vuelo, que es lo único que no se puede modificar), despegue (eliminar el avión que está primero), mostrar la cola (se debe mostrar el código del vuelo con el siguiente formato: codigo1->codigo2->codigo3), mostrar información del vuelo (mostrar todos los datos del vuelo, segun el código), salir (sale del sistema). Además el menú debe poder accederse siempre que se requiera, ejemplo si estoy en un submenú debe decir algo como volver al menú principal.

La estructura necesaria es la siguiente:

```
struct vuelos
{
        int codigo_vuelo;
        int cantidad_pasajeros;
        char ciudad_destino[30];
        char hora_salida[10];
        char aerolinea[30];
        char modelo_avion[20];
        char nombre_piloto[40];

};
```

Suponga que los datos ingresados están ordenados por hora.

# Algoritmos y estructuras de Datos
# P2PU

La tarea será grupal de 2 o tres personas, esta debe ser enviada a mi correo electrónico (markogonzalez84@gmail.com), a más tardar a las 23:59 del día Miércoles 2 de Marzo de 2011, hora de Chile, en formato pdf.

Deben crear una portada con el nombre y nick de cada uno(P2PU), un pequeño informe donde se muestre la problemática y los diagramas de flujo de cada una de las funciones echas, letra Arial 16, para títulos y 12 para texto, interlineado 1.5 y justificado.

**Tarea Opcional:** No tome el supuesto de que los datos ingresados estan ordenados, es decir si tengo la siguiente lista con los códigos de vuelo: A->B->C, donde las horas de salida son:

A = 10:00
B = 11:00
C = 12:00

Y quiero ingresar un vuelo D=11:30, automáticamente se inserte donde corresponde, o sea quede:

A->B->D->C

# Algoritmos y estructuras de Datos
# P2PU

Preguntas para debatir en los foros:

1.- ¿Porqué es más facil trabajar con listas doblemente enlazadas y circulares?

2.- ¿Cuál es la principal ventaja de crear una lista circular?

**Frase de la semana:** ”Un buen trabajo en equipo de personas comunes es más fuerte que varios genios trabajando individualmente”.