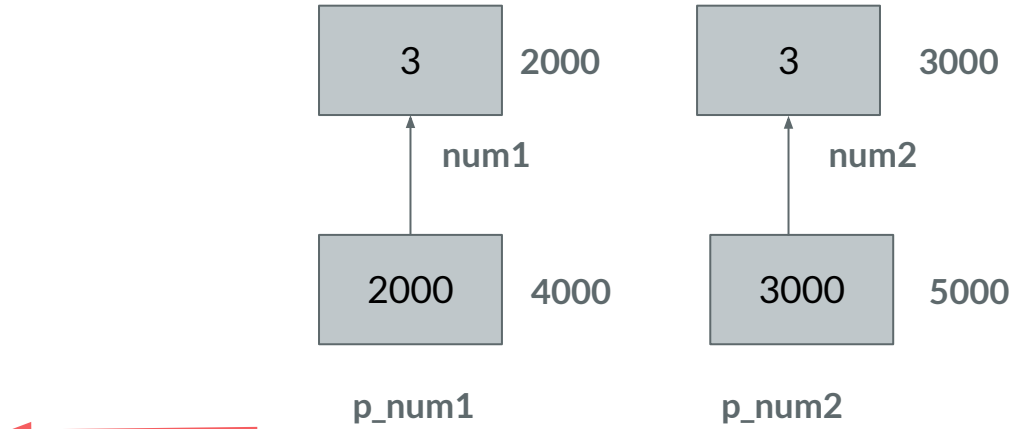
Estructuras de datos dinámicas II

1ºDAM IoT

Comparación de punteros

Comparar (==, !=) dos punteros significa evaluar las direcciones de memoria a las que apuntan.

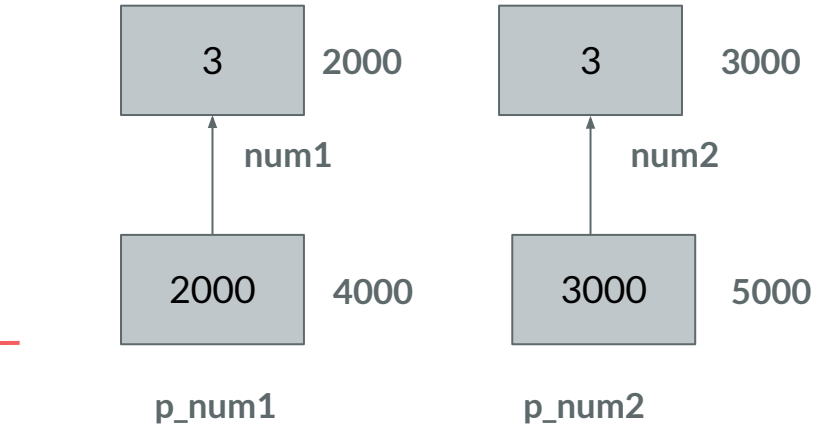
```c
int main(){
    int num1 = 3;
    int num2 = 3;
    int *p_num1, *p_num2;
    p_num1 = &num1;
    p_num2 = &num2;
    if(p_num1 == p_num2){
        printf("Son iguales");
    }else{
        printf("Son diferentes");
    }
    return 0;
}
```



Comparación de punteros

Comparar (==, !=) dos punteros significa evaluar las direcciones de memoria a las que apuntan.

```c
int main(){
    int num1 = 3;
    int num2 = 3;
    int *p_num1, *p_num2;
    p_num1 = &num1;
    p_num2 = &num2;
    if(*p_num1 == *p_num2){
        printf("Son iguales");
    }else{
        printf("Son diferentes");
    }
    return 0;
}
```



Punteros a arrays

Los punteros los podemos utilizar para recorrer un array.

El puntero guardaría la dirección de la posición 0 del array.

```c
float vec[5] = {1.5, 2.2, 3.1, 4.9, 5.0};
float *pe;
int f;
pe = vec;
for (f = 0; f < 5; f++)
{
    printf("%f ", pe[f]); /*1.500000 2.200000 3.100000 4.900000 5.000000*/
}

char cad[5] = "Hola";
char *pc;
int i;
pc = cad;
printf("%s", pc); /*Hola*/
```

pe apunta al inicio del array.

pc apunta al comienzo del array.

Punteros - Declaración

Cuando se declara un apuntador, éste NO apunta a ninguna variable en concreto. Dicho más correctamente, apunta a una dirección de memoria que no tiene por qué estar reservada para él y, peor aún, que puede estar siendo usada por otro apuntador o proceso.

Nunca debemos dar por hecho que el compilador inicializa las variables.

Lo recomendable es declararlos inicializándolos a NULL.

int *p=NULL;

Y para comprobar si está inicializado.

if (p==**NULL**) // si el puntero “p” no apunta a una memoria válida

Punteros - Declaración

Dos opciones para declarar punteros:

1. Declaramos el puntero y al mismo tiempo lo inicializamos a NULL. No reservamos memoria desde el principio.

   int* puntero = NULL;

2. Declaramos el puntero y hacemos que apunte a una variable, array, etc.

   int numero = 1;

   int *puntero;

   puntero = №

3. Declaramos el puntero y reservamos memoria para la variable que va a albergar.

   int* puntero = (...)malloc(...);

Usamos punteros para usar memoria sólo en la medida en la que haga falta. Mientras que apunta a NULL, no consume memoria.

Reserva de memoria dinámica

Para que un apuntador apunte a una variable válida, hay que realizar una “reserva de memoria” para dicha variable.

Este mecanismo de “reserva” es esencial porque es en el que se sustentan las estructuras de datos dinámicas.

Hasta ahora, la reserva de memoria para las variables se había hecho de forma estática (al arrancar el programa y en base a las variables declaradas).

Ahora tendremos la capacidad de crear variables (reservar memoria para ellas) con el programa ya en ejecución, haciendo que los apuntadores “recuerden” la dirección de memoria que (dinámicamente) se ha reservado para ellas.

Reserva de memoria dinámica

La instrucción que nos permite en C reservar memoria es la función ***malloc()***.

Es una abreviatura de Memory ALLOCation (reserva de memoria).

Al ser malloc() una instrucción que se usa en tiempo de ejecución (según el programa la precise o no), eso confiere al programa la capacidad de consumir más o menos memoria, a demanda y según sus necesidades.

malloc()

La función **malloc** admite como argumento la cantidad de bytes que hay que reservar, al tiempo que devuelve la dirección de memoria donde se ha establecido dicha reserva.

Para usar esta función debemos incluir en nuestro programa la librería stdlib.h.

#include <stdlib.h>

int* pentero; //declaración de un apuntador a un entero (int)

pentero = (int*)malloc(4); //Reserva espacio para un entero, 4 bytes

La invocación a malloc() reserva el espacio de un entero en una zona de memoria que tendrá una dirección expresada como un número entero. Ese "número de dirección" es lo que devuelve malloc().

El texto (int*) que precede a malloc() es un casting.

NOTA: devolverá un puntero nulo (NULL) si la reserva de memoria no puede realizarse, generalmente por falta de memoria disponible. Puede ser interesante comprobar si el puntero es igual NULL después de la reserva con malloc.

Reserva de memoria dinámica

¿Qué ocurre si no conocemos el tamaño del tipo de dato? Podemos hacer uso de la función ***sizeof()***.

int pentero; //declaración de un apuntador a un entero*

pentero = (int)malloc(**sizeof(int)**); //reservar memoria para él*

Ejemplo

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

int main()
{
    int entero = 5;
    int *ptr1;
    ptr1 = &entero;                          /*ptr1 apunta a entero, no es necesario reservar espacio*/
    int *ptr2 = NULL;                        /*Vacío el puntero, no apunta a nada*/
    *ptr2 = 5;                               /*Falla porque no hemos reservado espacio para el entero*/
    int *ptr3;                               /*El puntero puede estar apuntando a cualquier dirección*/
    *ptr3 = 5;                               /*No sabemos dónde esta guardando el valor 5*/
    int *ptr4 = (int *)malloc(sizeof(int)); /*Reservo espacio, ahora sí podemos asignar 5*/
    *ptr4 = 5;                               /*asigno un entero*/
    return 0;
}
```

free()

Cuando se reserva memoria dinámica ésta persiste hasta que finalice el programa o hasta que el programador libere dicho bloque de memoria mediante la función free().

```
int *ptr4 = (int *)malloc(sizeof(int)); /*Reservo espacio, ahora sí podemos asignar 5*/
*ptr4 = 5;                              /*asigno un entero*/
free(ptr4);                             // Libero memoria
```

Ejemplo

¿Qué imprime la salida del siguiente código? ¿Por qué?

```c
int main()
{
    int *p, *q;
    p = (int *)malloc(sizeof(int));
    *p = 5;
    q = (int *)malloc(sizeof(int));
    *q = 8;
    free(p);
    p = q;
    free(q);
    q = (int *)malloc(sizeof(int));
    *q = 6;
    printf("p = %d, q = %d\n", *p, *q);
    return 0;
}
```