UNIVERSIDAD DE GUADALAJARA

Seminario de Solución de problemas de Estructuras de Datos I

Investigación II: Punteros

Sección D21

Ramiro Lupercio Coronel

Ismael Iván López Murillo

220975903

22 / 09 / 2020

¿Qué es un puntero?

Los punteros (o apuntadores) son variables que se utilizan para almacenar direcciones de memoria hacia variables declaradas que almacenan valores de un tipo específico.

A todas las variables que se declaran se le asigna un espacio de memoria para almacenar su valor, dicho espacio depende del tipo y compilador y tiene una dirección específica que no se repite con ninguna otra activa. Un puntero en esencia almacena la dirección de la variable, o por así decirlo, “apunta” hacia donde se guarda una variable.

Los punteros deben especificar hacia qué tipo de dato están apuntando, y se declaran de esta manera:

```
tipoDatoApuntado * nombrePuntero;
```

Las operaciones de asignación con punteros difieren un poco con la lógica convencional.

Para apuntar un puntero a una variable se hace de esta forma:

```
puntero = &variableEntera;
```

Donde nuestra variableEntera es un int primitivo. Con el operador & se obtiene la dirección de memoria de nuestra variable y queda implícito que al hacer referencia a puntero estamos hablando de un puntero a int (int *).

Se puede modificar el valor de la variable a la que apunta nuestro puntero, o simplemente acceder a él, para ello debes hacer referencia a nuestro puntero con un * antes

```
* puntero = 0;
```

Al acceder a objetos mediante un puntero (o array, que en esencia es un puntero también), se debe hacer referencia a sus propiedades con el operador ->

```
objetoPuntero->imprimir();
```

Puntero paso por referencia:

Al momento de llamar a una función, se puede hacer de muchas formas distintas en lo que respecta a sus argumentos. Una de ellas es con paso por referencia.

Suponiendo que se tiene una función que convierta nuestro número en un valor absoluto, podemos simplemente asignarle el valor a nuestra variable al resultado que obtenemos de sacar el absoluto con dicho valor de la variable. Sin embargo, también podemos hacer que la función “convierta” nuestro valor a absoluto, sin intermediarios.

Para que los cambios que realicemos a un valor en una función se vean reflejados en la variable que se mandó a la función, tenemos que hacer una llamada de parámetros por referencia.

Así pues, si enviamos una variable a la función, la función usará esta variable como si fuera propia en el mismo ámbito.

Para ello, los parámetros de la función deberán ser declarados obteniendo su dirección de memoria de la variable, no su valor. De esta manera

```
void sumarUno(int &variable);
```

De esta forma, si hacemos referencia dentro de la función a variable++, cuando se acabe la función, en donde sea que se haya llamado a dicha función, el valor habrá sido cambiado.

Es importante mencionar que no se puede hacer paso por referencia con valores constantes, ya que estas no son variables y no tienen un espacio de memoria reservado, que es lo que en realidad mandamos a la función.

Ejemplo:

```
#include <iostream>

#include <math.h>

using namespace std;
```

```
void absoluto(int &);  //Prototipo
int main()
{
	int value;
	cin >> value;
	cout << “El absoluto de “ << value << “ es”;
	absoluto(value);
	cout << value;
	return 0;
}
void absoluto(int &val)
{
	val = sqrt(val*val);
}
```

En este ejemplo, obtenemos el valor absoluto de un valor, pero dentro del main jamás le asignamos un valor distinto a este, sin embargo, en la función a la que llamamos sí. Si no fuera paso por referencia, el valor de value quedaría intacto.

En este caso, a pesar de no usar un puntero explícito, se usa como intermediario entre la llamada a la función y la función en sí.

Puntero paso por valor

Si bien, el paso de parámetro por valor no hace uso de punteros, este es una forma de enviar valores hacia la función.

En este caso, se puede decir que se hace una copia del valor que tenía la variable en el momento de la llamada. Es decir, si tenemos una variable con un 2 almacenado, no se enviará la variable a la función, sino una copia de su valor, es decir, 2. Al momento de querer modificar el valor enviado dentro de la función, el comportamiento de este no tendrá ningún efecto con el valor de la variable original.

```
#include <iostream>

#include <math.h>

using namespace std;

int absoluto(int);  //Prototipo

int main()

{

	int value;

	cin >> value;

	cout << “El absoluto de “ << value << “ es”;

	value = absoluto(value);

	cout << value;

	return 0;

}

int absoluto(int val)

{

	return sqrt(val*val);

}
```

Siguiendo el mismo ejemplo que en el paso por referencia, en este caso se tuvo que modificar la estructura de la función, ya que modificar el valor del argumento de la función dentro de la función no tendrá ningún efecto con el ámbito al que fue llamado. Entonces en este caso se retorna el valor y se asigna a la variable en la llamada.

En este caso no se usa ninguna dirección de memoria ni puntero ni nada, solamente basta con el valor, el compilador hará una copia de dicha variable para declarar otra con el valor que se llamó.

Diferencia entre puntero paso por valor y paso por referencia.

El paso por valor y el paso por referencia son formas de pasar argumentos a funciones, pero tienen muchas diferencias notables.

Al pasar por valor, la variable que fue llamada no se podrá ver modificada en el cuerpo de la función, y como tal solo enviamos el valor.

En cambio, por referencia, vamos a recibir la variable como tal con la que fue llamada la función, podrá ser modificada y leída sin problemas, pero no podremos hacer llamadas a este tipo de función con constantes.

Es ideal que solamente usemos paso por referencia cuando tenemos que modificar el valor de la variable enviada dentro del cuerpo de la función, ya que así limitamos el manejo que tenemos con lógica de punteros.

Programa:

```
#include <iostream>


using namespace std;


int main()

{
```

```
    int numero = 0;
    int  * ptr = NULL;
    cout  << "Ingrese un numero:"<<endl;
    cin >> numero;
    ptr = &numero;
    cout << "Valor de la variable: "<<*ptr<<endl;
    cout << "Direccion de la variable: "<<&ptr << endl;
    return 0;
}
```

En este programa tengo dos anotaciones:

- En mi caso, por mi compilador tuve que cambie null por NULL, aunque lo ideal es usar nullptr, ya que es un puntero.
- *ptr = &numero es una línea incorrecta, ya que estamos diciendo que el valor de la variable de lo que apunta ptr será la dirección de número, y eso es incorrecto, ya que ptr apunta a null. Entonces lo ideal es decir que ptr = &numero, que significa que nuestro puntero apuntará a la dirección de número, pudiendo manipularlo de forma correcta.

Conclusión

Los punteros en sí suelen ser un tema complicado en la programación, especialmente cuando hablamos de C++, ya que C++ permite manipular demasiado a nivel de memoria las variables usando este tipo de apuntadores. Cosas extrañas como apuntadores triples (int ***) o apuntadores a funciones, apuntadores a direcciones, entre otros, suelen ser problemáticas a lo que cualquier programador de C++ suele enfrentarse. En otros lenguajes, como Java, este problema no existe ya que todo es una referencia, todo es un puntero (excepto los datos primitivos).

Comprender la lógica más básica de los punteros y ponerlo en práctica, ir probando cosas poco a poco es base fundamental para no tener problemas después con los punteros.

Conceptos como espacio de memoria, dirección de memoria, puntero, *, & y demás tenemos que tenerlos bien en claro al momento de programar.

Bibliografía:

- Adrigm. (2013, April 15). Punteros y referencias. Retrieved September 17, 2020, from https://www.genbeta.com/desarrollo/punteros-y-referencias
- Coding Games and Programming Challenges to Code Better. (n.d.). Retrieved September 17, 2020, from https://www.codingame.com/playgrounds/51214/manejo-dinamico-de-memoria-y-polimorfismo-practica-4/punteros-en-c