Tema 5: Punteros

Oscar Perpiñán Lamigueiro

1 Definición

2 Uso de punteros

3 Punteros a vectores

4 Punteros a cadenas de caracteres

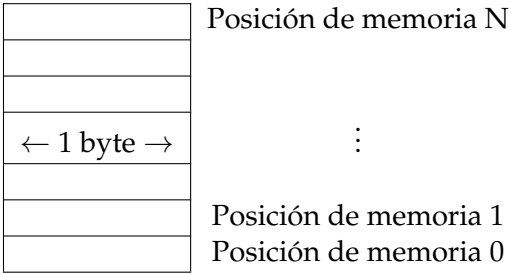
5 Punteros a estructuras

6 Funciones y punteros

7 Asignación dinámica de memoria

Datos y Memoria

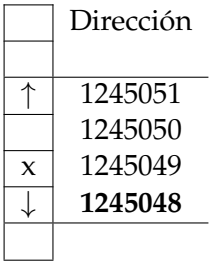
- Los datos de un programa se almacenan en la memoria del ordenador.
- La memoria del ordenador está estructurada en **bytes** (8 bits).
- Cada byte tiene una posición en la memoria (dirección).



Dirección de memoria de una variable

- Ejemplo: un dato `int` ocupa 4 bytes.

```
int x;
```



Operador &

El operador & (*ampersand*) aplicado a una variable cualquiera proporciona su dirección de memoria.

```
#include <stdio.h>

int main()
{
  // No hace falta asignar valor inicial
  // para que la variable
  // tenga dirección de memoria
  int x;

  printf("La variable x está almacenada en %lli.\n",
         &x);

  return 0;
}
```

¿Qué es un puntero?

Un puntero apunta a una variable

Un **puntero** (*pointer*) es una **variable** (tipo número entero) que contiene la dirección de memoria de una variable:

- El puntero es una referencia de la variable a la que apunta.
- El valor del puntero es la dirección de memoria de la variable.
- La variable está apuntada por el puntero.

1 Definición

2 Uso de punteros

3 Punteros a vectores

4 Punteros a cadenas de caracteres

5 Punteros a estructuras

6 Funciones y punteros

7 Asignación dinámica de memoria

Declaración de un puntero

Un puntero se declara:

- Indicando el tipo de datos de la variable a la que apunta.
- Incluyendo un asterisco * antes del identificador.

```
int main()
{
  // p1: puntero apuntando a una variable de tipo entero
  int *p1;
  // p2: puntero apuntando a una variable de tipo caracter
  char *p2;
  // p3: puntero apuntando a una variable de tipo real
  float *p3;
  // p4: puntero apuntando a una variable genérica
  void *p4;

  return 0;
}
```

Un puntero es una variable `int`

- El contenido de una variable puntero es la dirección de memoria, un valor de tipo entero.
- Su tamaño depende del sistema:
  - Sistemas de 32 bits ocupan 4 bytes.
  - Sistemas de 64 bits ocupan 8 bytes.

```
#include <stdio.h>

int main()
{
  int x, *p;
  // sizeof devuelve el numero de bytes de una variable o tipo de datos
  printf("La variable x ocupa %i bytes.\n",
         sizeof(x));
  printf("El puntero p ocupa %i bytes.\n",
         sizeof(p));

  return 0;
}
```

Asignación

```
int main()
{
  int x, y;
  // p1 apunta a x
  int *p1 = &x, *p2, *p3;
  // p2 apunta a y
  p2 = &y;
  // p3 apunta a la misma variable que p1, es decir, x
  p3 = p1;

  return 0;
}
```

Operador *

El operador * aplicado a un puntero proporciona el valor de la variable apuntada por el puntero.

```
#include <stdio.h>

int main()
{
  int x = 10, *p;
  // Operador & para obtener la direccion de x
  p = &x;
  // *p y x son lo mismo
  printf("La variable apuntada vale %i",
         *p);
}
```

Operaciones con punteros

- La **suma o resta de un entero a un puntero** produce una **nueva localización de memoria**.
- Se pueden **comparar punteros** utilizando expresiones lógicas para **comprobar si apuntan a la misma dirección de memoria**.
- La **resta de dos punteros** da como resultado el **número de variables entre las dos direcciones**.

1 Definición

2 Uso de punteros

3 Punteros a vectores

4 Punteros a cadenas de caracteres

5 Punteros a estructuras

6 Funciones y punteros

7 Asignación dinámica de memoria

Punteros y vectores

El identificador de un vector es un puntero *constante* que apunta al primer elemento del vector.

```
#include <stdio.h>

int main()
{
  int vector[3] = {1, 2, 3};
  int *p;
  // p apunta al primer elemento del vector
  p = &vector[0];
  printf("El primer elemento es %i\n", *p);
  // De forma mas concisa
  p = vector;
  printf("El primer elemento es %i\n", *p);

  return 0;
}
```

Recorrido de un vector

Podemos recorrer un vector a través de su puntero con sumas y restas

```
#include <stdio.h>
int main()
{
  int i, vector[3] = {1, 2, 3};
  int *p, *p1;
  p = vector;
  // p apunta a vector[0].
  printf("%i\t", *p);
  // p + 1 apunta a vector[1]
  p1 = p + 1;
  printf("%i\t", *p1);
  // *(p + 1) es equivalente a v[i + 1]
  printf("%i\t", *(p + 1));
  printf("%i\t", vector[1]);
  return 0;
}
```

Modificación de un vector

Podemos modificar un vector a través de su puntero

```c
#include <stdio.h>
int main(){
  int i, vector[3];
  int *p;
  p = vector;
  //vector[0] = 1
  *p = 1;
  //vector[1] = 2
  *(p + 1) = 2;
  //vector[2] = 3
  *(p + 2) = 3;

  printf("El vector es %i, %i, %i.\n",
        vector[0], vector[1], vector[2]);

  return 0;
}
```

Aritmética de punteros con vectores

```c
#include <stdio.h>
#define N 10

int main(){
  int vector[N] = {1};
  int *pVec, *pFin;
  // Puntero apuntando al segundo elemento
  pVec = vector + 1;
  // Puntero apuntando al ultimo elemento
  pFin = vector + N - 1;
  // Comparamos los punteros para avanzar
  while (pVec <= pFin)
    { // vector[i] = vector[i - 1] + 1
      *pVec = *(pVec - 1) + 1;
      printf("%i\t", *pVec);
      ++pVec; // Movemos el puntero por el vector
    }
  return 0;
}
```

1 Definición

2 Uso de punteros

3 Punteros a vectores

4 Punteros a cadenas de caracteres

5 Punteros a estructuras

6 Funciones y punteros

7 Asignación dinámica de memoria

Punteros y cadenas

El identificador de una cadena es un puntero *constante* que apunta al primer elemento.

```
#include <stdio.h>

int main()
{
  char letras[3] = {'a', 'b', 'c'};
  char *p;
  // p apunta al primer elemento
  p = &letras[0];
  printf("El primer elemento es %c\n", *p);
  // De forma mas concisa
  p = letras;
  printf("El primer elemento es %c\n", *p);

  return 0;
}
```

Recorrido de una cadena

```
#include <stdio.h>
int main()
{
  char mensaje[] = "Hola Mundo";
  char *p = mensaje;
  int i = 0;
  // Movemos el puntero por la cadena
  while(*p != '\0')
    {
      printf("%c", *p);
      p++; // Incrementa el puntero para
    } // pasar al siguiente caracter
  printf("\n");
}
```

Aritmética de punteros con cadenas

```
#include <stdio.h>

int main(){
  char texto[] = "Hola Mundo";
  char *pChar = texto;

  while (*pChar != '\0')
    ++pChar; // Movemos el puntero por la cadena

  // El identificador "texto" es un puntero que
  // apunta al primer caracter de la cadena.
  // Si lo restamos del puntero movil
  // tenemos el total.
  printf("La cadena tiene %i caracteres.\n",
        pChar - texto);

  return 0;
}
```

1 Definición

2 Uso de punteros

3 Punteros a vectores

4 Punteros a cadenas de caracteres

5 Punteros a estructuras

6 Funciones y punteros

7 Asignación dinámica de memoria

Punteros a estructuras

- Un puntero a una estructura se declara igual que un puntero a un tipo simple.
- Para acceder a un miembro de la estructura se emplea el operador ->.

```
#include <stdio.h>
typedef struct
{
  int y, m, d;
} fecha;

int main(){
  fecha f = {2000, 10, 15}, *p;
  // p apunta a la estructura
  p = &f;

  printf("%i-%i-%i",
        p->d, p->m, p->y);

  return 0;
}
```

Ejemplo

```c
#include <stdio.h>
typedef struct
{
  int y, m, d;
} fecha;

int main()
{
  fecha f, *p = &f;
  // Rellenamos la estructura a través de su puntero
  p->d = 15;
  p->m = 10;
  p->y = 2000;

  printf("%i-%i-%i",
        f.d, f.m, f.y);

  return 0;
}
```

Ejemplo con scanf

```c
#include <stdio.h>
typedef struct
{
  int y, m, d;
} fecha;

int main()
{
  fecha f, *p = &f;
  // Rellenamos la estructura a través de su puntero con
  // scanf. Atención al uso de &.
  scanf("%i", &(p->d));
  scanf("%i", &(p->m));
  scanf("%i", &(p->y));

  printf("%i-%i-%i",
        f.d, f.m, f.y);

  return 0;
```

1 Definición

2 Uso de punteros

3 Punteros a vectores

4 Punteros a cadenas de caracteres

5 Punteros a estructuras

6 Funciones y punteros

7 Asignación dinámica de memoria

Paso por referencia

- El uso de punteros en funciones permite el **paso por referencia**. De esta forma la función puede acceder (y **modificar**) a la variable original (*sin copia*).
- Las funciones que emplean **vectores y cadenas** como argumentos funcionan con **paso por referencia** (*el identificador de un vector es un puntero*).

Ejemplo

```
#include <stdio.h>
void operaciones (float x, float y,
                  float *s, float *p, float *d);
int main(){
  float a = 1.0, b = 2.0; // Datos
  float suma, producto, division; // Resultados
  operaciones(a, b, &suma, &producto, &division);
  printf("S: %f \t P: %f \t D: %f \t",
         suma, producto, division);
  return 0;
}
//Funcion con varios resultados
void operaciones (float x, float y,
                  float *s, float *p, float *d)
{// Cada puntero sirve para un resultado
  *s = x + y;
  *p = x * y;
  *d = x / y;
}
```

1 Definición

2 Uso de punteros

3 Punteros a vectores

4 Punteros a cadenas de caracteres

5 Punteros a estructuras

6 Funciones y punteros

7 Asignación dinámica de memoria

`malloc` y `free`

- La **asignación dinámica** de memoria permite definir **objetos (p.ej. vectores) de dimensión variable**.
- La función `malloc` permite asignar, durante la ejecución del programa, un bloque de memoria de n bytes consecutivos para almacenar los datos (devuelve `NULL` si no es posible la asignación)
- La función `free` permite liberar un bloque de memoria previamente asignado.

Uso de `malloc` y `free`

```
int *pInt;
...
// Reservamos la memoria suficiente para almacenar
// un int y asignamos su dirección a pInt
pInt = malloc(sizeof(int));

// Comprobamos si la asignación
// se ha realizado correctamente
if (pInt == NULL) {
     printf("Error: memoria no disponible.\n");
     exit(-1);
}

... // Codigo usando el puntero

free(pInt); // Liberamos memoria al terminar
```

Ejemplo

```c
#include <stdio.h>
#include <stdlib.h> //Necesaria para malloc y free
int main () {
  int *vec, i, N = 100;
  vec = malloc(sizeof(int) * N);
  //Comprueba si malloc ha funcionado
  if (vec == NULL) {
     printf("Error: memoria no disponible.\n");
     exit(-1);
    }
  // El resultado es un puntero-vector de N elementos
  for (i = 0; i < N; ++i)
    vec[i] = i * i; // Rellenamos el puntero-vector
  for (i = 0; i < N; ++i) // Mostramos contenido
    printf("%i \t", *(vec + i));
  free(vec); // Liberamos el puntero-vector
  return 0;
}
```