Apuntadores y Memoria dinámica

Apuntadores

Dra. Yolanda Moyao Martínez

Por que?

- Asignar memoria dinámica.
- Crean código eficiente y rápido (más cerca del hardware).
- Hacen expresiones compactas y concisas.
- Protegen datos pasados como parámetros a una función.
- Pasan estructuras de datos mediante un puntero sin ocasionar un exceso de código.

Uso

- Arreglos
- Estructuras
- Funciones
- Archivos

Definición

- Un apuntador es una variable que contiene la dirección en memoria de otra variable.
- La variable se refiere directamente a un valor y el puntero lo hace indirectamente.
- Se pueden tener apuntadores a cualquier tipo de variable.

Operadores

- El operador unario & devuelve la dirección de memoria de una variable.
- El operador de dirección o de referencia * devuelve el ``contenido de un objeto apuntado por un apuntador''.

Declaración

- tipo *nombre_apuntador;

  Donde,

  tipo: se refiere al tipo de la variable apuntada por este

  nombre_apuntador: se refiere al nombre de la variable de tipo apuntador

- Se debe asociar a cada apuntador un tipo particular.

Inicialización

1. Inicializarlo con la dirección de una variable que ya existe en memoria.

Por ejemplo:

```
char p1;
```

- `char *p = &p1;`

Inicio

2. Asignarle el contenido de otro apuntador que ya está inicializado:

Ejemplo:

```
char *p = &p1;
char *p3 = p; /* p ya está inicializado */
```

Inicio

3. Inicializarlo con cualquier expresión que devuelva un *lvalue*.

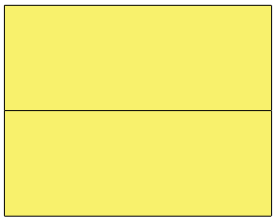
- Los *lvalue* más frecuentes son una cadena de caracteres, el identificador de un arreglo, el identificador de una función.

NOTA: si es global se inicia a NULL automáticamente

Gráficamente

int var;

int *apt;

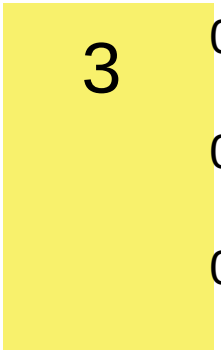


0x100

0x101

0x102

var=3;

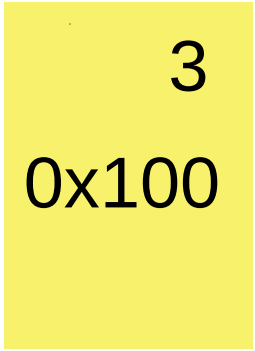


0x100

0x101

0x102

dirección



0x100

0x101

0x102

La dirección de var es =x100

El contenido de var es 3

apt = &var;

printf(“%d”, *apt)

Ejemplos

```
int *p,i;
p=0;            /*P apunta a NULL*/
```

- `p=NULL;  /* NULL equivale a 0*/`
- `p=&i;  /*la dirección de i se le asigna a p */`

NULL está definido en stdio.h

Mas ejemplos

double x, y, *p; /* dos variables de tipo double y una variable que es de tipo apuntador a un double (real) */

- p=&x; /* a p se le pasa la dirección de x */
- y=*p; /* a y se le pasa el contenido de p */
- y=*&x; /* son operadores unarios * y & se asocian de derecha a izquierda y dice el contenido de la dirección de x asignalo a y */
- &4 /* no se debe apuntar a constantes */
- int a[10]; /*se declara un arreglo */
- &a /* por si sola la variable a significa una dirección */

Aritmética

- int *p1, x;
- p1=&x;
- Si p1 tiene el valor 2000 (dirección)

Después de hacer p1++

p1 ahora será 2002

por que?

El incremento se adecua al tamaño en memoria del objeto apuntado

Ejemplos

y = *ip+10;

Añade 10 al objeto *ip (objeto referenciado por ip) y lo asigna a y

*ip += 1 igual *ip = *ip + 1 igual ++*ip igual (*ip)++

Incrementa en 1 el valor del objeto referenciado por ip.

Ejemplos

++ip igual ip++ igual ip = ip+1

Incrementa en 1 el valor del puntero, con lo que el nuevo puntero señala a otra posición

*ip++ igual *(ip++)

Accede a otro elemento

```
int x=5,y,*apt;
apt=&x;
y=*apt+x;   /*10*/
```

Práctica. Probar y corregir las líneas de código

1.
```
int *apun; /*float *apun*/
float valor= 10.5;
apun = &valor;

printf(“valor vale: %f”, *apun);
```

Ejercicio

2. int x = 20;

```
int *px;
px=&x;
printf("El valor de x es:%d", *px);
/*20*/
```

Ejercicio

3. int y=100;

   int *px;

   px=&y;

   printf(“El valor de y es %d”, *px);

Ejercicio

4. int p=20, *px;

   px=&p;

   printf(“el valor de p es %d”, *px);

Códigos de formato

%s espera un apuntador

%p espera un apuntador

%x entero hexadecimal

5. ¿Cual es la salida?

```
#include<stdio.h>
#include<conio.h>
main()
{
 char u,v='A';   /*la dirección de v es 100*/
 char *pu,*pv=&v;
*pv=v+1;
 u=*pv+1;
 pu=&u;
 printf("el valor de v es %x\n",&(v+1));     /* dir 101 */
 printf("dirección apuntada es %p \n",pv);  /*  dir 1200 */
 printf("el valor representado por pv es %c",*pv);  /*   B  */
 printf("el valor de u es %c\n",u);  /* C */
 printf("dirección apuntada es %p\n",pu);  /* dir 1500  */
 printf("el valor representado por pu es %c\n",*pu); /* c */
 printf("el código ascii de B es %d",'B');  /* asccii 66 */
}
```

Apuntador a arreglo

- Una variable de tipo arreglo puede considerarse como un apuntador al tipo del arreglo.
- Los apuntadores pueden ser utilizados en cualquier operación que involucre subíndices de arreglos.

Apuntador a arreglo

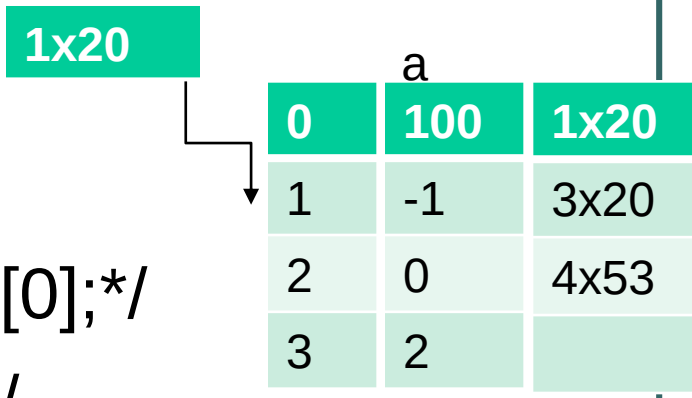
- A una variable de tipo arreglo no se le puede asignar una variable de tipo apuntador, debido a que el identificador del arreglo no es una variable.

Por ejemplo,

```
int *ap, a[20];
    a=ap;   /*no es válido */
    ap=a;   /*si esa válido*/
```

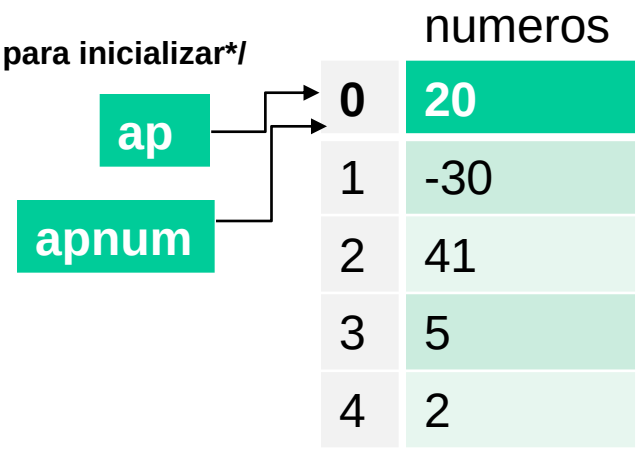
Acceso con el apuntador

```
int a[10], *ap, x;


*(ap+i)              /* equivale  a[i]*/
ap=a;                /* equivale ap=&a[0];*/
x=*ap;               /* equivale x=a[0]*/
*(ap+1)=100;         /* equivale a[1] = 100*/
ap = ap +9;          /* apunta al décimo elemento*/
```



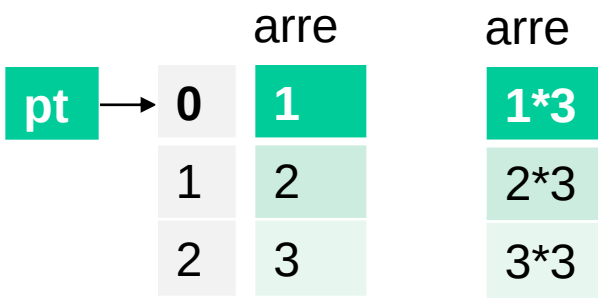
Ejemplo

```
#include <stdlib.h>
#include <stdio.h>

int main()
    {
      int numeros[5], i, *ap, *apnum, sum=0;
      ap = &numeros[0];
      srand(time(NULL));   /*la variable semilla para inicializar*/
      for (i=0; i <5; i++)
          *(ap+i) = rand() % 50-25;
      apnum = numeros;
      for (i=0; i <5; i++)
          {
            sum=sum+*(apnum+i);
            printf("%d \n",*(apnum+i),numeros[i]);
          }
       printf("La suma de los números es  %d \n",sum);
     }
```



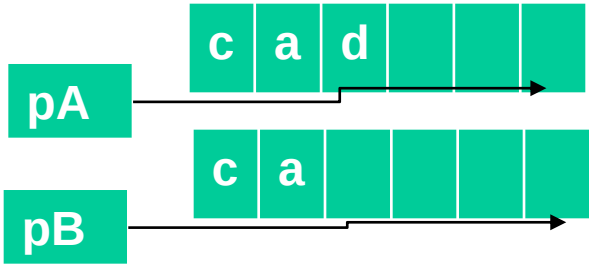
Ejemplo

```
#include<stdio.h>
#include<conio.h>
int main()
{      /* imprime el vector usando un puntero */
int arre[3]={1,2,3},i,*pt;
pt=arre;
for (i=0; i<3; i++)
  {
    *(pt+i)=*(pt+i)*3;
    printf("%d",*(pt+i));
  }
getch();
}
```



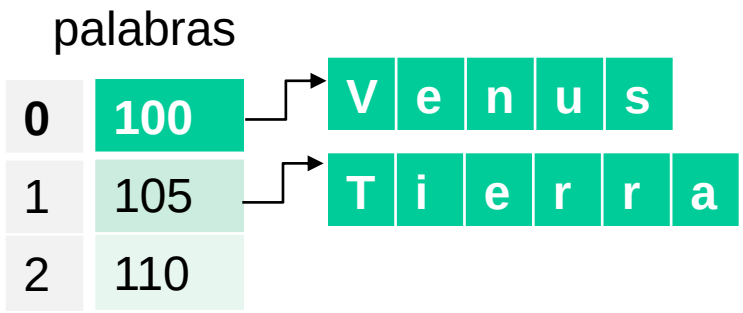
Práctica. ¿Cuál es la salida?

```
#include <stdio.h>
char strA[80] = "Cadena a usar para el programa de ejemplo";
char strB[80];
int main(void)
   {
     char *pA, *pB;

     puts(strA);
     pA = strA;
     puts(pA);
     pB = strB;
     putchar('\n');
     while((*pA != '\0') && (*pA != '.'))
        *pB++ = *pA++;
      *pB = '\0';
     pA = strA;       /*inicializar pues pA  cambió de dirección*/
     pB = strB;       /*inicializar pues pA  cambió de dirección*/
     puts(pA);
     puts(pB);
     return 0;
    }
```



Arreglo de apuntadores

- Los arreglos pueden contener apuntadores.
- Cada entrada en el arreglo es un apuntador al primer caracter de la cadena.



char *palabras[3] = {“Venus”,”Tierra”,”Marte”}

Arreglo de apuntadores

- En el arreglo solo están almacenados los apuntadores.
- Aunque el arreglo es de tamaño fijo, permite el acceso a cadenas de cualquier longitud.

```
char *palabras[5] =
{“oso”,”tigre”,”elefante”,”jirafa”,”rinoceronte”}
```

Ejemplo

```
#include <stdlib.h>
#include <stdio.h>
int main()
  {
   int x;
                        /*char profesiones[5][40]={{Arquitecto},{}};
                         scanf("%s",arr[0]);   */
   char *profesiones[ ] = {
      "Arquitecto",
      "Programador",
      "Doctor",
      "Abogado",
      "Geriatra",
      "Endocrinólogo",
      "Neumólogo",
      "Cardiólogo"
   };
   srand(time(NULL));
   x = rand() % 8;
   printf("Eres un %s \n",profesiones[x]);
}
```

Memoria dinámica

Introducción

Por qué?

- Manipular estructuras de datos de longitud desconocida.
- Programa que lee las líneas de un archivo y las ordena.
- Una manera de manejar ese “número indeterminado", sería declarar un arreglo de tamaño vergonzosamente grande.
- Es ineficiente.
- Limitado por ese valor máximo o gastaría esa enorme cantidad de memoria para procesar hasta el más pequeño de los ficheros.

Ventajas

- Los programas funcionan de manera más eficiente.
- Aprovechamiento óptimo de los recursos de almacenamiento en RAM.

Qué es?

- *Datos dinámicos*: su tamaño y forma es variable a lo largo de un programa, por lo que se crean y destruyen en tiempo de ejecución.
- Esto permite dimensionar la estructura de datos de una forma precisa: se va *asignando* memoria (heap) en tiempo de ejecución según se va necesitando.

Manipular la memoria de forma dinámica

- Por medio de punteros se puede reservar o liberar memoria dinámicamente.

- Existen varias funciones estándares, de la biblioteca <stdlib.h>

La función malloc

- Sirve para solicitar un bloque de memoria del tamaño suministrado como parámetro.
- Devuelve un puntero a la zona de memoria concedida:
- **tipo *malloc(size_t size);** /*Reservamos un size de bytes.*/

Nota: Si malloc es incapaz de conceder el bloque (p.ej. no hay memoria suficiente), devuelve un puntero nulo.

Operador sizeof

- El tamaño en bytes de un elemento de tipo *T* se obtiene con la expresión

sizeof (*T*)

Ejemplo

- int *p;
  p = (int*) malloc(sizeof(int));
  *p = 5;

Función free

- Cuando una zona de memoria reservada con malloc ya no se necesita, puede ser *liberada* mediante la función free.

void free(void *ptr);

Nota: ptr es un puntero de cualquier tipo que apunta a un área de memoria reservada previamente con malloc.

Nota: Si ptr apunta a una zona de memoria indebida, los efectos pueden ser desastrosos y también si se libera dos veces la misma zona.

Ejemplo

```
#include <stdio.h>
#include <stdlib.h>
main()
 {
   int* ptr; /* puntero a enteros */
   int *ptr2,i; /* otro puntero */
                /* reserva espacio para 4 enteros */
   ptr = (int*)malloc ( 4*sizeof(int) );
   for(i=0;i<4;i++){
      ptr[i] = 15+i; /* trabaja con el área de memoria*/
      ptr2 = ptr; /* asignación a otro puntero */
   }
    for(i=0;i<4;i++)
     printf("%d\n",*ptr2++);
        /* finalmente, libera la zona de memoria */
    free(ptr);
}
```

Ejemplo

```
#include<stdio.h>
#include<stdlib.h>
main()
{
    int *p,i;
    p = (int*) malloc(5*sizeof(int));
    for(i=0;i<5;i++){
       scanf("%d",&(*p));
       printf("%d\n",*p);
       p++;
      }
    free(p);
}
```

Ejemplo

Si queremos copiar el string b en a, que es sólo un puntero y no tiene área asignada para copiar los caracteres, podemos hacer:

```
char *a;
char b[ ] = "hola";
if ((a = (char*)malloc(strlen(b)+1)) == NULL) {
    printf("no hay memoria suficiente\n");
    exit(1);
 }
strcpy(a,b);
free((void*)a);
```

Ejemplo

```
int *arr,n,i;  /* int* arr*/
scanf("%d",&n);
if ((arr = (int*)malloc(sizeof(int) * n)) == NULL) {
   printf("no hay memoria suficiente\n");
   exit(1);
}
for(i = 0;i < n;i++)
   arr[i] =10;
free((void*)arr);
```

10
10
10

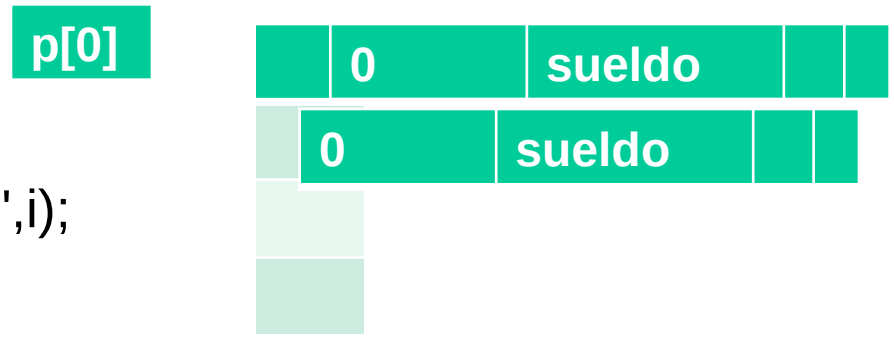
10
10
10
10
10

Ejemplo con estructuras

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

struct emp
{
  int codigo;
  float sueldo;
  char  nombre[40];
};
```

Ejemplo

```
main()
{
int cuantos,i;
struct emp *p;
printf("¿Cuántos empleados? ");
scanf("%d",&cuantos);
if ((p = (struct emp*)malloc(cuantos*sizeof(struct emp)))== NULL) {
    printf("no hay memoria suficiente\n");
    exit(1);
 }
for(i = 0;i < cuantos;i++) {
   p[i].codigo = i;
   p[i].sueldo = 0.0+i;
   printf("Dame el nombre %d\n",i);
   scanf("%s",p[i].nombre);
  }
```



```
for(i = 0;i < cuantos;i++)
  {printf("Nombre  %s\n",p->nombre);
   printf("Código  %d\n",(*p).codigo);
   printf("Sueldo  %.2f\n",p->sueldo);
   p++;
   printf("\n");
  }
getchar();
free((void*)p);
}
```

Ejercicio: Explica la salida

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int i;
 char *s;
 int j;
 for(i = 2;i < 6;i++) {
     if ((s = (char*)malloc(i+1)) == NULL) {
        printf("no hay memoria suficiente\n");
        exit(1);
     }
    for(j = 0;j < i;j++) s[j] = 'a';
    s[j] = '\0';
    printf("%s\n",s);
    free((void*)s);
 }
 return 0;
}
```

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
main()
{
        int filas = 2;
        int columnas = 3;
        int **x;
        int i, j;
        /* Reserva de Memoria para una matriz*/
        x = (int **)malloc(filas*sizeof(int*));
        for (i=0;i<filas;i++)
            x[i] = (int*)malloc(columnas*sizeof(int));
        x[0][0] = 1;
        x[0][1] = 2;
        x[0][2] = 3;
        x[1][0] = 4;
        x[1][1] = 5;
        x[1][2] = 6;
        for (i=0; i<filas; i++)
           {
             printf("\n");
             for (j=0; j<columnas; j++)
                  printf("\t%d", x[i][j] );
           }
        free(x);
         return 0;
}
```

Ejercicios

- Investiga la sintaxis de la función realloc e implementa un programa que use dicha función.
- Investiga la sintaxis de la función calloc e implementa un programa que use dicha función.

Precauciones

- Si una zona de memoria reservada con malloc se pierde, no se puede recuperar ni se libera automáticamente (no se hace *recolección de basura*).

- Si una zona de memoria liberada con free estaba siendo apuntada por otros punteros, esos otros punteros apuntarán a una zona ahora incorrecta.

Punteros mal apuntados

1. Pedimos memoria.
2. Algo va mal:
   - Confundimos el tipo al pedir espacio.
   - Confundimos el cast.
   - malloc() falla pero no lo comprobamos.

Ejemplo

```
int * p;
p = malloc( 8 * sizeof(char) );
```

3. Violación de segmento (SEGFAULT).

Goteos de memoria

1. Pedimos memoria.
2. Perdemos memoria:
   - Nos olvidamos de liberar la memoria.
   - Perdemos la referencia a ese bloque de memoria.

Ejemplo

```
int * p;
p = (int *) malloc(5 * sizeof(int));
p = (int *) malloc(4 * sizeof(int));
```

3. Somos malos programadores

Liberar demasiado

Cuestiones de portabilidad:

Liberar dos veces:

Ejemplo

```
p = q;
free(p);
free(q);
```

Liberar un NULL:

Ejemplo

```
p = NULL;
free(p);
```

Liberar demasiado (2)

Referenciar un espacio de memoria liberado:

Ejemplo

```
int *a, *b;
int c;
a = (int *) malloc(400);
b = a;
free(a);
c = b[1];
```