3. Memoria Dinámica y Punteros

Objetivos:

- **Distinguir los conceptos de memoria estática y memoria dinámica**
- **Comprender el concepto de puntero como herramienta de programación**
- **Conocer cómo se definen y cómo se usan los punteros en código algorítmico y C**
- ***Entender los mecanismos de memoria dinámica y su relación con los punteros***
- ***Conocer las operaciones básicas de gestión de memoria y su equivalente en C***

3. Memoria Dinámica y Punteros

1. **Introducción**
2. **Punteros en lenguaje algorítmico**
3. **Punteros en C**
4. ***Gestión de Memoria Dinámica en Código algorítmico***
5. ***Gestión de Memoria Dinámica en C***

Bibliografía

- **Biondi y Clavel. Tomo 2.**
- **Kernighan y Ritchie, Capítulo 5.**
- **García-Bermejo Giner, Capítulo 2.5**

3.1. Introducción

En la memoria del ordenador se guardan tanto los objetos que se definen (constantes y variables) como las instrucciones del programa.

Cada objeto y cada instrucción en la memoria tiene asignada una dirección.

Sólo se carga el programa principal en la memoria inicialmente. El resto de algoritmos (subprogramas / funciones) se va cargando según se van llamando.

Variables estáticas

- Se crean con el algoritmo (programa) en el momento que se declaran: **Se crean en tiempo de compilación**
- Existen (en la memoria del ordenador) mientras el algoritmo no termine
- Se destruyen cuando el algoritmo termina

La memoria dinámica

Módulo de Gestión de Memoria Dinámica (MGMD)

- Existe un espacio de memoria reservado por programa
- Las instrucciones en código máquina y las variables estáticas ocupan sólo una parte de él
- MGMD: Gestiona espacio disponible de memoria por programa según las necesidades
- Permite crear/destruir objetos temporales
- Se crean/destruyen en tiempo de ejecución
- Estos objetos se denominan **Variables Dinámicas**
- Introducen nuevos problemas:
  - ¿Cómo nombrar un número variable de objetos que pueden incluso no existir?
  - No se puede utilizar el esquema <nombre, tipo, valor>

Ejemplo Código Algorítmico	*Ejemplo en C*
{Variables estáticas}	*/* Variables estáticas */*
{Declaración de variables}	*int A;*
A, B: numérico;	*float B;*
C: carácter;	*char c;*

3.2 Punteros en lenguaje algorítmico

- Los punteros son recursos de programación que existen en algunos lenguajes
- Los punteros permiten hacer referencia a posiciones de memoria de las variables
  - Son variables estáticas (tienen nombre, tipo y un valor)
  - El tipo del puntero es fijo e indica el tipo de variable a la que puede apuntar
  - Su valor es el de la dirección de memoria de una variable (si la variable es dinámica, el puntero representa el nombre de la variable dinámica)
  - Se corresponden con el direccionamiento indirecto de operandos (Bloque 1, Tema 4)

Definición:

<definición-variable-puntero>::=
<id-variable> {, <id-variable>}: puntero a <tipo>;

Ejemplo:

P: puntero a carácter;
Q: puntero a numérico;

- El operador contenido (→) permite acceder al contenido de la variable a la cual está apuntando el puntero
- Inicialmente los punteros no apuntan a una dirección de memoria válida. Se usa el valor NIL para designar un puntero NULO.

Ejemplo:

```
P: puntero a numérico;
I: numérico;
Inicio
        P := nil;
        I:=10;
        P := I;  {P apunta a la dirección de memoria que ocupa I}
        Escribir P;
        Escribir P→;
        I := I * 2;
        Escribir P→;
Fin
```

3.3. Punteros en C

En C los punteros no sólo apuntan a posiciones de memoria dinámica; también pueden apuntar a variables estáticas (a su dirección en memoria)

¿Cómo se organiza la memoria asociada a un programa?

- Como una colección de posiciones de memoria consecutivas. En ellas se almacenan los distintos tipos de datos, que ocupan, por ejemplo:
  - 1 char = 1 byte
  - 1 int = 2 ó 4 bytes
  - 1 float = 4 u 8 bytes

En C un puntero también es una variable estática cuyo contenido es una dirección de memoria.

- Los punteros, por lo tanto, guardan la dirección de la palabra donde comienza un dato (conjunto de celdas de memoria: 1, 2 ó 4 palabras, en función del tipo de dato y del tamaño de palabra)

- En C una variable de tipo puntero se define como:
  <tipo> *<variable>;
- Inicialmente un puntero no apunta a ningún sitio. En C el valor NULL se reserva para indicar que el puntero está vacío (equivalente al NIL del código algorítmico).

Ejemplo:

```
int *pe;        /* ip sólo puede apuntar a variables enteras */
char *pc;       /* pc sólo puede apuntar a variables carácter */
float *pr;      /* pr sólo puede apuntar a variables reales*/
double *prd, /* prd sólo puede apuntar a variables reales de doble
   precisión*/
```

- **Operadores asociados a punteros**
  - **Operador dirección: &**
    me da la dirección de una variable de la memoria (no sirve ni para apuntar a constantes ni expresiones).
  - **Operador indirección: ***
    me da el contenido de una posición de memoria (generalmente almacenada en un puntero. Es el equivalente al operador → del código algorítmico).

Ejemplo:

```
int main(){
int *pe;        /* ip sólo puede apuntar a variables enteras */
char *pc;       /* pc sólo puede apuntar a variables carácter */
int i;
char c;

i = 10;
c = 'a';
pe = NULL;
pc = NULL;
/* Las direcciones de memoria son enteros largos (long int)*/
printf("\n i = %d, c = %c, pe = %ld, pc = %ld", i, c, pe, pc);
pe = &i;
pc = &c;
printf("\n i = %d, c = %c, pe = %ld, pc = %ld", i, c, pe, pc);
printf("\n i = %d, c = %c", i, c);
printf(", contenido de pe = %d, contenido de pc = %c", *pe, *pc);

printf("\n Es lo mismo i = %d que el contenido de pe %d", i, *pe);
printf("\n Es lo mismo c = %c que el contenido de pc %d", i, *pc);
}
```

3.3.2. Los punteros y los argumentos a funciones en C

En C se usan punteros para realizar el paso de parámetros de tipo resultado o dato-resultado.

Código algorítmico	C	Ejemplos
Parámetro Dato	Paso por valor	printf(**"%d"**, **entero**)
Parámetro Resultado	Valor devuelto por la función ó Paso por referencia	**int** main() **float** cambio(float euros) **float** sqrt (float v)
Parámetro Dato-Resultado	Paso por referencia	scanf("%d", **&entero**)

En C**, por defecto**, todos los parámetros a las funciones se pasan por valor (la función recibe una copia del parámetro, que no se ve modificado por los cambios que la copia sufra dentro de la función).

Para que un parámetro de una función pueda ser modificado, ha de pasarse por referencia, y en C eso sólo es posible pasando la dirección de la variable en lugar de la propia variable (&).

Ese parámetro que se pasa por referencia será un puntero.

Si se pasa la dirección de una variable, la función puede modificar el contenido de esa posición mediante el operador *.

Ejemplo: Intercambio de dos valores de dos variables del mismo tipo

```
{VERSION ERRONEA}
intercambia_mal (int a, int b) {
int tmp;
tmp = a;
a = b;
b = tmp;}

{VERSION CORRECTA}
intercambia_bien (int *a, int *b) {
int tmp;
tmp = *a;
*a = *b;
*b = tmp; }
```

3.3.3. Aritmética de punteros en C

El compilador C es capaz de "adivinar" cuál es el tamaño de una variable de tipo puntero y realiza los incrementos / decrementos de los punteros de la forma adecuada.

```
int *p;
int i;
p = &i;
/* p:       Apunta a la variable i*/
/* p + 0:   Apunta a la variable i */
/* p + 1:   Apunta a la siguiente posición de memoria,
   avanzando el tamaño de un entero */
/* p – 1:   Apunta a la anterior posición de memoria,
   avanzando el tamaño de un entero */
```

Volveremos a ver estos conceptos cuando hayamos definido los tipos vectoriales y veamos la equivalencia entre el nombre de una variable vector y la dirección de memoria que ocupa

Los operadores * y & son unarios y tienen más prioridad a la hora de evaluarlos que los operadores binarios.

```
*pe = *pe + 10;
printf (“\n i=%d, *pe=%d, pe=%ld”, i, *pe, pe);
i = *pe + 10;
printf (“\n i=%d, *pe=%d”, i, *pe);
*pe += 1;
printf (“\n i=%d, *pe=%d”, i, *pe);
```

Es necesario utilizar paréntesis cuando aparezcan en la misma expresión que otros operadores unarios como ++ o --, ya que en ese caso se evaluarían de izquierda a derecha

```
(*pe)++;
printf (“\n i=%d, *pe=%d”, i, *pe);
*pe++;
printf (“\n i=%d, *pe=%d”, i, *pe);
```

Dado que los punteros son variables, también pueden usarse como asignaciones entre direcciones. Así:

```
int *pe, *qe;
pe = qe; /* Indica que pe apunta a donde apunte el puntero qe*/
```