**Apuntes de la clase de Programacion Fecha: 7 octubre de 2013**

Lic. Osmar Cabrera H.

**Punteros (Se recomienda leer el capítulo 5 del libro de Kernighan y Ritchie)**

Básicamente, el puntero es una variable cuyo contenido es la dirección de otra variable. Los punteros son muy usados en C, y son una de sus características principales.

Una máquina típica tiene un arreglo de celdas de memoria numerados o direccionados de forma consecutiva que pueden ser manipuladas de forma individual o en grupos contiguos. Una situación común es que cualquier byte puede ser un char, un par de celdas de un byte puede ser tratado como un short int, y cuatro bytes adyacentes forman un long. Un puntero es un grupo de celdas (a menudo dos o cuatro) que puede contener una dirección. Así que si es un char y es un puntero que apunta a la misma, se podría representar la situación de la siguiente manera:

**Variable**

**Puntero**

El operador & da la dirección de un objeto (variable), entonces la sentencia:

asigna al puntero la dirección de la variable . Entonces se dice que “ apunta a ”. El operador “&” sólo se aplica a elementos almacenados en memoria (como variables y arreglos).

El operador \* es el “desreferenciador”. Al aplicarlo a un puntero, se accede al objeto al que apunta. Un ejemplo del uso de estos operadores es el siguiente:

int x = 1, y = 2, z[10];

int \*ip; /\* ip es un puntero a un int \*/

ip = &x; /\* ip apunta a la variable x \*/

y = \*ip; /\* ahora y es 1 \*/

\*ip = 0; /\* ahora x es 0 \*/

ip = &z[0]; /\* ahora ip apunta a z[0] \*/

*Observación*: un puntero está restringido a apuntar a objetos de un tipo en específico (por eso se definen como un tipo de dato).

Si , entonces las siguientes operaciones son válidas (para modificar el valor de o para hacer otras operaciones):

\*ip = \*ip + 10; //incrementa x en 10

y = \*ip + 1; //es lo mismo que y=x+1

\*ip += 1; //incremental el valor de x en 1

++\*ip; //también incrementa x en 1;

(\*ip)++ //se usa el paréntesis para incrementar x. Si no se coloca, se incrementa el valor del puntero.

un tipo de dato).

Si , entonces las siguientes operaciones son válidas (para modificar el valor de o para hacer otras operaciones):

\*ip = \*ip + 10; //incrementa x en 10

y = \*ip + 1; //es lo mismo que y=x+1

\*ip += 1; //incremental el valor de x en 1

++\*ip; //también incrementa x en 1;

(\*ip)++ //se usa el paréntesis para incrementar x. Si no se coloca, se incrementa el valor del puntero.

Como los punteros son variables, se pueden hacer operaciones sin “desreferenciar”. Por ejemplo: iq = ip (con esto, e apuntan al mismo objeto).

**Paso por valor y paso por referencia:**

*Ejemplo 1:* ¿Qué valor se imprime en pantalla en cada uno de estos programas?

|  |  |
| --- | --- |
| #include<stdio.h>  void sub(int);  main(){  int a=3;  sub(a);  printf(“%d”,a);  }  void sub(int a){  a=10;  } | #include<stdio.h>  void sub(int \*);  main(){  int a=3;  sub(&a);  printf("%d",a);  }  void sub(int \*p){  \*p=10;  } |

Observación: para imprimir el valor de un puntero (en hexadecimal), se utiliza printf(“%p”,pa);

*Ejemplo 2:* ¿Son formas equivalentes de definir la función swap?

|  |  |
| --- | --- |
| void swap(int x, int y){  int temp;  temp = x;  x = y;  y = temp;  } | void swap(int \*px, int \*py){  int temp;  temp = \*px;  \*px = \*py;  \*py = temp;  } |

x

px

**En la función swap**

**En la función que llama**

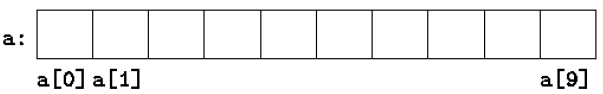
y

py

**Punteros y vectores**

En C, existe una fuerte relación entre los punteros y los vectores (en general, los arreglos); y la mayoría de las veces pueden usarse indistintamente (en realidad los nombres de los vectores son punteros *constantes* que apuntan al primer elemento del vector).

La declaración int a[10]; define un bloque de 10 elementos consecutivos llamados a[0],…,a[9].

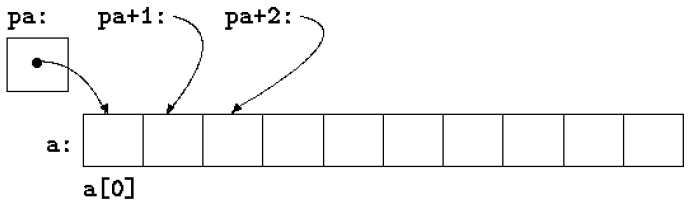


La notación se refiere al elemento del vector. Si es un puntero a int, entonces:

int \*pa;

pa = &a[0];

hace que apunte el elemento cero de . Es decir, contiene la dirección de .

**

Si apunta a un elemento de , entonces apunta a la siguiente posición, y apunta a un elemento alejado de (esto se conoce como *aritmética de punteros*). Por definición, el valor de una variable o expresión de tipo vector es la dirección del elemento cero del vector. Por lo tanto, estas expresiones son equivalentes:

pa = &a[0]; pa = a;

Además, las siguientes notaciones son equivalentes:

* a[i] → \*(a+i)
* &a[i] → a+i
* Si es un puntero: pa[i] → \*(pa+i)

Si se supone que p=vect, la relación entre punteros y vectores puede resumirse como se indica en las líneas siguientes:

* \*p equivale a vect[0], a \*vect y a p[0]
* \*(p+1) equivale a vect[1], a \*(vect+1) y a p[1]
* \*(p+2) equivale a vect[2], a \*(vect+2) y a p[2]

**Observación**: Existe una diferencia entre el nombre de un vector y un puntero: un puntero es una variable, por lo que pa=a and pa++ son operaciones *legales*. Pero el nombre de un vector (y en general, el de un arreglo) es una constante, por lo que a=pa and a++ son operaciones *ilegales*.

**Vectores y funciones**

Cuando el nombre de un vector se pasa a una función, lo que se pasa es la ubicación del elemento inicial. Dentro de la función llamada, este argumento es una variable local, y por lo tanto el nombre de un arreglo como parámetro es un puntero; esto es, una variable que contiene una dirección (paso por referencia). Un ejemplo sería la función strlen:

#include<stdio.h>

int strlen(char []);

int strlen2(char \*);

int main(){

char cad[30];

gets(cad);

int a;

a=strlen(cad);

a=strlen2(cad);

printf("%d\n",a);

return 0;

}

|  |  |
| --- | --- |
| int strlen(char cad1[]){  int i=0;  while(cad1[i]!='\0') i++;  return i;  } | int strlen2(char \*pc){  int i=0;  while((\*pc++)!='\0') i++;  return i;  } |

Las llamadas como:

strlen("hello, world"); /\* constante de tipo cadena \*/

strlen(array); /\* char array[100]; \*/

strlen(ptr); /\* char \*ptr; \*/

funcionan correctamente.

**Observación importante:** la instrucción char cad[30];ya reserva un espacio en memoria para ese vector. Si sólo definimos un puntero de tipo char, no se reserva memoria. La asignación dinámica de memoria (asociada a punteros) se verá más adelante.

**Punteros a carácter y funciones**

Una constante “cadena”, escrita como: “Hola”, es un arreglo de caracteres. Estas constantes pueden usarse como parámetros de funciones. Una forma común de usarlas es a través de la función printf:

printf("Hola mundo\n");

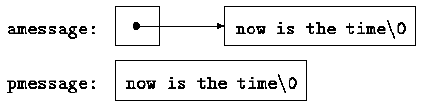
Cuando una constante cadena aparece en un programa, lo hace a través de un puntero a carácter; y printf recibe un puntero que apunta al inicio del arreglo de caracteres. Estas constantes no solo se restringen a argumentos de funciones. Si se declara como: char \*pc; entonces:

pc = "hola que tal"; asigna a un puntero al arreglo de caracteres. Esto NO es una copia de la cadena, sólo es una relación entre punteros.

Existe una diferencia importante entre estas definiciones:

char amesage[] = "now is the time"; /\* un vector de caracteres \*/

char \*pmessage = "now is the time"; /\* un puntero \*/

 es un arreglo, lo suficientemente grande para contener a la cadena (se pueden modificar los elementos del vector); mientras que es un puntero, inicializado a apuntar una constante cadena. El puntero puede cambiar de objetivo, pero el resultado es indefinido si se intenta modificar el contenido de la cadena.

Ejemplo:

#include<stdio.h>

void copia(char \*, char \*);

int main(){

char cad1[30];

copia(cad1,"hola chau");

printf("%s\n",cad1);

return 0;

}

void copia(char \*c1, char \*c2){

int i=0;

while(\*(c2+i)!='\0'){

\*(c1+i)=\*(c2+i);

i++;

}

\*(c1+i)='\0';

}

**Relación entre matrices y punteros**

En el caso de las matrices la relación con los punteros es un poco más complicada. Supóngase una declaración como la siguiente:

int mat[5][3], \*q;

Se debe prestar especial atención a lo siguiente: una matriz es un **vector de vectores**.

Recuérdese también que, por la relación entre vectores y punteros, (mat+i) apunta a mat[i]. Recuérdese que la fórmula de direccionamiento de una matriz de N filas y M columnas establece que la dirección del elemento (i, j) viene dada por:

dirección (i, j) = dirección (0, 0) + i\*M + j

Si la matriz tiene M columnas y si se hace q = &mat[0][0] (dirección base de la matriz), el elemento mat[i][j] puede ser accedido de varias formas. Basta recordar que dicho elemento tiene por delante i filas completas, y j elementos de su fila:

\*(q + M\*i + j) // fórmula de direccionamiento

\*(mat[i] + j) // primer elemento fila i desplazado j elementos

(\*(mat + i))[j] // [j] equivale a sumar j a un puntero

\*((\*(mat + i)) + j) // accede al elemento mat[i][j]

Todas estas relaciones tienen una gran importancia, pues implican una correcta comprensión de los punteros y de las matrices. De todas formas, hay que indicar que las matrices no son del todo idénticas a los vectores de punteros: Si se define una matriz explícitamente por medio de vectores de punteros, las filas pueden tener diferente número de elementos, y no queda garantizado que estén contiguas en la memoria (aunque se puede hacer que sí lo sean). No sería pues posible en este caso utilizar la fórmula de direccionamiento y el acceder por columnas a los elementos de la matriz.

En realidad, una matriz se representa en memoria como un vector, donde se colocan las filas una al lado de otra; por lo tanto una matriz es un vector, donde sus elementos son vectores. Por ejemplo, una matriz A de 10x10 se representa en memoria por un vector de 100 elementos.

*La matriz es un arreglo de arreglos de una dimensión*. Como el nombre de un arreglo es un puntero a su primer elemento (y ese elemento es un arreglo de una dimensión), A es realmente un puntero al primer arreglo de 10 elementos (más información sobre estos temas en http://stackoverflow.com/questions/546860/passing-arrays-and-matrices-to-functions-as-pointers-and-pointers-to-pointers-in).

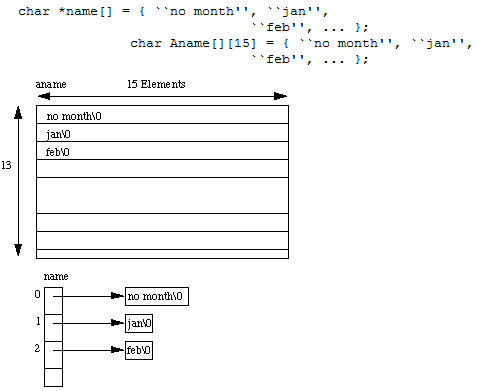
**Formas de pasar una matriz a una función**

En el ejemplo, se pasa el nombre de una matriz de 10x10:

void funcion(int (\*arreglo)[10]);  
void funcion(int arreglo[][10]);  
void funcion(int arreglo[10][10]);  
void funcion(int arreglo[42][10]);

**Puntero a arreglos vs. Arreglo de punteros**

* int (\*arreglo)[n]; //Un puntero a un arreglo de n enteros.
* int \*arreglo[n]; //Un arreglo de punteros a enteros

****

**Ejemplo de lectura e impresión de datos**

#include<stdio.h>

int main(){

int i,j,m,n;

printf("Ingrese la cantidad de filas y columnas de la matriz:");

printf("\nFilas: "); scanf("%d",&m);

printf("Columnas: "); scanf("%d",&n);

int mat[m][n];

int \*p[m],(\*r)[n],\*\*q;

r=mat;

q=p;

for(i=0;i<m;i++) \*(p+i)=\*(mat+i);

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

for(j=0;j<n;j++){

printf("mat[%d][%d]: ",i,j); scanf("%d",(\*(p+i)+j));

}

}

printf("\nImprimiendo con p\n");

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

for(j=0;j<n;j++){

printf("%d\t",\*(\*(p+i)+j));

}

printf("\n");

}

printf("\nImprimiendo con r\n");

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

for(j=0;j<n;j++){

printf("%d\t",(\*(r+i))[j]);

}

printf("\n");

}

printf("\nImprimiendo con q\n");

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

for(j=0;j<n;j++){

printf("%d\t",\*(\*(q+i)+j));

}

printf("\n");

}

return 0;

}