Apuntadores

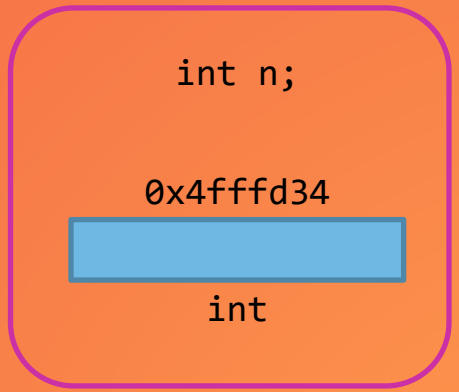
Punteros o apuntadores

- Una variable puntero es una variable que contiene direcciones de otras variables.

- Un puntero es una variable que contiene una dirección de memoria, y utilizando punteros su programa puede realizar muchas tareas que no sería posible utilizando tipos de datos estándar.

Direcciones en memoria

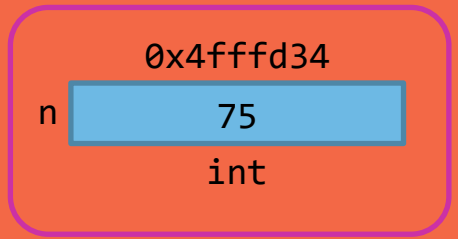
- Cuando una variable se declara, se asocian tres atributos fundamentales con la misma:
  - nombre
  - tipo
  - dirección en memoria.



Asocia al nombre **n**, el tipo **int** y la dirección de alguna posición de memoria donde se almacena el valor de n

Direcciones en memoria

- Esta caja representa la posición de almacenamiento en memoria.
- El nombre de la variable está a la izquierda de la caja, la dirección de variable está encima de la caja y el tipo de variable está debajo en la caja.
- Si el valor de la variable se conoce, se representa en el interior de la caja.



Al valor de una variable se accede por medio de su nombre.
Por ejemplo, se puede imprimir el valor de n con la sentencia:

printf(“%d”,n);

A la dirección de la variable se accede por medio del *operador de dirección* **&**
Por ejemplo, se puede imprimir la dirección de n con la sentencia:

printf(“%p”, &n);

Direcciones en memoria

```
#include <stdio.h>
void main()
{
          int n = 75;
          printf("n = %d\n",n);          //Visualiza el valor de n
          printf("&n = %p\n",&n);        //Visualiza dirección de n
}
```

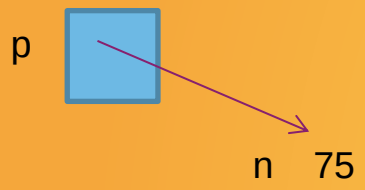
- Ejecución:

```
n = 75
&n = 0x4fffd34

Nota: 0x4fffd34 es una dirección en código hexadecimal
      “0x” es el prefijo correspondiente al código hexadecimal
```

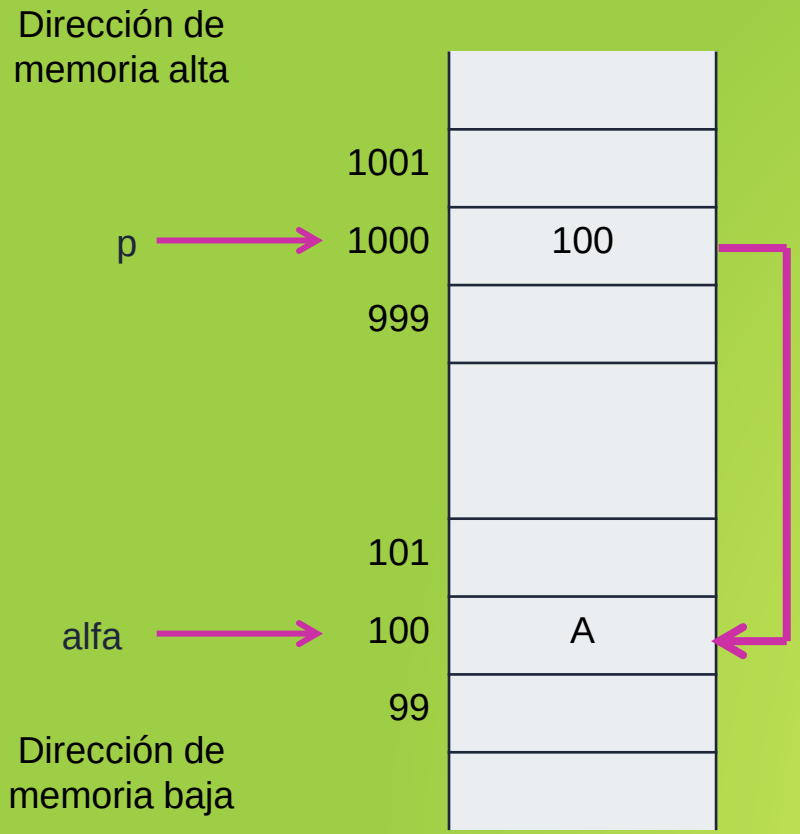
Puntero (Apuntador)

- Los punteros se rigen por estas reglas básicas:
  - Un puntero es una variable como cualquier otra.
  - Una variable puntero contiene una dirección que apunta a otra posición en memoria.
  - En esa posición se almacenan los datos a los que apunta el puntero.
  - Un puntero apunta a una variable de memoria.



El valor de un puntero es una dirección. La dirección depende del estado de la computadora en la cual se ejecuta el programa.

Puntero (Apuntador)



p contiene el valor 100, que es la dirección de alfa

*p es el valor del elemento al que apunta p.

Por consiguiente, *p toma el valor 'A'

Puntero (Apuntador)

```c
#include <stdio.h>
void main()
{
        int n = 75;
        int *p = &n;        //p variable puntero, tiene dirección de n
        printf("n = %d, &n = %p, p = %p\n",n,&n,p);
        printf("&p = %p\n",&p);
}
```

```
n = 75, &n = 0x4fffd34, p = 0x4fffd34
&p = 0x4fffd30
```

La variable p se denomina «puntero» debido a que su valor «apunta» a la posición de otro valor.

Declaración de punteros

- Al igual que cualquier variable, las variables punteros han de ser declaradas antes de utilizarlas.
- La declaración de una variable puntero debe indicar al compilador el tipo de dato al que apunta el puntero; para ello se hace preceder a su nombre con un asterisco (*), mediante el siguiente formato:

```
<tipo de dato apuntado> *<identificador de puntero>
```

- Algunos ejemplos de variables punteros:

```
int *ptr1;          //Puntero a un tipo de dato entero (int)
long *ptr2;         // Puntero a un tipo de dato entero largo (long int)
char *ptr3;         // Puntero a un tipo de dato char
Float *f;           // Puntero a un tipo de dato float
```

Un operador * en una declaración indica que la variable declarada almacenará una dirección de un tipo de dato especificado.

Inicialización de punteros

- Al igual que otras variables, C no inicializa los punteros cuando se declaran y es preciso inicializarlos antes de su uso.
- La inicialización de un puntero proporciona a ese puntero la dirección del dato correspondiente.
- Después de la inicialización, se puede utilizar el puntero para referenciar los datos direccionados.
- Para asignar una dirección de memoria a un puntero se utiliza el operador de referencia &.

```
&valor
```

- Significa la “dirección de valor”.

Inicialización de punteros

- Por consiguiente, el método de inicialización también denominado *estático* requiere:
  - Asignar memoria (estáticamente) definiendo una variable y a continuación hacer que el puntero apunte al valor de la variable.

```
int i;    //Define una variable i
int *p;   //Define un puntero a un entero p
p = &i;   //Asigna la dirección de i a p
```

  - Asignar un valor a la dirección de memoria.

```
*p = 50
```

- Cuando ya se ha definido un puntero, el asterisco delante de la variable puntero indica “el contenido de” de la memoria apuntada por el puntero y será del tipo dado.

Inicialización de punteros

- Otros ejemplos de inicialización estáticos:

```
int edad = 50;                  //Define una variable edad de valor 50

int *p_edad = &edad;            //Define un puntero de enteros inicializándolo
                                //con la dirección de edad
```

```
char *p;
char alfa = 'A';
p = &alfa;
```

```
char cd[] = “Compacto”;
char *c;
c = cd;                 //c tiene la dirección de la cadena cd
```

Indirección de punteros

- Después de definir una variable puntero, el siguiente paso es inicializar el puntero y utilizarlo para direccionar algún dato específico en memoria.
- El uso de un puntero para obtener el valor al que apunta, es decir, su dato apuntado se denomina indireccionar el puntero «desreferenciar el puntero»; para ello, se utiliza el **operador de indirección *** .

```
int edad;
int *p_edad;
p_edad = &edad;
*p_edad = 50;
```

- Si se desea imprimir el valor de edad, se puede utilizar la siguiente sentencia:

```
printf (“%d”, edad);                    //imprime el valor de edad
```

- También se puede imprimir el valor de edad desreferenciando el puntero a edad:

```
printf ( “%d”, *p_edad);                //indirecciona p_edad
```

Ejemplo

- El listado del siguiente programa muestra el concepto de creación, inicialización e indirección de una variable puntero.

```
#include <stdio.h>

char c;                         //variable global de tipo caracter
int main()
{
        char *pc;               //un puntero a una variable caracter
        pc = &c;
        for (c='A'; c<='Z' ; c++)
                printf("%c ", *pc);
        return 0;
}
```

- La ejecución de este programa visualiza el alfabeto.

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Ejemplo

```
#include <stdio.h>

char c;
int main()
{
        char *pc;
        pc = &c;
        for (c='A'; c<='Z' ; c++)
                printf("%c",*pc);
        return 0;
}
```

- La variable puntero **pc** es un puntero a una variable carácter.
- La línea **pc = &c** asigna a pc la dirección de la variable **c**.
- El bucle for almacena en **c** las letras del alfabeto y la sentencia printf("%c",*pc); visualiza el contenido de la variable apuntada por **pc**.
- **c** y **pc** se refieren a la misma posición en memoria.
- Si la variable **c**, que se almacena en cualquier parte de la memoria, y **pc**, que apunta a esa misma posición, se refiere a los mismos datos, de modo que el cambio de una variable debe afectar a la otra.

Punteros y verificación de tipo

- Los punteros se enlazan a tipos de datos específicos, de modo que C verificará si se asigna la dirección de un tipo de dato al tipo correcto de puntero.
- Así, por ejemplo, si se define un puntero a float, no se le puede asignar la dirección de un carácter o un entero.
- Por ejemplo, este segmento de código no funcionará:

```
float *fp;
char c;
fp = &c;     //no es válido
```

- C no permite la asignación de la dirección de **c** a **fp**, ya que **fp** es una variable puntero que apunta a datos de tipo real, *float*.
- C requiere que las variables puntero direccionen realmente variables del mismo tipo de dato que está ligado a los punteros en sus declaraciones.

Punteros null y void

- Normalmente un puntero inicializado adecuadamente apunta a alguna posición específica de la memoria.
- Sin embargo, un puntero no inicializado, como cualquier variable, tiene un valor aleatorio hasta que se inicializa el puntero.
- En consecuencia, será preciso asegurarse que las variables puntero utilicen direcciones de memoria válida.
- Existen dos tipos de punteros especiales muy utilizados en el tratamiento de sus programas: los punteros void y null (nulo).
- Un puntero nulo no apunta a ninguna parte -dato válido- en particular, es decir, «un puntero nulo no direcciona ningún dato válido en memoria».
- Un puntero nulo se utiliza para proporcionar a un programa un medio de conocer cuando una variable puntero no direcciona a un dato válido.

- Los punteros nulos se utilizan con frecuencia en programas con arrays de punteros.
- Cada posición del array se inicializa a NULL; después se reserva memoria dinámicamente y se asigna a la posición correspondiente del array, la dirección de la memoria.

```
#define NULL 0
char *p = NULL;
```

void

- En C se puede declarar un puntero de modo que apunte a cualquier tipo de dato, es decir, no se asigna a un tipo de dato específico.
- El método es declarar el puntero como un puntero void *, denominado puntero genérico.

```
void *ptr;    //Declara un puntero void o puntero genérico
```

- El puntero ptr puede direccionar cualquier posición en memoria, pero el puntero no está unido a un tipo de dato específico.
- De modo similar, los punteros void pueden direccionar una variable float, una char, o una posición arbitraria o una cadena.

Punteros a punteros

- Un puntero puede apuntar a otra variable puntero.
- Este concepto se utiliza con mucha frecuencia en programas complejos de C.
- Para declarar un puntero a un puntero se hace preceder a la variable con dos asteriscos (**).

- En el ejemplo siguiente ptr5 es un puntero a un puntero.

```
int valor = 100;
int *ptr1 = &valor;
int **ptr5 = &ptr1;
```

- ptr1 y ptr5 son punteros.
- ptr1 apunta a la variable valor de tipo int.
- ptr5 contiene la dirección de ptr1.

Punteros a punteros

- Se puede asignar valores a **valor** con cualquiera de las sentencias siguientes:

```
valor = 95;
*ptr1= 105;          //Asigna 105 a valor
**ptr5 = 99;         //Asigna 99 a valor
```

```
char c = 'z';
char * pc = &c;
char ** ppc = &pc;
char *** pppc = &ppc;

***pppc = 'm';          //Cambia el valor de c a 'm'
```

Operadores de dirección e indirección

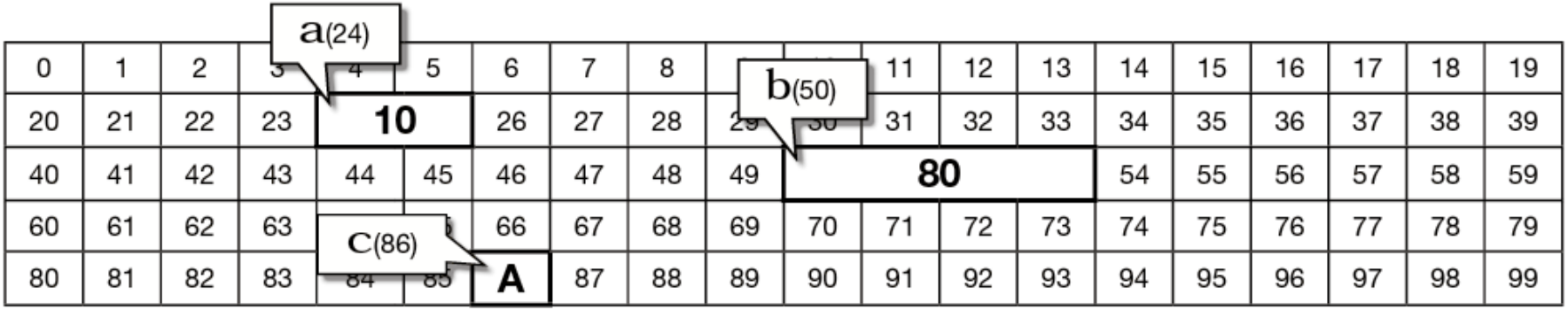
Puntero y direcciones de memoria

La memoria de la computadora se compone de un conjunto de *bytes* numerados consecutivamente. Luego, podemos almacenar datos en celdas de memoria formadas por grupos de 1 o más *bytes* según sea la longitud del dato que vayamos a almacenar.

Supongamos que en un programa definimos las siguientes variables:

```
int a  = 10;  // 2 bytes
long b = 80;  // 4 bytes
char c = 'A'; // 1 byte
```

La declaración de estas variables hará que el programa, al comenzar a ejecutarse, reserve una cierta cantidad de *bytes* de memoria que dependerá de la longitud de sus tipos de datos. De esta forma, la distribución de la memoria de la computadora podría llegar a ser la siguiente:



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23			26	27	28	29	30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49					54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85		87	88	89	90	91	92	93	94	95	96	97	98	99

El gráfico debe interpretarse de la siguiente manera: cada celda representa 1 *byte* de memoria cuya dirección está dada por un número secuencial.

Según el ejemplo, la dirección de la variable `a` es 24 y ocupa una celda compuesta por 2 *bytes*, la dirección de la variable `b` es 50 y utiliza 4 *bytes*. La variable `c` requiere un único *byte* y se ubica en la dirección de memoria número 86.

El operador de dirección &

Al anteponer el operador `&` (léase “ampersand”) al identificador de una variable obtenemos su dirección de memoria.

Siguiendo con el ejemplo anterior, el valor de la variable `a` es 10 y el valor de `&a` es su dirección de memoria que, en este caso, es 24. Veamos la siguiente tabla:

a =	10	&a =	24
b =	80	&b =	50
c =	‘A’	&c =	86

Contenido de las variables vs. sus direcciones de memoria.

En la tabla, vemos el contraste entre el contenido de una variable y su dirección de memoria, la cual podemos obtener a través del operador `&`.

Los punteros

Llamamos “puntero” a una variable capaz de contener una dirección de memoria.

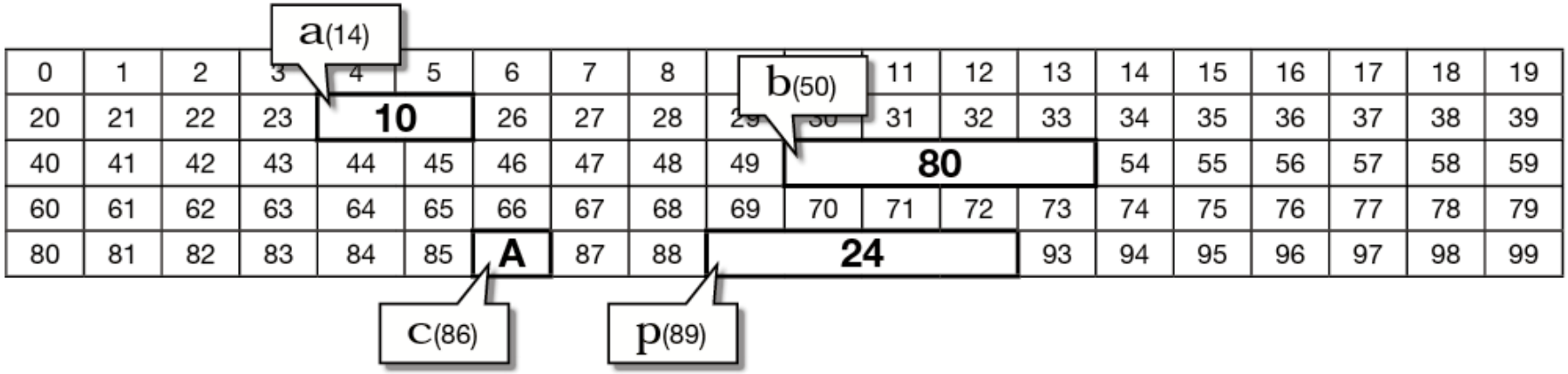
En las siguientes líneas de código, retomamos el ejemplo anterior y agregamos la variable `p` a la que le asignamos la dirección de memoria de la variable `a`. Luego diremos que “`p` es un puntero a `a`” o, simplemente, diremos que “`p` apunta a `a`”.

```
int a  = 10;  // 2 bytes
long b = 80;  // 4 bytes
char c = 'A'; // 1 byte

int* p = &a;
```

Como `a` es de tipo `int` entonces el tipo de datos de `p` es `int*` (léase “int asterisco” o “puntero a entero”). Los punteros o direcciones de memoria se representan en celdas de 4 *bytes*.

Ahora, considerando a la variable `p` la distribución de memoria podría ser la siguiente:



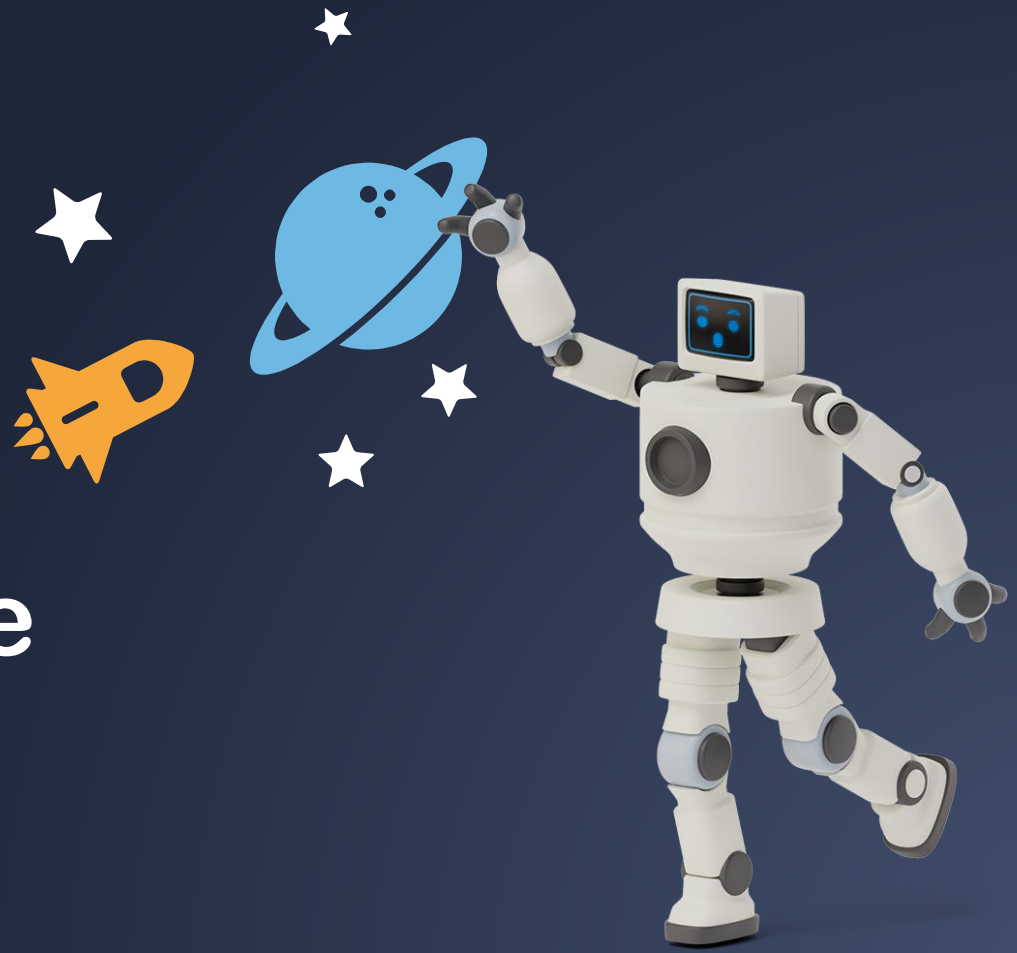
Representación de una variable de tipo puntero.

Luego, si queremos acceder al espacio de memoria direccionado por `p` tendremos que hacerlo a través del operador de indirección `*` (léase "asterisco") de la siguiente manera:

```
*p = 20;
```

En este caso, asignamos el valor 20 en el espacio de memoria direccionado por `p`. Claro que, como "`p` apunta a `a`", indirectamente, la asignación la estaremos haciendo sobre esta variable.

Aritmética de apuntadores



Aritmética de apuntadores

- Al contrario que un nombre de array, que es un puntero constante y no se puede modificar, un puntero es una variable que se puede modificar.
- Como consecuencia, se pueden realizar ciertas operaciones aritméticas sobre punteros.
- A un puntero se le puede sumar o restar un entero n; esto hace que apunte n posiciones adelante, o atrás de la actual.
- Una variable puntero puede modificarse para que contenga una dirección de memoria n posiciones adelante o atrás.

```
int v[10];
int *p;
p = v;
p = p+6;      //Contiene la dirección del 7° elemento
```

- A una variable puntero se le puede aplicar el operador ++, o el operador -- .
- Esto hace que contenga la dirección del siguiente, o anterior elemento.

```
float m[20];
float *r;
r = m;
r++;      //Contiene la dirección del elemento siguiente
```

- Recuérdese que un puntero es una dirección, por consiguiente, sólo aquellas operaciones de “sentido común” son legales.
- No tiene sentido, por ejemplo, sumar o restar una constante de coma flotante.

Ejemplo

- Si p apunta a la letra A en el alfabeto, si se escribe
- p = p + 1
- Entonces p apunta a la letra B

- Se puede utilizar esta técnica para explorar cada elemento de alfabeto sin utilizar una variable de índice.

```
p = &alfabeto[0];
for(i = 0; i<strlen(alfabeto); i++)
{
     printf("%c" , *p);
     p = p + 1;
}
```

- Las sentencias del interior del bucle se pueden sustituir por:

```
printf("%c" , *p++);
```

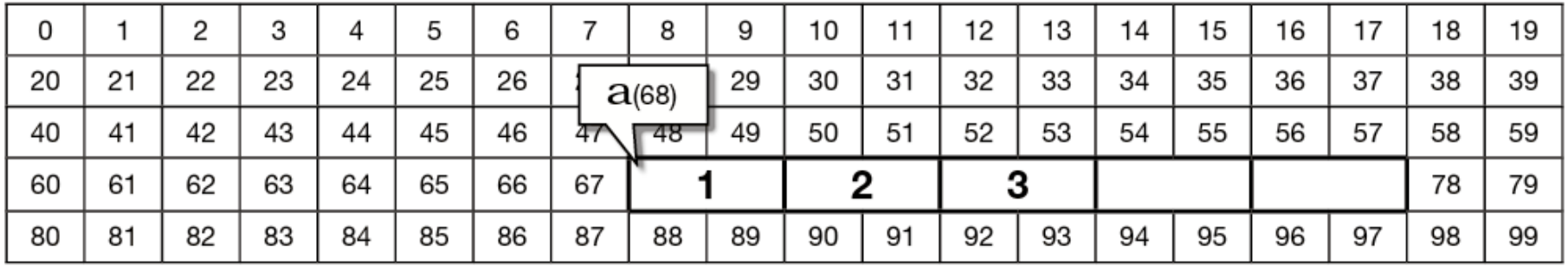
Relación entre arrays y punteros

Un *array* representa a un conjunto de celdas de memoria consecutivas y su identificador (la variable) es, en realidad, la dirección de memoria del primer elemento.

Es decir que, sea el *array* `a` definido como vemos a continuación:

```
int a[5] = {1, 2, 3};
```

podemos representarlo de la siguiente manera:



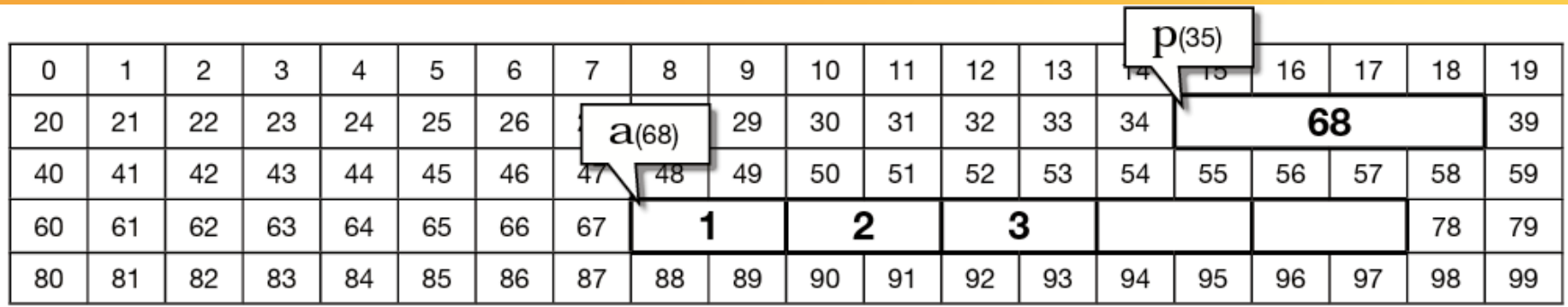
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26			29	30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	**1**		**2**		**3**						78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99

En el gráfico vemos que el *array* se aloja a partir de la dirección de memoria 68, en 5 celdas consecutivas de 2 *bytes* cada una. La variable `a` es la dirección de memoria de la primera de estas celdas.

Relación entre arrays y punteros

Resulta entonces que un `int[]` *matchea* (encaja) en un `int*` lo cual nos permite acceder al primer elemento del *array* a través de su dirección de memoria como vemos en las siguientes líneas de código:

```
int a[5];
int *p=a; // p apunta al primer elemento del array
*p = 10;  // equivale a hacer: a[0]=10
```



Inicialización de un array de punteros a cadena

- La inicialización de un array de punteros a cadenas se puede realizar con una declaración similar a ésta:

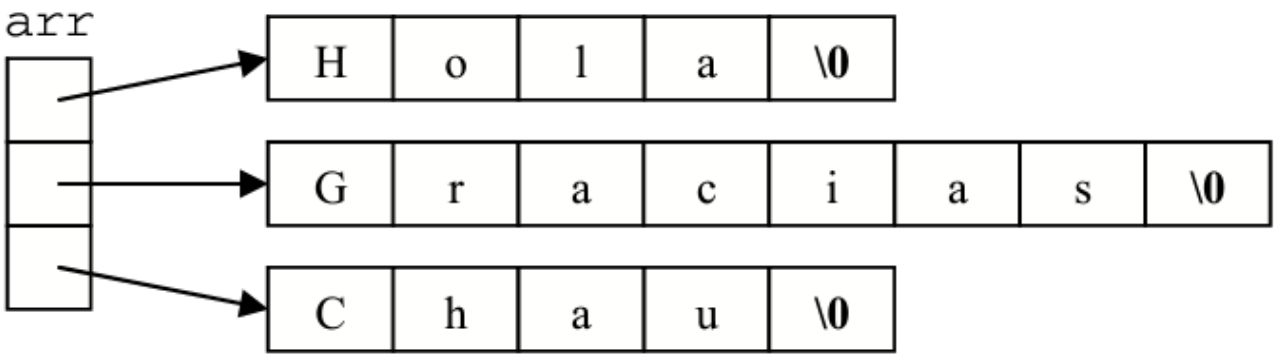
```
char *nombres_meses[12] = { "Enero", "Febrero", "Marzo",
                            "Abril' , "Mayo" , "Junio" ,
                            "Julio" , "Agosto" , "Septiembre" ,
                            "Octubre", "Noviembre", "Diciembre" };
```

Arrays de cadenas

Dado que una cadena es un `char*` entonces un *array* de cadenas debe implementarse como un *array* de punteros a carácter como se muestra en la siguiente línea de código:

```
char* arr[] = {"Hola", "Gracias", "Chau"};
```

Gráficamente, el *array* `arr` debería imaginarse así:



Array de cadenas de caracteres.