MODULO 6

Punteros

Breve descripción

En este documento se explica qué son las variables puntero y para qué empezaremos a utilizarlas.

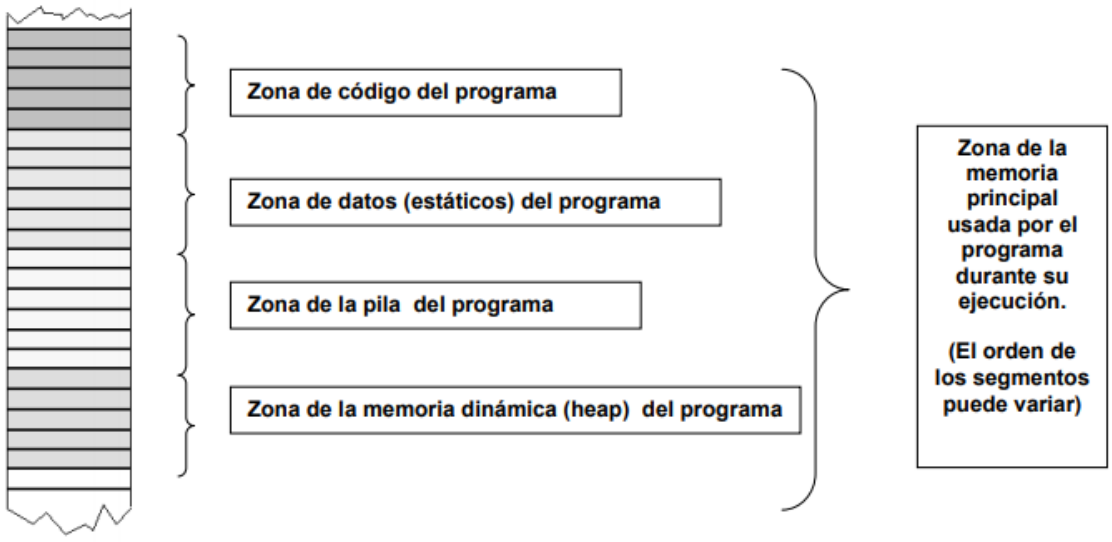
Versión 1.1

Martin Jerman
Martin.jerman@inspt.utn.edu.ar

Introducción

Todo dato con que trabaja la computadora, (así como toda instrucción cuando se ejecuta), está en la memoria principal. Tanto los datos como las instrucciones están codificadas como sucesiones de ceros y unos. Al momento de ejecutar un programa, para ese programa en particular se distinguen cuatro zonas distintas, todas ubicadas en la memoria principal de la computadora. Cada zona es un 'segmento' y tiene una función determinada. Los segmentos son los siguientes:

- Segmento de Código: aquí está el programa que se está ejecutando, ya traducido a lenguaje de máquina.
- Segmento de Datos: aquí se ubican las variables del programa principal (las que hemos declarado en main).
- Segmento de Pila (o Stack): este segmento, del que hablaremos después es fundamental para que podamos llamar a funciones.
- Segmento Extra (o heap o montículo): sirve para utilizar la memoria dinámica. De él nos ocuparemos más tarde.



Las direcciones de memoria

Vamos a considerar en particular a los datos. ¿De qué modo se almacenan en la memoria? Primero hay que hacer algunas consideraciones acerca de la estructura de la memoria. La memoria de la computadora se compone de celdas. Cada celda es un byte, es decir que está formada por 8 bits. Todas las celdas de la memoria tienen el mismo tamaño, y son idénticas entre sí. Solo pueden diferenciarse por él número que tienen asignado.

El número de posición de una celda es la dirección de la celda. Toda celda de la memoria tiene una dirección asignada. Esa dirección es única para cada celda y no puede ser cambiada. En ningún momento una computadora tiene 'celdas vacías'. En toda celda hay bits. Un bit puede tomar valor cero o uno. Mediante la codificación con dígitos binarios la computadora trata tanto los datos como las instrucciones.

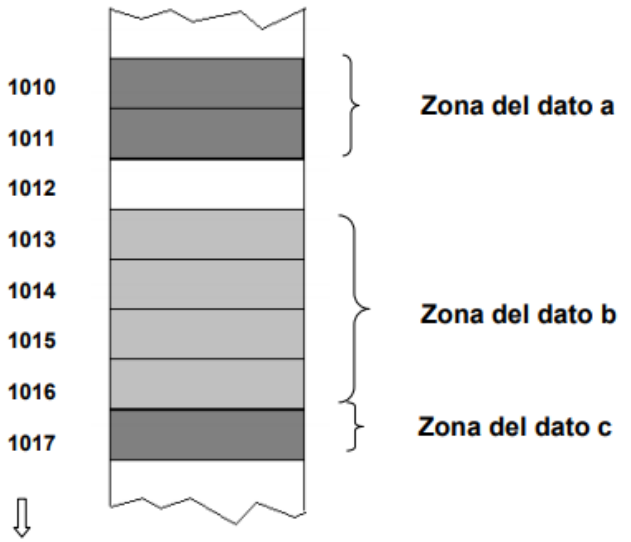
Toda celda de la memoria tiene una dirección asignada. Esa dirección es única para cada celda y no puede ser cambiada.

Cada dato contenido en la memoria ocupa una o más celdas, dependiendo del tamaño de este. Por ejemplo, los caracteres ocupan una sola celda de memoria, los enteros ocupan al menos 2 celdas,

los reales ocupan al menos 4 celdas, y otros datos ocupan más celdas. En todo caso, lo que siempre se verifica es que un dato, del tipo que sea, **ocupa un conjunto contiguo de celdas**.

La dirección de la primera celda de un dato (dirección de comienzo de la zona del dato) es la dirección del dato. En el dibujo lateral, el dato `A` ocupa las celdas `1010` y `1011`. La primera celda que ocupa es la `1010`, entonces la dirección de `A` es `1010`. El dato `B` ocupa las celdas `1013`, `1014`, `1015` y `1016`; la dirección de `B` es `1013`. La dirección del dato `C` es `1017`.

En `C`, el operador & se utiliza para indicar la dirección de una variable determinada.



Para el ejemplo anterior:

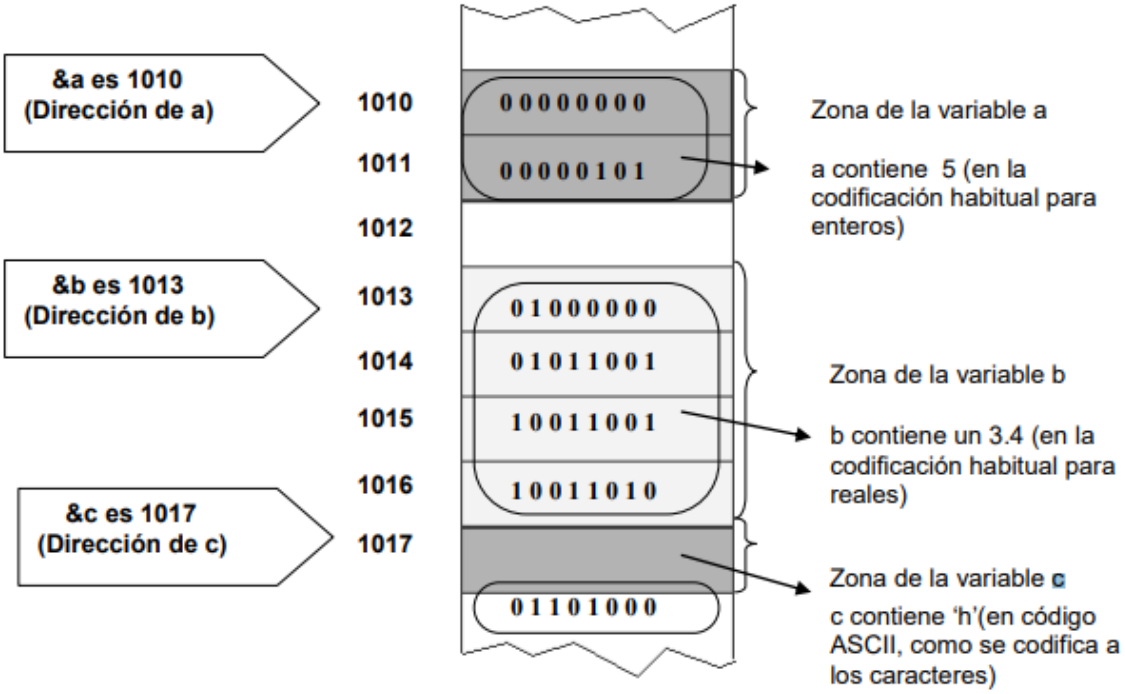
`&a` es `1010`

`&b` es `1013`

`&c` es `1017`

No se debe confundir nunca la dirección de una variable con el contenido de esta.

En un grafico más cercano a lo que ocurre en la realidad:



En el gráfico anterior se ha indicado el contenido de cada variable tal como se almacena; a la derecha se especifica el valor correspondiente a cada secuencia de bits (los caracteres se codifican en ASCII, los enteros en complemento a 2, los reales en la notación IEEE 754, etc.). Según la situación graficada antes a modo de ejemplo, a contiene el dato `5`. La dirección de a, llamada `&a` es `1010`, `b` contiene el dato `3.4`. La dirección de `b` es `&b`, con valor `1013` c contiene el dato `'h'`. La dirección de `c` es `&c`, con valor `1017`.

Variables puntero

Los punteros son variables que almacenan la dirección de otra variable. Su declaración tiene la siguiente forma:

```
tipo * identificador; //establece que identificador es una variable puntero a dato tipo.
```

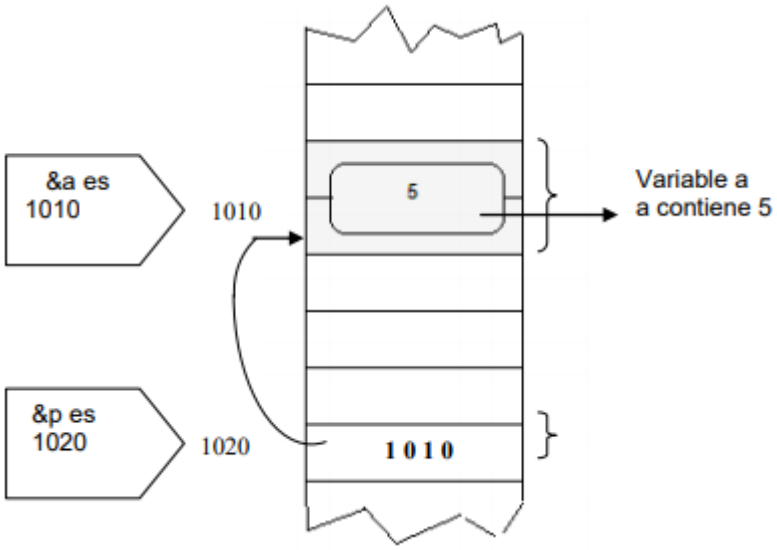
Ejemplo:

```
int *p;
```

indica que p es un puntero a entero. Si `p` se ha declarado puntero, entonces `*p` es el modo de referirse al contenido de la dirección almacenada en `p`. El operador `*` es el operador de 'des referencia'.

Des referenciar una variable puntero es obtener el dato almacenado en la dirección contenida en el puntero. Asignarle un valor a `p` es darle el valor de una dirección de un dato tipo entero. Así, si se tiene `int a=3;` la asignación `p=&a` le asigna a `p` la dirección de `a`.

La representación esta situación puede ser la siguiente:



Variable puntero **p** (representación simplificada, la dirección ocupa más de un byte) contiene la dirección de **a**, entonces **p** apunta a la variable **a**.

Ahora, el contenido de la variable a se puede manipular usando tanto el identificador a como con la notación *p. Por ejemplo, se puede hacer:

```
a=9;
```

o bien

```
*p=9;
```

Puede emitirse el contenido de a usando

```
printf ("%d", a);
```

o bien

```
printf("%d", *p);
```

y puede ingresarse el contenido de a usando

```
scanf("%d", &a);
```

o bien

```
scanf("%d",p);
```

Ya que el `scanf` requiere que se indique la dirección en la cual se almacenará el dato que ingresa, y en el código anterior, **p contiene esa dirección**.

Otro ejemplo:

```
char *q
```

establece que q es una variable puntero a char.

Si se tiene

```
char c;
c='r';
```

Se puede efectuar

```
q=&c;
*q='s';
printf("%d", c); //y se emitirá 's'
```

¿Cómo sabe la computadora, cuando se des referencia un puntero?, si debe tomar una, ¿dos o más celdas de la memoria a partir de la indicada en el puntero?, ¿y cómo tratar al dato contenido (es decir que codificación se ha usado)? La respuesta está en que, cuando se declara una variable tipo puntero, se **indica el tipo de dato apuntado**. Cuando se declara `int * p;` se está indicando que, a partir de la dirección que este contenida en p se deben tomar tantas celdas como corresponde a un entero, e interpretar el contenido según la codificación de enteros. Si, en cambio, la codificación es:

```
float *q;
```

Se indica que, a partir de la dirección contenida en `q`, hay que tomar tantos bytes como corresponde a un `float`, e interpretar ese contenido con notación de `float`. El tamaño que corresponde a una variable o a un tipo determinado se puede conocer utilizando el operador `sizeof`. Así, mediante `sizeof(a)` se puede conocer el tamaño em bytes de la variable `a`, o bien se puede efectuar `sizeof(float)` para conocer el tamaño de un dato tipo float, etc. Cómo se puede comprobar comparando `sizeof (int *)` con `sizeof (float *)` y con `sizeof (char *)`, todos los punteros tienen el mismo tamaño, ya que todas las direcciones de memoria miden lo mismo. Las variables tipo punteros pueden apuntar diversos tipos de datos, incluso a punteros, y también hay punteros a void. La notación

```
void * t;
```

indica que t es un puntero, pero que no se especifica el tipo de dato apuntado. Este tipo de punteros **no puede des referenciarse** (porque no hay información acerca del tipo de dato apuntado). Al momento de la definición de un puntero, éste no tiene un contenido establecido. Decimos que **esta indeterminado**, es decir que el lugar de memoria que ocupa no es inicializado automáticamente con ningún valor particular. El valor nulo de los punteros es `0` (cero), y la constante asociada a ese valor se llama `NULL`. Es decir que al hacer

```
int *p; //p esta indeterminado; si se intenta efectuar *p se producirá un error
int a=1;
p=NULL; //o p=0, asigna el valor nulo a p; si se intenta *p dará un error
p=&a; //ahora si p apunta a un lugar de memoria y se puede efectuar *p
```

Sumar una constante a un puntero

Si se tiene:

```
int *p, a=5;
```

```
p=&a;
```

y se efectúa

```
p=p+1;
```

p **avanzará la cantidad de celdas que corresponde al tamaño de un entero**. Si p contenía 1000 y cada int utiliza 4 celdas, entonces p+1 vale 1004, p+2 vale 1008, y así sucesivamente.

Consideremos ahora:

```
float *q, b;
```

```
q= &b;
```

si `q` contiene `2000`, y cada float utiliza 6 bytes, entonces `q+1` vale `1006`, `q+2` vale `1012`, etc. Es decir, que al sumarle a una variable puntero q, una constante positiva entera x, la variable q, incrementa su dirección en `x * sizeof(tipoapuntado_por_q )`. En otras palabras, si `int` ocupara 2 bytes, `char` ocupara 1 byte, y `float` ocupara 6 bytes, entonces sumar 1 a una variable puntero p la hace avanzar:

- 4 bytes si p es puntero a `int`
- 1 byte se `p` es un puntero a `char`
- 6 bytes si p es un puntero a `float`

análogamente para los otros tipos.

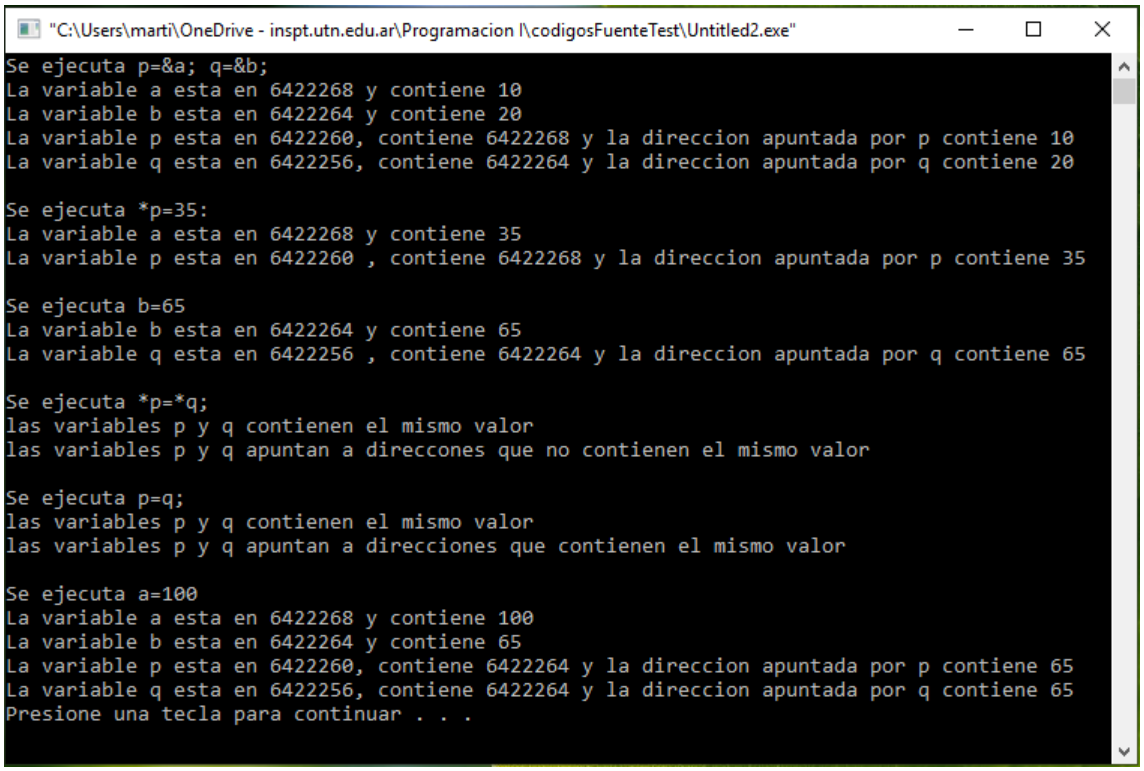
Ejemplo de análisis

Para entender mejor cómo se comportan las variables puntero, veamos el siguiente código y análisis de su salida por pantalla:

```
#include<stdio.h>
int main()
{
	int a, b, *p, *q;
	a=10;
	b=20;
	p=&a;
	q=&b;
	printf("Se ejecuta p=&a; q=&b;\n");
	printf("La variable a esta en %d y contiene %d\n", &a, a);
	printf("La variable b esta en %d y contiene %d\n", &b, b);
	printf("La variable p esta en %d, contiene %d y la direccion apuntada por p contiene %d\n",&p, p,*p);
	printf("La variable q esta en %d, contiene %d y la direccion apuntada por q contiene %d\n",&q, q,*q);
	printf("\nSe ejecuta *p=35:\n");
	*p=35;
	printf("La variable a esta en %d y contiene %d\n", &a, a);
	printf("La variable p esta en %d , contiene %d y la direccion apuntada por p contiene %d\n",&p, p,*p);
```

```
	printf("\nSe ejecuta b=65\n");
	b=65;
	printf("La variable b esta en %d y contiene %d\n", &b, b);
	printf("La variable q esta en %d , contiene %d y la direccion apuntada
por q contiene %d\n ",&q, q,*q);
	printf ("\nSe ejecuta *p=*q;\n");
	*p=*q;
	if (*p==*q)
		printf ("las variables p y q contienen el mismo valor\n");
	else
		printf ("las variables p y q no contienen el mismo valor\n");
	if (p==q)
		printf ("las variables p y q apuntan a direcciones que contienen
el mismo valor\n");
	else
		printf ("las variables p y q apuntan a direccones que no contienen
el mismo valor\n");
	printf ("\nSe ejecuta p=q;\n");
	p=q;
	if (*p==*q)
		printf ("las variables p y q contienen el mismo valor\n");
	else
		printf ("las variables p y q no contienen el mismo valor\n");
	if (p==q)
		printf ("las variables p y q apuntan a direcciones que contienen
el mismo valor\n");
	else
		printf ("las variables p y q apuntan a direccones que no contienen
el mismo valor\n");
	printf("\nSe ejecuta a=100\n");
	a=100;
	printf("La variable a esta en %d y contiene %d\n", &a, a);
	printf("La variable b esta en %d y contiene %d\n", &b, b);
	printf("La variable p esta en %d, contiene %d y la direccion apuntada por
p contiene %d\n",&p, p,*p);
	printf("La variable q esta en %d, contiene %d y la direccion apuntada por
q contiene %d\n",&q, q,*q);
	system ("pause");
	return 0;
}
```

Este programa nos muestra la siguiente salida:



Haga el seguimiento del código fuente y compárelo con la salida para ver el comportamiento de los punteros.