

# Programación en Pascal. Memoria dinámica

Escuela Técnica Superior de Ingeniería de Telecomunicación  
Universidad Rey Juan Carlos

gsync-profes (arroba) gsync.urjc.es

Diciembre de 2018



©2018 GSyC

Algunos derechos reservados.  
Este trabajo se distribuye bajo la licencia

Creative Commons Attribution Share-Alike 4.0

- 1 Introducción
- 2 Punteros
- 3 Listas encadenadas

# Formas de almacenamiento de datos en memoria

Hay tres formas de almacenar datos en la memoria del ordenador

- Variables y constantes globales, también llamadas *estáticas*  
Ya las conocemos
- Variables y constantes locales, también llamadas *automáticas*  
Ya las conocemos
- Memoria dinámica  
Dedicaremos este tema a presentar una introducción a la memoria dinámica

# Memoria estática

Como sabemos, se usa de dos formas

- Variables globales. Muy peligrosas. En este curso, *supenso seguro*
- Constantes globales. Las usamos en ocasiones: para el tamaño de un array o para magnitudes físicas o matemáticas universales

A las variables y constantes globales también se las llama *estáticas*, porque se pueden usar en todo el programa, *viven* en memoria durante toda la vida del programa

# Memoria automática

- Variables locales  
De un procedimiento, de una función o del cuerpo principal de un programa
- Parámetros de funciones y procedimientos  
Su ámbito está restringido a un subprograma (función o procedimiento), no se pueden usar fuera de su ámbito.

A estas variables y parámetros también se les llama *variables automáticas*, porque se crean en memoria automáticamente cuando el programa entra en su subprograma, y se liberan automáticamente al abandonar el subprograma

Tanto en la memoria estática como en la memoria automática, teníamos una limitación muy severa: antes de usar una variable, constante o parámetro teníamos que fijar y limitar el tamaño que ocuparía

- Un número real, un boolean, un entero, etc almacenan exactamente 1 valor
- Una cadena (en nuestro dialecto de Pascal, Object Pascal sin directivas adicionales) permite un máximo de 255 caracteres
- En un array el programador indica el número de posiciones. Ese espacio queda reservado y ocupado, no importa si se usa o no
  - Si es necesario almacenar más datos de los previstos inicialmente, no se puede
  - Si no se ocupa todo el array con valores útiles, se desperdicia memoria

# Memoria dinámica

La *memoria dinámica* es una técnica que permite que el programador no indique en el código fuente del programa cuánto espacio ocupará un dato o estructura de datos, si no que en *tiempo de ejecución*, esto es, *sobre la marcha*, el propio programa va ocupando y liberando la memoria que necesita. Esto puede gestionarlo

- El programador
- El lenguaje de programación

- En muchos lenguajes de programación, el manejo de la memoria dinámica es responsabilidad del programador. Esto tiene cierta dificultad y es propenso a errores, aunque se puede conseguir un código muy eficiente, muy rápido, que consume muy poca memoria.
  - Es típico de lenguajes de los años 1970 y 1980 como C y Pascal, aunque hay muchas excepciones
- En otros lenguajes de programación, el propio compilador (o intérprete) se ocupa de gestionar la memoria dinámica. Esto los hace más fáciles de usar y con menos errores. Aunque el código suele ser menos eficiente.
  - Es típico de lenguajes de los años 1990 y posteriores como python, java y javascript. Aunque hay muchas excepciones

# Punteros

Usando memoria dinámica, para manejar un valor ya no tratamos con una única entidad, una *variable*, sino con dos: *punteros* apuntando a *datos*

- El puntero no contiene el dato directamente, sino la dirección de memoria donde está el dato. El puntero *apunta al dato*
- A partir del puntero se obtiene el dato, a esto se llama *aplicar una indirección*, o, coloquialmente, *atravesar el puntero*

- En los lenguajes como C y Pascal, el programador es responsable de
  - Reservar memoria para un dato
  - Asegurarse de que el puntero apunta a una zona de memoria correctamente reservada
  - Liberar la memoria cuando ya no sea necesaria
- En lenguajes como python, java y javascript, muchas de estas tareas se hacen automáticamente
  - Por ejemplo, la memoria la libera *de vez en cuando* el *recolector de basura*. Pero el hecho de que este recolector se ejecute sin control directo del programador puede ser un inconveniente

# Listas encadenadas

Hay muchas formas de manejar memoria dinámica, la más sencilla es la *lista encadenada*, también llamada *lista enlazada*. Es una estructura similar a una cadena (de eslabones, no de texto).

Está formada por nodos

- Se inicia la lista como una cadena vacía
- Cada nodo contiene información de un elemento (típicamente un registro) y apunta al nodo siguiente
- Se añaden uno a uno los nodos necesarios
- Cuando ya no son necesarios, se liberan. Sería equivalente a quitar eslabones de la cadena metafórica

# Declaración e inicialización

```

{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program punteros_01;

type
  TipoLista = ^TipoNodo; // TipoLista es un puntero a un nodo

  TipoNodo = record
    valor: integer; // El valor que nos interesa
    sig: TipoLista; // Puntero al siguiente nodo
  end;

procedure inicia_lista(var lista:TipoLista);
begin
  lista := nil
end;

```

# Adición de nuevo elemento

```
procedure aniade_elemento(var lista:TipoLista; valor:integer);
var
  nodo : TipoLista;
begin
  new(nodo);           // Reservamos memoria para un nuevo nodo
  nodo^.valor := valor; // Le damos su valor
  nodo^.sig := lista;  // Enlazamos el nodo a la cadena
  lista := nodo        // Ahora la lista empieza por el nuevo nodo
end;
```

# Procesamiento de la lista

```
procedure recorre_lista(var lista:TipoLista);  
begin  
  while (lista <> nil) do begin // Mientras no estemos en el fin  
    writeln(lista^.valor);      // Procesamos el nodo  
    lista := lista^.sig        // Pasamos al siguiente elemento  
  end  
end;
```

Este procesamiento recorre la lista en orden inverso al de creación: el primer elemento en entrar es el último en salir

# Liberación de memoria

```
procedure libera_lista(var lista:TipoLista);
var
  previo : TipoLista;
begin
  while (lista <> nil) do begin
    previo := lista;           // Guardamos el elemento anterior
    lista := lista^.sig;      // La lista apunta al siguiente
    dispose(previo)           // Borramos el anterior
  end
end;
```

# Uso de los procedimientos anteriores

```
var
  lista : TipoLista;
begin
  inicia_lista(lista);
  aniade_elemento(lista, 1);
  aniade_elemento(lista, 2);
  aniade_elemento(lista, 3);

  recorre_lista(lista);
  libera_lista(lista);
end.
```

Código fuente completo:

[https://gsync.urjc.es/~mortuno/fpi/punteros\\_01.pas](https://gsync.urjc.es/~mortuno/fpi/punteros_01.pas)

# Recomendaciones

Para un estudiante de introducción a la programación, el uso de memoria dinámica tiene cierta complejidad. Para evitar errores, reutiliza este código con los mínimos cambios imprescindibles

- Usa las mismas funciones y tipos de datos, puedes usar exactamente los mismos nombres
- Límitate a cambiar el campo *valor* de *Tiponodo*
  - Para estructuras sencillas, reemplaza *valor* por los campos que necesites
  - Para estructura un poco más complejas, sustituye *valor* por un tipo registro, que contenga los campos necesarios
- En el procedimiento *recorre\_lista*, reemplaza el *writeln* por la llamada al subprograma que corresponda