# **Punteros**

**Un puntero es una variable que almacena direcciones**, a diferencia del resto de las variables que almacenan valores de un tipo determinado (números o caracteres).

Más allá de esta importante diferencia, los punteros no encierran ningún “misterio”: son recipientes donde se puede escribir o leer datos, lo mismo que se hace con el resto de las variables.

Para declarar una variable puntero hay que anteponer el \* en la declaración.

int \*pi;

float \*pf;

pi es la declaración de un puntero a entero, y por lo tanto podrá almacenar la dirección física de una variable de tipo entero;

pf es la declaración de un puntero a float, y por lo tanto podrá almacenar la dirección física de una variable de tipo float;

Mientras no se le asigne nada contendrá un valor indeterminado (“basura”) al igual que el resto de las variables.

No es la operación más frecuente el trabajo con direcciones de memoria físicas, pero en C podemos hacerlo, y resulta de mucha utilidad en algunos casos.

Para obtener la dirección de una variable hay que usar el **operador de dirección &** antepuesto a un nombre de variable. Por ejemplo:

int var, \*pint;

var=15;

pint=&var;

cout<<var<<endl;

cout<<\*pint<<endl;

La salida en este caso será:

15

15

15 es el valor que se le asignó a var, esto es, lo que contiene var.

15 es el valor que se le asignó a var, pero llegamos a él por medio de la dirección que contiene pint.

Veamos el código en detalle:

var=15; // se asigna el valor 15 a var

pint=&var; // se asigna a pint la dirección de var

cout<<var<<endl;// se muestra lo que hay en var

cout<<\*pint; // se muestra lo que hay en la dirección que

//contiene pint

El **operador \*** se llama operador de indirección, y permite acceder al contenido de una dirección. Como pint contiene la dirección de var, \*pint nos lleva al contenido de var.

Del mismo modo, en el siguiente fragmento de código se modifica el valor de var por medio de pint.

int var, \*pint;

pint=&var;

\*pint=15;

cout<<var;

Cuando escuchamos o leemos que un puntero “apunta” a una dirección, nos estamos refiriendo a la dirección que ese puntero contiene; en el ejemplo visto como pint “apunta” a var, por medio de \*pint podemos leer o escribir en var.

En resumen, lo importante es lo siguiente:

- **Las variables tiene un nombre por medio del cual el programador trabaja con ellas, y una dirección física en la memoria de la máquina.** El nombre que le damos nos interesa sólo a nosotros, ya que nos simplifica el trabajo. El programa ejecutable no trabaja con los nombres, sino que traduce todo a posiciones relativas de memoria, es decir a direcciones físicas.

- **C nos permite trabajar directamente con las direcciones** que las variables ocuparán (en tiempo de ejecución) en la memoria. Podemos entonces escribir y leer una determinada posición de la memoria por medio de los punteros. Recordar que los punteros son variables que almacenan direcciones; si ponemos delante de una variable puntero el operador \* podremos leer o escribir en la dirección que está almacenada en el puntero (la dirección a la que apunta el puntero).

**Punteros como parámetros de funciones**

Los punteros son particularmente útiles cuando necesitamos que una función modifique una o más variables locales a otra función. En ese caso la función propietaria de las variables que se quieren modificar llama a la función que las modificará enviando como parámetros las direcciones de esas variables. Siempre que se envía un parámetro por dirección, este debe ser recibido por una variable puntero.

int main(){

int var1, var2;

modifica\_var(&var1, &var2); //se llama a modifica\_var() y

//se envía lasdirecciones de //las variables

cout<<var1<<”\t”<<var2; // se mostrarán los valores

//asignados en modifica\_var()

return 0;

}

void modifica\_var(int \*pvar1, int \*pvar2){

\*pvar1=8;

\*pvar2=10;

}

Nota: **sólo** deben enviarse direcciones como parámetros cuando se necesite que la función llamada modifique el valor de una o más variables.

**Punteros y vectores**

Un caso especial de los punteros son los nombres de los vectores.

Cuando declaramos un vector, por ejemplo int v[10], reservamos espacio en memoria para 10 variables enteras. No podemos hacer ninguna operación si no decimos específicamente cuál de los elementos del vector queremos mostrar o escribir.

El nombre del vector, sin subíndice, almacena la dirección de inicio de ese vector en la memoria. Como ya sabemos que el único tipo de variable que puede almacenar una dirección es una variable puntero, el nombre del vector es entonces un puntero, pero a diferencia del resto de los punteros es un **puntero constante**: contiene la dirección de memoria donde empieza el vector y no puede modificarse.

Siguiendo el razonamiento anterior, podemos afirmar que:

**v es igual a &v[0]**

lo que implica que podríamos usar indistintamente ambas notaciones con el mismo resultado. Veamos un ejemplo

#include <iostream>

using namespace std;

void cargarVector(int \*, int);

void mostrarVector(int \*, int);

int main(){

int vec[5];

cargarVector(&vec[0],5);

mostrarVector(vec,5);

return 0;

}

void cargarVector(int \*v, int tam){

int i;

for(i=0;i<tam;i++){

cin>>v[i];

}

}

void mostrarVector(int \*v, int tam){

int i;

for(i=0;i<tam;i++)

{

cin>>v[i];

cout<<endl;

}

}

Como un vector es una variable compuesta, no podemos mandarlo como parámetro por valor de una función, como lo hacemos con la variables simples. La única forma de enviar como parámetro un vector es por dirección.

La función main() envía la dirección de inicio del vector, y las funciones llamadas reciben esa dirección en una variable puntero; luego como las funciones tiene la dirección pueden escribir o leer los datos que contiene el vector utilizando el nombre del puntero definido como parámetro y la notación de subíndice.

La función void cargarVector(int \*, int), recibe la dirección de inicio del vector y su tamaño. La almacena en un puntero (v) y en una variable entera (tam) respectivamente. Luego asigna a cada posición un valor que se ingresa por teclado.

La función void mostrarVector(int \*, int), recibe la dirección de inicio del vector y su tamaño. La almacena en un puntero (v) y en una variable entera (tam) respectivamente. Luego muestra por pantalla el contenido de cada una de las posiciones del vector.

Como lo que las funciones necesitan es la dirección de inicio del vector, no les interesa ni altera su funcionamiento que esa dirección sea enviada haciendo referencia explícita a la dirección del elemento 0 (&vec[0]), o utilizando solamente el nombre del vector (vec).

Nota: en la explicación se utilizó un vector de tipo entero. Las consideraciones expuestas son aplicables para vectores de cualquier tipo de dato.