Creación de una lista dinámica.

Programación I – Laboratorio I.
Tecnicatura Superior en Programación.
UTN-FRA

Autores: *Ing. Ernesto Gigliotti*

Revisores: *Lic. Mauricio Dávila*

Versión : 1

Índice de contenido

1 Creación de una lista dinámica

1.1 Generación dinámica de memoria

La función *malloc* nos permite obtener un espacio de memoria en tiempo de ejecución (cuando el programa se ejecuta y se ejecuta esta función, el sistema operativo asigna la porción de memoria al programa). Este comportamiento nos permitirá la generación dinámica de variables. En el siguiente ejemplo definimos un *array* de una cantidad determinada de variables del tipo *int*:

```
int numeros[5];
```

Si necesitamos que el tamaño del *array* sea definido en tiempo de ejecución (por ejemplo si le preguntamos al usuario cuántos números va a ingresar, y luego pretendemos guardar los números que ingresa en dicho *array*) esta solución "estática" de definir un *array* de 5 elementos, no nos serviría, ya que si el usuario decide ingresar 6 o más números, no habría posibilidad de almacenarlos.

El siguiente ejemplo nos permitiría solucionar el problema:

```
int cantidad = pedirCantidadAlUsuario();
int* numeros;
numeros = (int*)malloc(sizeof(int)*cantidad);
```

De esta manera reservamos espacio para una cantidad de variables del tipo *int*, según la variable cantidad, la cual se carga con el número que devuelve una supuesta función que le pide ingresar un número al usuario. Es importante destacar que no podemos hacer lo siguiente:

```
int numeros[cantidad];
```

Ya que las definiciones de *arrays* estáticos deben realizarse con un valor conocido en tiempo de compilación, y el valor de la variable *cantidad* se conoce en tiempo de ejecución.

1.2 Creación de tipos de datos particulares

La generación de tipos de datos particulares se realiza mediante estructuras, en las cuales se pueden definir campos indicando su tipo y nombre, los cuales en conjunto forman un nuevo tipo de dato que el programador no tenía en el lenguaje puro (*int, double, char,* etc.)

En el siguiente ejemplo se observa la creación del tipo de dato *Persona* el cual está compuesto por un nombre y una edad

```
struct S_Persona
{
	int edad;
	char nombre[20];
}Persona;
```

Cuando se define de forma estática una variable del tipo *Persona*, dicha variable ocupa en memoria la cantidad de *bytes* que ocupa un *int* más 20 *chars*.

Para generar de manera dinámica una variable del tipo *Persona*, simplemente cambiamos la manera de calcular la cantidad de *bytes* que le pasamos a la función *malloc.*

1.3 Utilización de memoria dinámica para la creación de tipos de datos

Un vez más, utilizamos memoria dinámica cuando no sabemos la cantidad de variables que vamos a crear de antemano, sino que dicho valor se define en tiempo de ejecución. Imaginemos un programa donde se le pide al usuario que ingrese un nombre y una edad, de forma indefinida hasta que decida salir, si lo planteamos sin la utilización de memoria dinámica:

```
do {
      Persona persona;
      preguntarNombre(&persona->nombre);
      persona.edad = preguntarEdad();

      // aquí se debe hacer algo con la variable Persona creada, ya que en el
      // próximo loop, se pisarán los datos y los anteriores se perderán.

}while(preguntarSalir()!='S');
```

Para solucionar el problema, podemos tener un *array* estático de variables *Persona* definido con anterioridad:

```
Persona lista[20];
```

De esta manera, después de cargar el nombre y la edad, podemos copiar la variable "persona" a un ítem del *array* llamado "lista" e incrementar el índice.

```
lista[index] = persona;
index++;
```

¿Qué ocurre cuando el usuario quiere ingresar la persona numero 21? no existirá lugar para guardar dicha persona, por lo que la solución "estática" al problema, no es una solución después de todo.

La ventaja de la utilización de memoria dinámica nos permite crear la variable *persona* en cada iteración del bucle, de modo que a medida que las necesitamos, las iremos creando. Necesitaremos crear de forma dinámica la variable *persona* dentro del bucle, y también el *array* llamado "lista", el cual tendrá un valor inicial, y luego haremos que crezca su tamaño mediante la función *realloc*.

1.4 Solución al problema del *array* estático mediante memoria dinámica

Comenzaremos creando el *array* de forma dinámica:

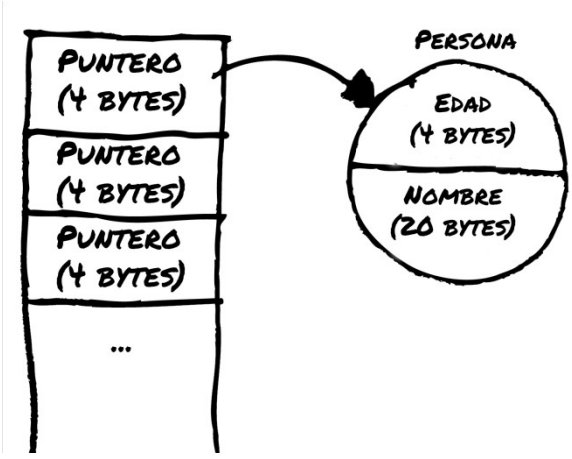
```
int size = 10;
int index=0;
Persona* lista = (Persona*)malloc(sizeof(Persona)*size);
do {
	Persona persona;
	preguntarNombre(&persona->nombre);
	persona.edad = preguntarEdad();


	lista[index] = persona;
	index++;
	if(index>=size)
	{
		// incrementamos el tamaño del array
		size+=10;
		lista = realloc(lista,sizeof(Persona)*size);
	}
}while(preguntarSalir()!='S');
```

1.4.1 Desventajas de este modelo

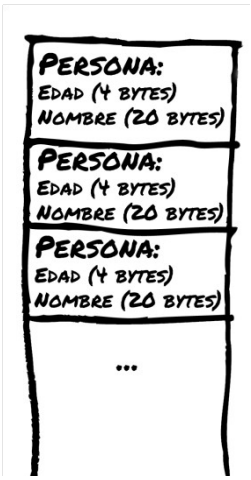
Cuando incrementamos el tamaño de la lista, estamos desperdiciando mucho espacio, ya que lo incrementamos en 10 (se podría haber multiplicado por dos el tamaño, o usar otro algoritmo de crecimiento) por lo que generamos 10 posiciones del tamaño del tipo de dato *Persona*, el cual en nuestro caso tiene la edad (4 *bytes*) y el nombre (20 *bytes*), por lo que desperdiciamos mucho espacio sin saber si se va a utilizar o no.

La solución a este problema, es no tener una lista en donde cada ítem sea una variable del tipo *Persona*, sino tener una lista donde cada ítem sea un **puntero** a una variable del tipo *Persona*, es decir, un *array* de punteros. Como cada puntero ocupa 4 *bytes* (en arquitecturas de 32 *bits*) o 8 *bytes* (en arquitecturas de 64 *bits*), cada ítem ocupará menos espacio que si reservamos el valor para guardar variables *Persona*.

Gráfico comparativo lista de variables y lista de punteros



Array de punteros



Array de tipos Persona

1.5 Solución al problema con lista de punteros

Modificaremos el ejemplo anterior, para crear en forma dinámica una lista de punteros del tipo *Persona*:

```
int size = 10;
int index=0;
Persona** lista = (Persona**)malloc(sizeof(Persona*)*size);
do {
      Persona persona;
      preguntarNombre(&persona->nombre);
      persona.edad = preguntarEdad();


    lista[index] = &persona; // Ver explicacion sobre esta línea
    index++;
    if(index>=size)
    {
       // incrementamos el tamaño del array
       size+=10;
       lista = realloc(lista,sizeof(Persona*)*size);
    }

}while(preguntarSalir()!='S');
```

Como se observa en el ejemplo, ahora nuestra lista es de punteros a variables del tipo *Persona*. Luego de cargar el nombre y la edad en la variable auxiliar *persona*, almacenamos el puntero a esta variable dentro de la lista de punteros:

```
lista[index] = &persona;
```

Es importante destacar, que **esta solución no es correcta**, debido a que la variable auxiliar *persona* tiene siempre la misma dirección y en las diferentes iteraciones se asignan diferentes valores a sus campos. Si se deja el código como en el ejemplo, la lista se cargaría con punteros hacia la misma posición de memoria, donde se encuentra nuestra variable auxiliar, y los sucesivos datos que son cargados se perderían.

Para solucionar este problema, debemos crear un espacio de memoria para una variable *Persona*, cada vez que el usuario ingresa un nuevo ítem en la lista. Esto es posible mediante el uso de *malloc*:

```
      Persona* persona = (Persona*)malloc(sizeof(Persona));
      preguntarNombre(persona->nombre);
      persona->edad = preguntarEdad();

      lista[index] = persona;
```

Ahora la variable auxiliar no es una variable del tipo *Persona*, sino un puntero a dicha variable, la cual se genera reservando memoria en forma dinámica. Luego de cargar los datos en dicha variable, se copia el puntero a la variable, a la lista. En la próxima iteración, se generará otro espacio de memoria para una nueva variable *Persona*, para guardar los nuevos datos ingresados, por lo que no corremos riesgo de que se pisen con datos anteriores.

Gráfico indicando un *array* de punteros del tipo *Persona*, donde cada ítem del *array* apunta a un espacio de memoria del tamaño de una variable del tipo *Persona* creado con **malloc**.