Los punteros en C

25 de Marzo de 2021

Esquema de la clase

Iremos aproximándonos al concepto y la necesidad de usar punteros. Pero primero hagamos un pequeño repaso de la clase anterior

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

#define N 4096
#define P 0.7
#define SEED 260572

int main()
{
    int i,red[N]; // red es un vector.
    double p;
    i = 0;
    srand(SEED);

    while (i<N)
    {
        p = (double)rand()/(double)RAND_MAX;
        if (p<P) red[i] = 1;
        else red[i] = 0;
        i++;
    }

    return 1;
}
```

```c
void llenado_red(int *red);

int main()
{
    int red[N]; // red es un vector.

    llenado_red(red);
    return 1;
}

void llenado_red(int *red)
{
    int i;
    double p;
    i = 0;
    srand(SEED);

    while (i<N)
    {
        p = (double)rand()/(double)RAND_MAX;
        if (p<P) red[i] = 1;
        else red[i] = 0;
        i++;
    }
}
```

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

int funcion(int a);

int main()
{
    int a,resultado;
    a = 1;

    printf("El valor de a antes de la funcion es %d\n\n",a);

    resultado = funcion(a);

    printf("El valor de la variable resultado es %d\n\n",resultado);
    printf("El valor de a luego de la funcion es %d\n\n",a);

    return 1;
}

int funcion(int a)
{
    a = 2*a;
    printf("El valor de a dentro de la funcion es %d\n\n",a);
    return a;
}
```

```
El valor de a antes de la funcion es 1

El valor de a dentro de la funcion es 2

El valor de la variable resultado es 2

El valor de a luego de la funcion es 1
```

¿Y si en vez de hacer una copia de “a” le pudiera pasar su dirección de memoria?

Parámetro por valor $\rightarrow$ por referencia

Es decir, no le indico qué valor usar sino donde tiene que buscar la variable. En otras palabras, hacia donde tiene que “apuntar”.

Veamos como se organizan los datos en la memoria

```c
int a;
float b;
double c;

a = 10;
b = 17.9;
c = 32.4;
```

Declaración de las variables

Definición (inicialización) de las variables

En C usamos & para acceder a la dirección de la memoria, es decir:

44 = &a
48 = &b
52 = &c

Entonces… podríamos definir una nueva variable que apunte hacia otra variable:

```c
int *dir_a = &a
float *dir_b = &b
double *dir_c = &c
```

Direcciones de memoria	Nombre de la variable	Valor de la variable
44	a	10
45		
46		
47		
48	b	17.9
49		
50		
51		
52	c	32.4
53		
54		
55		
56		
57		
58		
59		

```c
int a;
float b;
double c;

a = 10;
b = 17.9;
c = 32.4;
```

Declaración de las variables

Definición (inicialización) de las variables

En C usamos & para acceder a la dirección de la memoria, es decir:

44 = &a
48 = &b
52 = &c

Entonces… podríamos definir una nueva variable que apunte hacia otra variable:

```c
int *dir_a = &a
float *dir_b = &b
double *dir_c = &c
```

Direcciones de memoria	Nombre de la variable	Valor de la variable
44 | 47	a	10
48 | 51	b	17.9
52 | 59	c	32.4
60 | 63	dir_a	44
64 | 67	dir_b	48
68 | 75	dir_c	52

Punteros

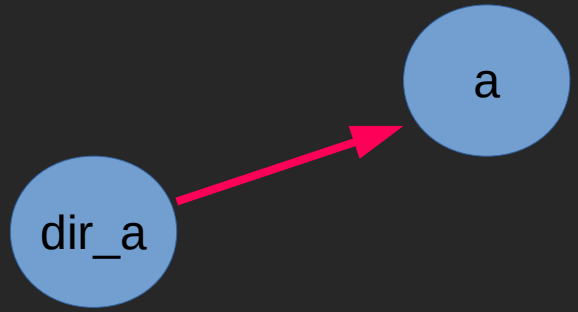
Es una variable (como cualquier otra) con una característica especial: contiene la dirección de otra variable.

El puntero debe ser del mismo tipo que la variable enlazada.

Con “&” obtengo la dirección de la memoria de la variable “a”.

Para conocer el valor que se almacena en esa posición de memoria usamos el *.

Veamos un ejemplo



Punteros

```c
int main()
{
    int a = 46;          //defino y declaro mi variable
    int *dir_a = &a;     //defino y declaro mi puntero

    printf("El valor de a directamente usando la variable es: %i\n\n", a);
    printf("La direccion de a usando la variable es: %p\n\n", &a);

    printf("El valor de a mediante el uso del puntero es: %i\n\n", *dir_a);
    printf("La direccion de a mediante el uso del puntero es: %p\n\n", dir_a);

    printf("La direccion del puntero es: %p\n\n", &dir_a);

   return 0;
}
```

```
El valor de a directamente usando la variable es: 46

La direccion de a usando la variable es: 0x7ffed962b0d0

El valor de a mediante el uso del puntero es: 46

La direccion de a mediante el uso del puntero es: 0x7ffed962b0d0

La direccion del puntero es: 0x7ffed962b0e0
```

Muy bien pero... ¿para que me sirven los punteros?

Para poder manipular datos a lo largo de todo el código (incluso en subrutinas) y no trabajar con copias.

```c
#include <math.h>

void funcion(int a);

int main()
{
    int a;
    a = 1;

    printf("El valor de a antes de la funcion es %d\n",a);
    printf("La direccion de a antes de la funcion es %p\n\n",&a);

    funcion(a);

    printf("El valor de a luego de la funcion es %d\n",a);
    printf("La direccion de a luego de la funcion es %p\n\n",&a);

    return 1;
}

void funcion(int a)
{
    a = 2*a;
    printf("El valor de a dentro de la funcion es %d\n",a);
    printf("La direccion de a dentro de la funcion es %p\n\n",&a);
    return;
}
```

```
El valor de a antes de la funcion es 1
La direccion de a antes de la funcion es 0x7ffd0dd56210

El valor de a dentro de la funcion es 2
La direccion de a dentro de la funcion es 0x7ffd0dd561fc

El valor de a luego de la funcion es 1
La direccion de a luego de la funcion es 0x7ffd0dd56210
```

Hagamos el mismo código de antes pero ahora usemos punteros

```c
void funcion(int *dir_a);

int main()
{
    int a;          //declaro la variable a
    int *dir_a;     //declaro el puntero

    a = 1;
    dir_a = &a;     //direccion de la variable a

    printf("El valor de a antes de la funcion es %d\n\n",a);

    funcion(dir_a);      //le estoy pasando la direccion de a y no su valor

    printf("El valor de a luego de la funcion es %d\n\n",a);

    return 1;
}

void funcion(int *dir_a) //aca *dir_a es la direccion de memoria de a
{
    //aca *dir_a es el valor que almacena esa direccion de memoria
    *dir_a = 2*(*dir_a);
    //le pido que me acualice el valor en esa direccion de memoria
    printf("El valor de a dentro de la funcion es %d\n\n",*dir_a);
}
```

```
El valor de a antes de la funcion es 1

El valor de a dentro de la funcion es 2

El valor de a luego de la funcion es 2
```

Pregunta: ¿para todo tenemos que usar punteros?

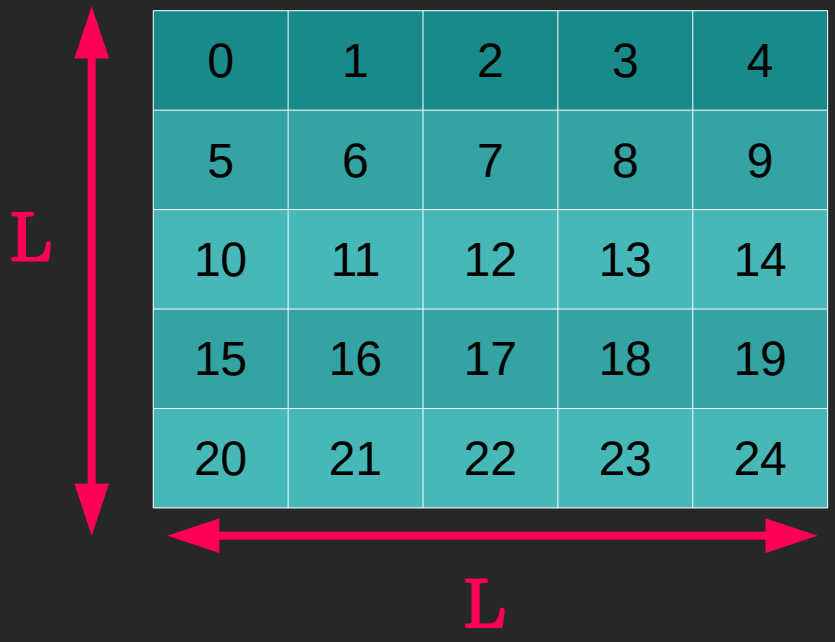
No, sólo en aquellos casos en donde queremos modificar constantemente alguna variable a lo largo del código.

Para terminar, veamos como podemos armar una red mediante punteros

```c
void llenado_red(int *red);

int main()
{
    int red[N]; // red es un vector.

    llenado_red(red);
    return 1;
}

void llenado_red(int *red)
{
    int i;
    double p;
    i = 0;
    srand(SEED);

    while (i<N)
    {
        p = (double)rand()/(double)RAND_MAX;
        if (p<P) red[i] = 1;
        else red[i] = 0;
        i++;
    }
}
```

Un vector es un puntero!

Una matriz (o red) de tamaño N*M se puede pensar como un vector de N*M componentes.



La componente Mij de la matriz corresponde a la componente i*L+j del vector con i,j desde 0 hasta L-1

```c
#define L 10
#define P 0.7
#define SEED 260572

void llenado_red(int *red);
void imprimo_red(int *red);

int main()
{
    int n = L*L;
    int *red; // red es un puntero.
    red = (int *)malloc(n*sizeof(int));

    llenado_red(red);
    imprimo_red(red);
    free(red);
    return 1;
}
```

```c
void llenado_red(int *red)
{
    int i,n;
    double p;
    i = 0;
    srand(SEED);
    n = L*L;

    while (i<n)
    {
        p = (double)rand()/(double)RAND_MAX;
        if (p<P) *(red+i) = 1;
        else *(red+i) = 0;
        i++;
    }
}
```

```c
void imprimo_red(int *red)
{
    int i,j,indice;
    i = 0;
    printf("\n\n");
    while (i<L)
    {
        j = 0;
        while (j<L)
        {
            indice = i*L+j;
            printf("%i  ",*(red+indice));
            j++;
        }
        i++;
        printf("\n");
    }
    printf("\n\n");
}
```

```
1  1  1  1  0  1  1  1  0  1
0  1  0  1  1  1  1  0  1  1
1  1  1  0  0  0  0  1  0  1
1  0  1  1  1  0  1  1  0  1
1  0  1  1  1  1  1  0  1  1
1  1  0  1  0  0  1  0  1  0
1  1  0  1  1  1  1  0  0  1
1  1  1  1  1  1  1  0  1  0
1  1  1  1  1  0  1  0  0  1
0  1  1  0  0  1  1  1  1  1
```