Apuntadores

Apuntador

Cuando se declara una variable, el compilador reserva un espacio de memoria para ella y asocia el nombre de ésta a la dirección de memoria desde donde comienzan los datos de esa variable. Las direcciones de memoria se suelen describir como números en hexadecimal.

El apuntador es una variable cuyo valor es la dirección de memoria de otra variable. Se dice que un apuntador “apunta” a la variable cuyo valor se almacena a partir de la dirección de memoria que contiene el apuntador.

Declaración de apuntadores

01

Para declarar un apuntador se especifica el tipo de dato al que apunta, el operador '*', y el nombre del apuntador. La sintaxis es la siguiente:

<tipo de dato apuntado> *<indentificador del apuntador>

A continuación se muestran varios ejemplos:

```
int *ptr1; // Apuntador a un dato de tipo entero (int)
char *cad1, *cad2; // Dos apuntadores a datos de tipo carácter (char)
float *ptr2; // Apuntador a un dato de tipo punto-flotante (float)
```

Asignación de apuntadores

02

Se pueden asignar a un apuntador direcciones de variables a través del operador de referenciación (‘&’) o direcciones almacenadas en otros apuntadores.

```
int i = 5;
int *p, *q;
p = &i; // Se le asigna a ’p’ la dirección de ’i’
q = p; // Se le asigna a ’q’ la dirección almacenada en ’p’ (la misma de ’i’)
```

Desreferenciación de apuntadores

03

La desreferenciación es la obtención del valor almacenado en el espacio de memoria donde apunta un apuntador. En C esto se hace a través del operador '*', aplicado al apuntador que contiene la dirección del valor.

Ejemplo:

```
int x = 17, y;
int *p;
p = &x;
printf("El valor de x es: %d", *p); // Imprime 17
y = *p + 3; // A 'y' se le asigna 20
```

Desreferenciación de apuntadores

03

C además provee el operador binario ‘->’, utilizado para obtener campos de un registro con un apuntador al mismo de una manera más fácil y legible. Ejemplo:

```
struct Data
    {
    char nombre[20];
    int edad;
    };

struct Data d;
struct Data *pd = &d;

(*pd).edad = 23;
pd->edad = 23;

printf("Edad: %d", pd->edad);
```

Verificación de tipos en apuntadores

04

Al igual que el resto de las variables, los apuntadores se enlazan a tipos de datos específicos (apuntadores a variables de cierto tipo), de manera que a un apuntador sólo se le pueden asignar direcciones de variables del tipo especificado en la declaración del apuntador. Ejemplo:

```
int *p1;
float *p2;
int x;
p1 = &x; // Esto es válido
p2 = &x; // Esto no es válido (el compilador genera un error)
```

Direcciones inválidas

05

Normalmente, un apuntador inicializado adecuadamente apunta a alguna posición específica de la memoria. Sin embargo, algunas veces es posible que un apuntador no contenga una dirección válida, en cuyo caso es incorrecto desreferenciarlo (obtener el valor al que apunta) porque el programa tendrá un comportamiento impredecible y probablemente erróneo, aunque es posible que funcione bien. Un apuntador puede contener una dirección inválida debido a dos razones:

1. Cuando un apuntador se declara, al igual que cualquier otra variable, el mismo posee un valor cualquiera que no se puede conocer con antelación, hasta que se inicialice con algún valor (dirección). Ejemplo:

```
float *p;
printf("El valor apuntado por p es: %f", *p); // Incorrecto
*p = 3.5; // Incorrecto
```

Direcciones inválidas

2. Después de que un apuntador ha sido inicializado, la dirección que posee puede dejar de ser válida si se libera la memoria reservada en esa dirección, ya sea porque la variable asociada termina su ámbito o porque ese espacio de memoria fue reservado dinámicamente y luego se liberó . Ejemplo:

```
int *p, y;

void func(){
int x = 40;
p = &x;
y = *p; // Correcto
*p = 23; // Correcto
}

int main(){
func();
y = *p; // Incorrecto
*p = 25; // Incorrecto
}
```

Apuntadores a apuntadores

06

Dado que un apuntador es una variable que apunta a otra, fácilmente se puede deducir que pueden existir apuntadores a apuntadores, y a su vez los segundos pueden apuntar a apuntadores, y así sucesivamente. Estos apuntadores se declaran colocando tantos asteriscos (‘*’) como sea necesario. Ejemplo:

```
char c = 'z';
char *pc = &c;
char **ppc = &pc;
char ***pppc = &ppc;
***pppc = 'm'; // Cambia el valor de c a ’m’
```

Apuntadores constantes y apuntadores a constantes

07

Es posible declarar apuntadores constantes. De esta manera, no se permite la modificación de la dirección almacenada en el apuntador, pero sí se permite la modificación del valor al que apunta. Ejemplo:

Ejemplo:

```
int x = 5, y = 7;
int *const p = &x; // Declaración e inicialización del apuntador constante
*p = 3; // Esto es válido
p = &y; // Esto no es válido (el compilador genera un error)
```

Apuntadores constantes y apuntadores a constantes

07

También es posible declarar apuntadores a datos constantes. Esto hace que no sea posible modificar el valor al que apunta el apuntador.

Ejemplo:

```
int x = 5, y = 7;
const int *p = &x; // Declaración e inicialización del apuntador a constante
p = &y; // Esto es válido
*p = 3; // Esto no es válido (el compilador genera un error)
y = 3; // Esto es válido
```

Apuntadores, arreglos y aritmética de apuntadores

Los arreglos y apuntadores están fuertemente relacionados. El nombre de un arreglo es simplemente un apuntador constante al inicio del arreglo. Se pueden direccionar arreglos como si fueran apuntadores y apuntadores como si fueran arreglos. Ejemplos:

```
int lista_arr[5] = {10, 20, 30, 40, 50};
int *lista_ptr;
lista_ptr = lista_arr; // A partir de aquí ambas variables apuntan al mismo sitio
printf("%d\n",lista_arr[0]); // Imprime 10
printf("%d\n",lista_ptr[0]); // Instrucción equivalente a la anterior
printf("%d\n",*lista_arr); // Instrucción equivalente a la anterior
printf("%d\n",*lista_ptr); // Instrucción equivalente a la anterior
printf("%d\n",lista_arr[3]); // Imprime 40
printf("%d\n",lista_ptr[3]); // Instrucción equivalente a la anterior
```

Apuntadores, arreglos y aritmética de apuntadores

Es posible sumar y restar valores enteros a un apuntador. El resultado de estas operaciones es el desplazamiento de la dirección de memoria hacia adelante (suma) o hacia atrás (resta) por bloques de bytes del tamaño del tipo de dato apuntado por el apuntador. Esto permite recorrer arreglos utilizando apuntadores. Ejemplos:

```
int lista[5] = {10, 20, 30, 40, 50};
int *p;

p = &lista[3]; // 'p' almacena la dirección de la posición 3 del arreglo
p = lista + 3; // Instrucción equivalente a la anterior
printf("%d\n",lista[2]); // Imprime 30
printf("%d\n",*(lista+2)); // Instrucción equivalente a la anterior
```