Efrain Robles Pulido Codigo: 221350095

Punteros

Un puntero no es más que una variable, en la cual se almacena una dirección de memoria. Al ser una dirección de memoria, le podemos decir a un puntero que en ese lugar donde apunta queremos almacenar un valor.

Cuando una variable de tipo puntero tiene almacenada una dirección de memoria, se dice que «apunta» al valor que está en esa dirección.

Un valor especial llamado NULL puede ser asignado a cualquier puntero para indicar aún no está apuntando a ninguna parte de la memoria.

El operador unario * de los punteros es el operador de **derreferenciación**. Lo que hace es entregar el valor que está en la dirección de memoria.

En otras palabras, * significa «lo apuntado por».

Derreferenciar el puntero NULL no está permitido. Al hacerlo, lo más probable es que el programa se termine abruptamente y sin razón aparente. Errores de este tipo son muy fastidiosos, pues son difíciles de detectar, e incluso pueden ocurrir en un programa que ha estado funcionando correctamente durante mucho tiempo.

Si existe alguna remota posibilidad de que un puntero pueda tener el valor NULL, lo sensato es revisar su valor antes de derreferenciarlo.

Puntero por paso por referencia

Los parámetros formales no son variables locales a la función, sino **alias** de los propios parámetros actuales. **¡No se crea ninguna nueva variable!** Por tanto, cualquier modificación de los parámetros formales afectará a los actuales.

La idea es que como solo se puede pasar el valor de una variable a una función lo que hacemos es pasar la dirección de una variable a través de un parámetro de puntero y luego con el operador de indirección podemos acceder al contenido de la variable original.

Ejemplo:

```
#include <iostream>

using namespace std;

void funcion(int *q);

int main() {

  int a;

  int *p;

  a = 100;

  p = &a;

  // Llamamos a funcion con un puntero
```

```
  funcion(p); // (1)
  cout << "Variable a: " << a << endl;
  cout << "Variable *p: " << *p << endl;
  // Llamada a funcion con la dirección de "a" (constante)
  funcion(&a);  // (2)
  cout << "Variable a: " << a << endl;
  cout << "Variable *p: " << *p << endl;
  return 0;
}
void funcion(int *q) {
  // Cambiamos el valor de la variable apuntada por
  // el puntero
  *q += 50;
  q++;
}
```

Puntero por paso por valor

Los parámetros formales son variables locales a la función, es decir, solo accesibles en el ámbito de ésta. Reciben como valores iniciales los valores de los parámetros actuales. Posteriores modificaciones en la función de los parámetros formales, al ser locales, no afectan al valor de los parámetros actuales.

Cuando pasamos un puntero como parámetro de una función por valor pasa lo mismo que con cualquier otro objeto.

Sin embargo, no sucede lo mismo con el objeto apuntado por el puntero, puesto que en ambos casos será el mismo, ya que tanto el puntero como el parámetro tienen como valor la misma dirección de memoria. Por lo tanto, los cambios que hagamos en los objetos apuntados por el puntero se conservarán al abandonar la función.

Ejemplo:

```
#include <iostream>
using namespace std;

double calcula_media(double num1, double num2)
{
 double media = (num1 + num2)/2.;
 return media;
```

```
}
int main(){
 double numero1, numero2;
  cout << "Introduzca dos números reales: ";
  cin >> numero1 >> numero2;

  double resultado = calcula_media(numero1, numero2);

  cout << "La media es " << resultado << endl;
}
```

Diferencia entre puntero paso por valor y paso por referencia

Paso por valor se refiere a un mecanismo para copiar el valor del parámetro de la función a otra variable, mientras que el paso por referencia se refiere a un mecanismo de pasar los parámetros reales a la función.

Parámetro actual

Además, pasar por valor hace una copia del parámetro real. Sin embargo, al pasar por referencia, la dirección del parámetro real pasa a la función.

Asociación con la función

Otra diferencia entre el paso por valor y el paso por referencia es que, al pasar por valor, la función obtiene una copia del contenido real, mientras que, en el paso por referencia, la función accede al contenido de la variable original.

Requisito de memoria

Además, el paso por valor requiere más memoria que pase por referencia.

Tiempo requerido

El requisito de tiempo es otra diferencia entre pasar por valor y pasar por referencia. Pasar por valor requiere más tiempo, ya que implica copiar valores, mientras que pasar por referencia requiere menos tiempo ya que no hay copia.