Tema 4.
Gestión dinámica de la memoria

Diseño de Algoritmos.

E.U. Politécnica
I.T.Industrial (Electricidad)

M. Carmen Aranda Garrido
Despacho: I-307

Gestión dinámica de la memoria

Contenidos:

1. Introducción.
2. Punteros y operaciones básicas con punteros.
3. Asignación y liberación dinámica de memoria.
4. Arrays dinámicos.
5. Listas enlazadas con punteros.

1. Introducción.

- Variables estáticas:
  - Se conoce su nombre.
  - Se conoce cuando empieza/acaba su existencia.
  - Se conoce el espacio que ocupan en memoria.

Se Crean en Tiempo de Compilación

1. Introducción.

Problema

¿Que sucede si a priori no conocemos la cantidad de espacio de almacenamiento que vamos a precisar?

Solución ⇔ Hacer una previsión??

Ejemplo:

```
Tipos
   struct persona {char nombre[30];
                   int edad;}
Variables
    struct persona poblacion[30];
```

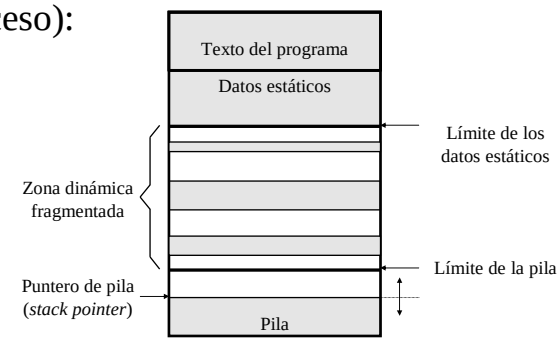
1. Introducción.

- Datos dinámicos:
  - No se conoce el momento en que empieza/acaba su existencia.
  - El espacio que van a ocupar en memoria es variable.

Se Crean en Tiempo de Ejecución

1. Introducción.

- Memoria asignada a un programa en ejecución (proceso):



1. Introducción.

- Para trabajar con datos dinámicos necesitamos:
  - Funciones para asignar y liberar memoria.
  - Punteros para acceder a los datos dinámicos.

2. Punteros.

¿Que Es un Puntero?

Variable Estática

Un puntero es una variable que almacena una dirección de memoria

2. Declaración de Punteros.

```
Tipos
  typedef int *ptrint;
/*puntero a enteros*/
Variables
  ptrint p;
/*puntero a enteros*/
Variables
  int *p;
/*puntero a enteros*/
```

1ª FORMA

2ª FORMA

2. Declaración de Punteros

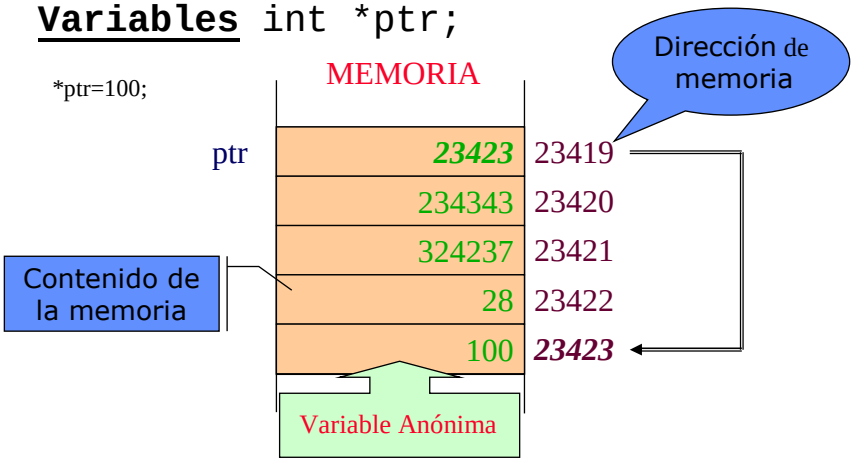
```
typedef struct
{ int num;
   char car;
} tiporegistro;
typedef tiporegistro *tipopuntero;
tipopuntero p;
```

Así:
p es la dirección de un registro con dos campos. (tipo puntero)
***p es un registro con dos campos (tipo registro)**
(*p).num es una variable simple (tipo entero)
p->num es una variable simple (tipo entero)
&x es la direccion de una variable x, siendo x, por ejemplo int x;
Si deseamos que una variable tipo puntero no apunte a nada, asignamos la palabra reservada NULL (p=NULL)

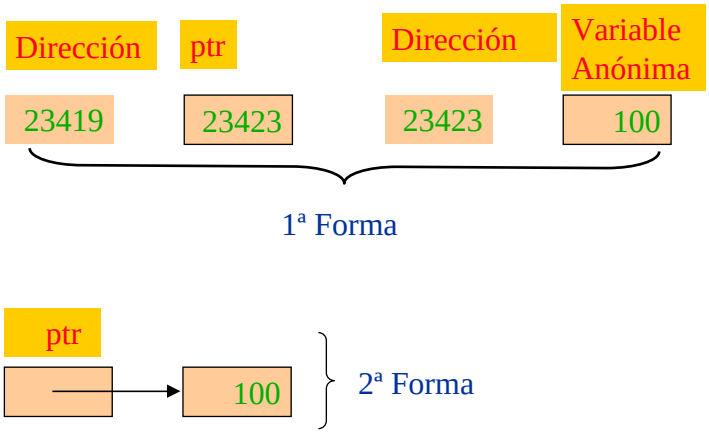
2. Variables Anónimas

Variables `int *ptr;`

*ptr=100;



2. Representación Gráfica de las Variables Puntero



3. Asignación y liberación dinámica de memoria

malloc(): asigna un bloque de memoria.

```
void *malloc(size_t size);
```

- ***Argumento*:** Número de bytes que se quieren reservar (`unsigned int`).
- ***Devuelve*:** Puntero a la zona de memoria reservada (de tipo `void *`).
  - Si no hay suficiente memoria devuelve **NULL**.
  - **SIEMPRE** hay que comprobar si la función devuelve o no **NULL**.
- ***Útil*:** Usar `sizeof()` y usar conversión de tipos para el puntero.
- **Ejemplo:** Declaramos un puntero a entero y le asignamos la dirección de memoria de un bloque de tamaño 1 entero. El valor devuelto por `malloc()` se asigna al puntero tras efectuar una conversión de tipo adecuada:

```
int *a;
a = (int *) malloc (sizeof(int))
```


3. Asignación y liberación dinámica de memoria

realloc(): cambia el tamaño de un bloque de memoria ya asignado.

```
void *realloc(void *memblock, size_t
  size);
```

- El nuevo tamaño puede ser mayor o menor y no se pierde la información que hubiera almacenada (si cambia de ubicación se copia).

calloc(): asigna espacio para un array.

```
void *calloc(size_t num,size_t size);
```

free(): libera un bloque de memoria.

```
void free(void *memblock);
```

Todas las funciones están en **stdlib.h**

3. Asignación y liberación dinámica de memoria

- **EJEMPLO**: Asignación dinámica de memoria a un entero.

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

void main() {
  int a, *b;
  b = (int *) malloc(sizeof(int)); /* Reserva Mm para UN entero */
  if (b == NULL)
     printf("No hay memoria suficiente.");
  else {
     a=5;
     *b=6;
     printf("Dirección de a: %p. Valor de a: %d\n",
       &a, a);
     printf("Dirección b: %p. valor al que apunta: %d\n",
       b, *b);
     free(b); /* Importante: Liberar la memoria */
  }
}
```

SALIDA EN PANTALLA:

```
Dirección de a: FFF4. Valor de a: 5
Dirección b: 05E2. Valor al que apunta: 6
```

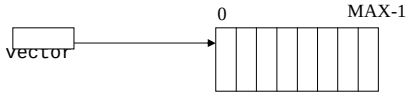

4. Arrays dinámicos.

- A veces no se sabe **cuánta memoria** se necesita para un array.
  - **SOLUCIÓN: Crear un array con memoria dinámica.**
  - Se declara un puntero y se reserva la memoria necesaria cuando el programa se está ejecutando.

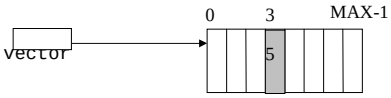
4. Arrays dinámicos.

- Comparación de arrays estáticos y dinámicos:

```
int vector[MAX];
```

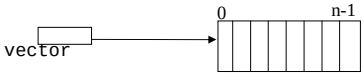



```
vector[3]=5;
```

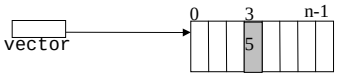



```
int *vector;
```


```
vector=(int *)malloc(n*
sizeof (int));
```




```
vector[3]=5;
```



4. Arrays Dinámicos: Ejemplo

```
#include <stdio.h>
#include <stdlib.h>
void main() {
  float *V, *ptr; /* ptr es para acceder al array V por punteros */
  int N, i;
  printf("Número de valores: "); scanf("%d",&N);
  if ((V=(float *) malloc(N*sizeof(float))) == NULL) {
    printf("No hay suficiente memoria.\n");
  else {
    for(i=0,ptr=V; i<N; i++,ptr++) {
       printf("Valor >> ");
       scanf("%f",ptr /* o &V[i] */);
    }
    for(i=0,ptr=V;i<N;i++,ptr++)
       printf("\nValor %d >> %f", i, *ptr /* o V[i] */);
    free(V);
  }
}
```

4. Arrays Dinámicos: Ejemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h> // para getch()
void main ()
{
  float *V=NULL;
  int N=0,i; char c;
  do{
    V=(float *)realloc((float *)V,(N+1)*sizeof(float));
    printf("Dame un valor>> "); scanf("%f",&V[N]);
    printf("Quieres introducir otro valor? (S/N >> ");
    c=getch();
    N++;
  }while(c=='S' || c=='s');
  for(i=0;i<N;i++) printf("\nValor %d >> %f\n",i,V[i]);
  free(V);
}
```


4. Arrays Dinámicos: Ejemplo

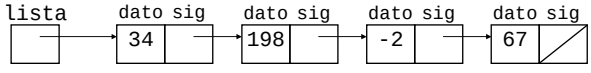
```
#include <stdio.h>
#include <stdlib.h>

typedef char Tpalabra[20];
typedef struct {
        Tpalabra nombre;
        int edad;
} Tpersona;

typedef struct {
        int num_personas;
        Tpersona *personas;       // apuntador al vector dinámico de personas
} Tlista;
```

```
void main ()
{
        Tlista mi_lista;
        int i;
        printf ("Escribe el numero de personas del vector\n");
        scanf ("%d",&mi_lista.num_personas);
        // Ahora se el tamaño inicial del vector. Reservo espacio de memoria
        mi_lista.personas = (Tpersona *) malloc (mi_lista.num_personas*sizeof
(Tpersona));
        if (mi_lista.personas == NULL)
                printf ("Operacion incorrecta");
        else{
                // cargo el vector con los datos leidos del teclado
                for (i=0; i<mi_lista.num_personas; i++)
                {
                        printf ("Escribe el nombre:\n ");
                        scanf ("%s",mi_lista.personas[i].nombre);
                        printf ("Escribe la edad:\n ");
                        scanf ("%d",&mi_lista.personas[i].edad);
                };

                // Añado más elementos


                // cuando termino de trabajar con el vector, libero el
espacio de memoria                      free (mi_lista.personas);
        }
}
```

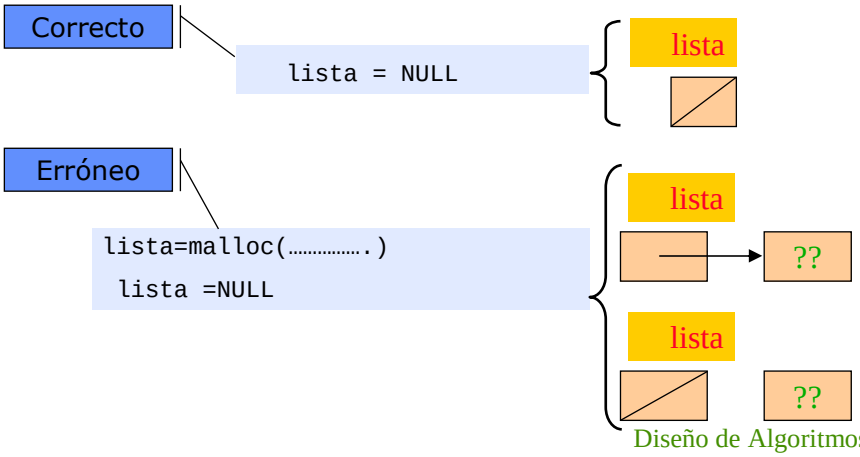

5. Listas enlazadas con punteros.

- Un puntero puede apuntar a variables que a su vez contienen punteros.
- Cuando en cada variable anónima o *nodo* tenemos un solo puntero que apunta al siguiente nodo tenemos una *lista enlazada*.
- *Ejemplo:*

```
#include <stdlib.h>
typedef struct nodo
{ int dato;
  struct nodo *sig;
} tnodo;
tnodo *lista;
```



5. Listas enlazadas con punteros. Inicialización



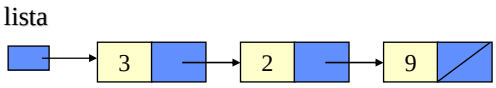
5. Operaciones básicas sobre listas enlazadas.

Suponiendo:

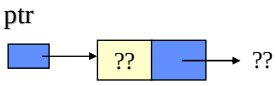
```
typedef struct nodo { int dato;
          struct nodo *sig; }
  tiponodo;
typedef tiponodo *tipolista;
tipolista lista; /*cabeza de la
  lista*/
tipolista nodo; /*nuevo nodo a
  insertar*/
tipolista ptr; /*puntero auxiliar*/
```

5. Insertar un Nodo al Principio

Suponiendo:

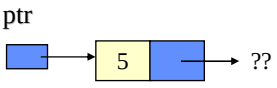



```
1.- ptr=malloc(sizeof(tiponodo));
```

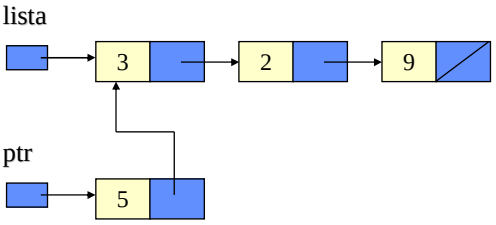


5. Insertar un Nodo al Principio

```
2.- ptr->dato= 5
```

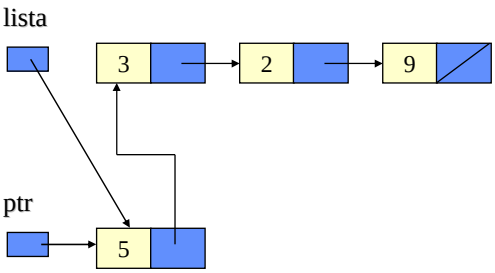



```
3.- ptr->sig= lista
```



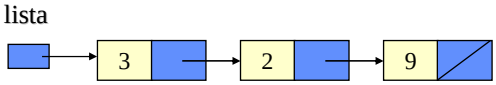
5. Insertar un Nodo al Principio

```
4.- lista= ptr
```

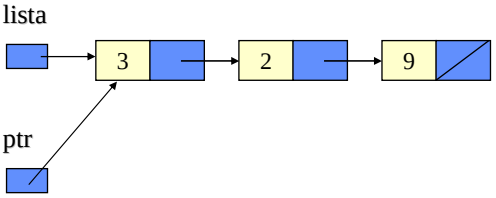


5. Eliminar el Primer Nodo

Suponiendo:

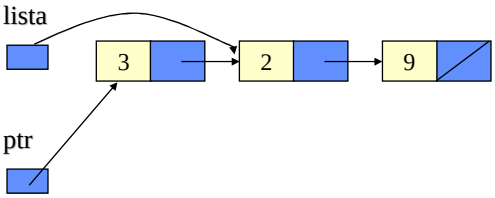



```
1.- ptr = lista
```

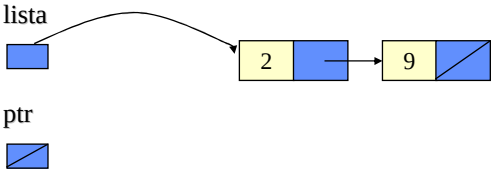


5. Eliminar el Primer Nodo

```
2.- lista = lista->sig;
```



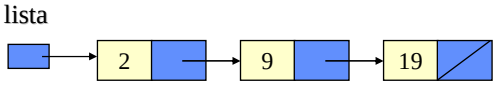

```
3.- free(ptr);
```



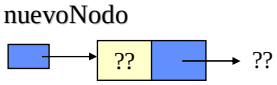
5. Insertar un Nodo en una Lista Enlazada Ordenada

Partimos de la lista:

Queremos insertar el número: 15

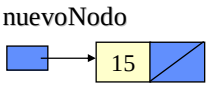



```
1.-
  nuevonodo=malloc(sizeof(tiponodo))
```



5. Insertar un Nodo en una Lista Enlazada Ordenada

```
2.- nuevoNodo->dato = 15
    nuevoNodo->sig  = NULL
```




```
3.- Algoritmo que inserta el nodo
    en la posición correcta.
```

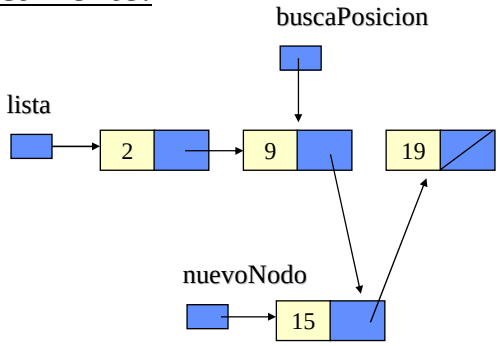

5. Insertar un Nodo en una Lista Enlazada Ordenada

- Si la lista no está vacía utilizamos unm bucle similar al siguiente:

```
while  ((ptr→sig!=NULL) &&
(nuevonodo→dato >
  (ptr→sig)→dato)))
   ptr=ptr→sig;
Nodo→sig=ptr→sig;
Ptr→sig=nodo;
```

5. Insertar un Nodo en una Lista Enlazada Ordenada

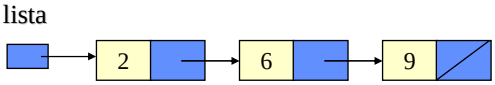
Gráficamente:



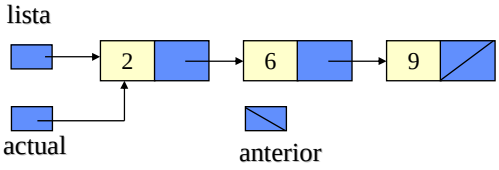
5. Eliminar un Nodo de una Lista Enlazada

Suponiendo:

Queremos borrar el número: 6
Supondremos que está en la lista.



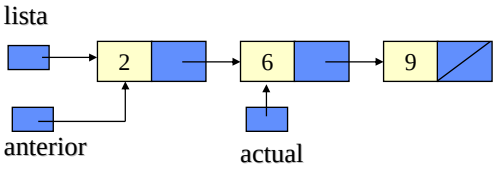

```
1.- actual   = lista
    anterior = NULL
```



5. Eliminar un Nodo de una Lista Enlazada

2.- Búsqueda del nodo a borrar.

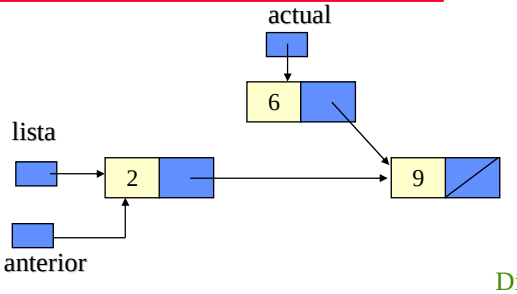
```
while (actual->dato != dato)
{    anterior = actual;
    actual= actual->sig;
}
```



5. Eliminar un Nodo de una Lista Enlazada

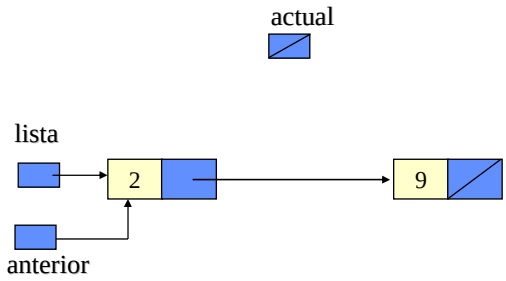
3.- Actualizar los punteros

```
if (anterior ==NULL)
   lista = lista->sig
else
   anterior->sig = actual->sig
```



5. Eliminar un Nodo de una Lista Enlazada

```
4.- free(actual);
```



5. Ejemplo: Lista de números enteros

```
#include <stdio.h>
#include <stdlib.h>
typedef struct nodo{
    int dato;
    struct nodo *enlace;
} TipoNodo;
typedef TipoNodo *LISTA;
void mostrar_lista(LISTA ptr);
void insertar(LISTA&  ptr, int
    elemento);
int main(){
  LISTA  n1 = NULL;
  int elemento;
    // Inserta elementos hasta //leer
    el cero
do {
    printf("\nIntroduzca elemento:");
    scanf("%d", &elemento);
    if(elemento != 0)
      insertar(n1, elemento);
  } while(elemento != 0);
  printf("\nLa nueva lista enlazada
    es: ");
 mostrar_lista(n1);
}
void mostrar_lista(LISTA ptr){
   while(ptr != NULL){
     printf("%d",ptr->dato);
     ptr = ptr->enlace;
   }
printf("\n");
}
```

5. Ejemplo: Lista de números enteros

```
void insertar(LISTA& ptr, int
    elemento){
  LISTA p1, p2;
  p1 = ptr;
  if(p1 == NULL){
    p1 = malloc(sizeof(TipoNodo));
    if (p1 != NULL){
      p1->dato = elemento;
      p1->enlace = NULL;
      ptr = p1;
    }
  }
```

```
else{
    while(p1->enlace != NULL)
      p1 = p1->enlace;
    p2 = malloc(sizeof(TipoNodo));
    if(p2 != NULL){
      p2->dato = elemento;
      p2->enlace = NULL;
      p1->enlace = p2;
    }
  }
}
```