Unitat 8 – Punters i memòria dinàmica

- OBJECTIU GENERAL:

  Conèixer el funcionament dels punters i la reserva dinàmica de memòria i saber utilitzar aquestes eines en els programes desenvolupats en C.

Professor: Ximo Tur

IES Sa Colomina

Punters

- El valor de cada variable està emmagatzemat en un lloc determinat de la memòria, caracteritzat per una ***direcció*** (que se sol expressar amb un número hexadecimal). L'ordinador manté una ***taula de direccions*** que relaciona el nom de cada variable amb la seua direcció en la memòria.

Tabla 6.1. Tabla de direcciones.

Variable	Dirección de memoria
a	00FA:0000
b	00FA:0002
c	00FA:0004
p1	00FA:0006
p2	00FA:000A
p	00FA:000E

- Gràcies als noms de les variables (identificadors) no cal que el programador es preocupe de la direcció de memòria on estan emmagatzemats les seues dades. No obstant això en certes ocasions és més útil treballar amb les direccions que amb els propis noms de les variables. El llenguatge C disposa de ***l'operador direcció*** (&) que permet determinar la direcció d'una variable, i d'un tipus especial de variables destinades a contindre direccions de variables. Estes variables són els punters.

Punters

Un **punter** és una variable que conté com a valor una direcció de memòria. Aquesta direcció correspon habitualment a la direcció que ocupa una altra variable en memòria. Es diu llavors que el punter *apunta a aquesta variable*. La variable apuntada pot ser de qualsevol tipus elemental, estructurat o inclús un altre punter.

Alguns dels avantatges que aporta l'ús de punters en C són:

- Constituïxen l'única forma d'expressar algunes operacions.
- El seu ús produïx codi compacte i eficient.
- Són imprescindibles per al pas de paràmetres per referència a funcions.
- Tenen una forta relació amb el maneig eficient de taules i estructures.
- Permeten realitzar operacions d'assignació dinàmica de memòria i manipular estructures de dades dinàmiques.

En la declaració de punters i la posterior assignació de direccions de memòria als mateixos, s'utilitzen respectivament els operadors unarios * i &.

L'operador **&** permet obtindre la *direcció que ocupa una variable en memòria*.

L'operador d'indirecció ***** permet obtindre el *contingut d'un objecte apuntat per un punter*.

Punters

DECLARACIÓ DE PUNTERS

- En la declaració de variables punter s'usa també l'operador *, que s'aplica directament a la variable a la qual precedix. El format per a la declaració de variables punter és el següent:

```
tipus_de_dades * nom_variable_punter;
```

- Es diu que un *punter apunta a una variable* si el seu contingut és la direcció d'eixa variable. Un *punter* ocupa d'ordinari 4 bytes de memòria, i *s'ha de declarar o definir d'acord amb el tipus de la dada* a què apunta. Per exemple, un *punter* a una variable de tipus ***int (é***s a dir contindrà direccions de memòria on s'emmagatzemaran valors sencers*) es declara* de la manera següent:

```
int *N;
```

	M				187	188	189	190	191	192	193	194		
	189													

M apunta a un dada char

	N				187	188	189	190	191	192	193	194		
	189													

N apunta a un dada int

	P				187	188	189	190	191	192	193	194		
	189													

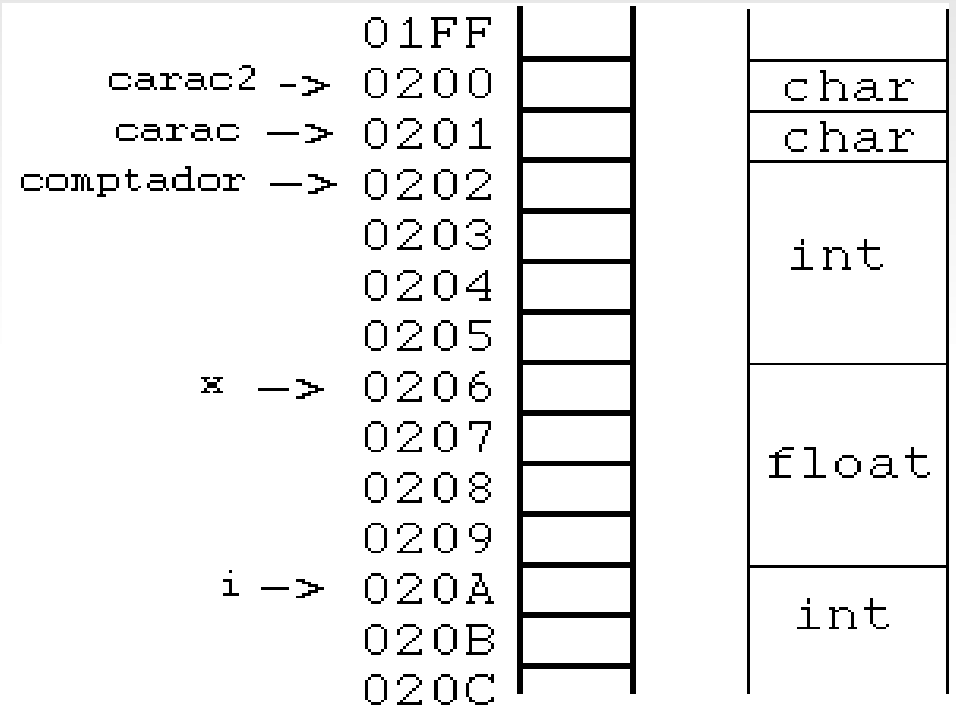
P apunta a un dada float

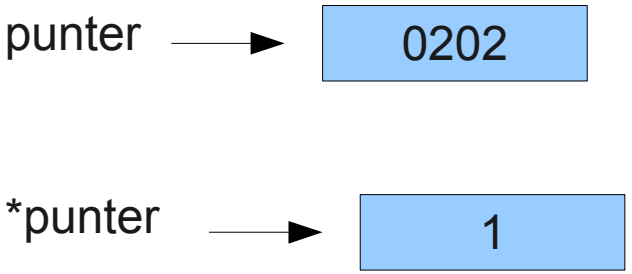
Punters

Un punter és una variable que conté una adreça de memòria (normalment d'una altra variable). Es diu que la variable punter o el punter apunta a aquesta segona variable.

```
int comptador = 1;
int *punter;

punter=&comptador;
```





Els operadors de direcció (&) i indirecció (*)

Els operadors de manipulació de punters: & i *

- L'operador & (operador d'adreça) és un operador unari que retorna l'adreça de memòria del seu operant. El seu operant pot ser qualsevol tipus de variable, incloent-hi les variables punters. A l'exemple anterior, l'assignació:

```
punter=&comptador;
```

- Fa que a la variable punter s'emmagatzemi l'adreça de la variable comptador.
- L'operador * (operador d'indirecció) és un altre operador unari que retorna el valor de la variable on està apuntant l'operant (que serà un punter). Per exemple, la sentència:
- num = *punter;
- assignarà a la variable num (declarada prèviament com a int) el valor 1.
- **L'operador & actua sobre qualsevol variable. L'operador * actua sobre variables punters.**
- No s'ha de confondre l'operador unari * amb l'operador binari * que representa el producte de dos nombres. En expressions complicades en les quals pugui haver confusions, es pot posar parèntesis per evitar aquestes confusions.

Els operadors de direcció (&) i indirecció (*)

- Com s'ha dit, el llenguatge C disposa de ***l'operador direcció*** (&) que permet trobar la direcció de la variable a què s'aplica. Un *punter és* una verdadera variable, i per tant pot canviar de valor, és a dir, pot canviar la variable a què apunta. Per a accedir al valor depositat en la zona de memòria a què apunta un *punter* s'ha d'utilitzar l'*operador* ***indirecció*** (*). Per exemple, per a les següents declaracions i sentències,

```
int i, j, *p;          // p es un puntero a int
p = &i;                // p contiene la dirección de i
*p = 10;               // i toma el valor 10
p = &j;                // p contiene ahora la dirección de j
*p = -2;               // j toma el valor -2
```

- Les constants i les expressions no tenen direcció, per la qual cosa no se'ls pot aplicar l'operador (&). Tampoc pot canviar-se la direcció d'una variable.punter pot tindre valor 0 (equivalent a la constant simbòlica predefinida NULL). No es pot assignar una direcció absoluta directament (caldria fer un casting). Aquestes sentències són il·legals:

```
p = &34;               // las constantes no tienen dirección
p = &(i+1);            // las expresiones no tienen dirección
&i = p;                // las direcciones no se pueden cambiar
p = 17654;             // habría que escribir p = (int *)17654;
```

Els operadors de direcció (&) i indirecció (*)

- Si el tipus de dades és **void**, es definix un punter genèric de manera que el seu tipus de dades implícites serà el de la variable la direcció del qual se li assigne. Per exemple, en el següent codi, ip és un punter genèric que al llarg del programa apunta a objectes de tipus distints, primer a un sencer i posteriorment a un caràcter.

```
void *ip;
int x;
char car;
.  .  .
ip = &x;    /* ip apunta a un entero */
ip = &car;  /* ip apunta a un carácter */
```

```
int    *p;
double *q;
void   *r;
p = q;              // ilegal
p = (int *)q;       // legal
p = r = q;          // legal
```

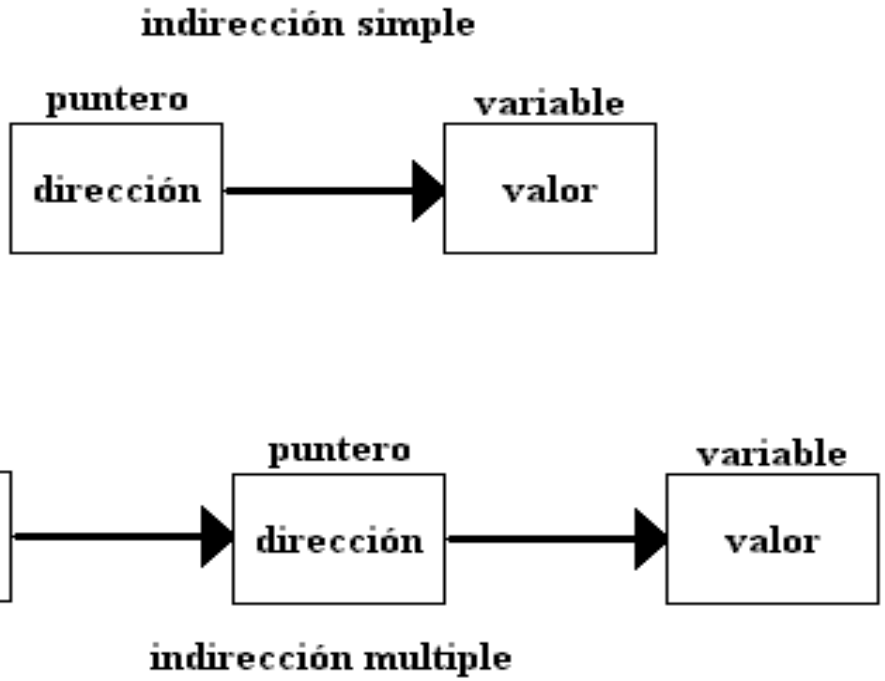
Els operadors de direcció (&) i indirecció (*)

- Existix també la indirecció múltiple, que un punter conté la direcció d'un altre punter que al seu torn apunta a una variable. El format de la declaració és el següent:

*tipus **nom_punter;*

- En el següent exemple, *pnum* apunta a *num*, mentres que *ppnum* apunta a *pnum*. Així que la sentència ***ppnum* = 24; assigna 24 a la variable *num*.

```
int num;
int *pnum;
int **ppnum;
.  .  .
pnum = &num;
ppnum = &pnum;
**ppnum = 24;
printf( "%d", num )
```



Operacions amb punters

Assignació de punters

- És possible assignar un punter a un altre punter, sempre que ambdós apunten a un mateix tipus de dades. Després d'una assignació de punters, ambdos apunten a la mateixa variable, perquè contenen la mateixa direcció de memòria

```
int x, y, z;
int *p1, *p2;
.  .  .
x = 4;
p1 = &x;
p2 = p1;
y = *p1;
z = *p2;
```

Comparació de punters

- Normalment s'utilitza la comparació de punters per a conèixer les posicions relatives que ocupen en memòria les variables apuntades pels punters.

```
int *p1, *p2, preu, quantitat;
p1 = &preu;
p2 = &quantitat;
```

- La comparació p1 > p2 permet saber quina de les dos variables (preu o quantitat) ocupa una posició de memòria major. És important no confondre està comparació amb la dels valors de les variables apuntades, *p1 > *p2 .

Operacions amb punters

Aritmètica de punters

- Si se suma o resta un nombre enter a un punter, el nombre de posicions de memòria incrementades o decrementades depén, no sols del número sumat o restat, sinó també de la grandària del tipus de dades apuntades pel punter. És a dir, una sentència com:

```
var_punter = var_punter + N;
```

- S'interpreta internament com:

```
var_punter = direcció_var_punter + N +mida_tipus_var_punter;
```

- Per exemple, tenint en compte que la grandària d'un float és de 4 bytes, i que la variable *num* se situa en la direcció de memòria 2088, quin és el valor de pnum al final del següent codi?

```
Float num, *punt, *pnum;
....
punt = &num;
pnum = punt +3;
```

- La resposta és 2100. És a dir 2088 + (3 x 4)
- NOTA: No es poden sumar, multiplicar o dividir 2 punters.

Arrays i punters

- En C els arrays i els punters estan molt relacionats. Un nom d'un array és simplement un punter:

```
int llista[5] = {10,20,30,40,50};
```

- Així *llista [0]* conté el valor 10
- Peró *llista* té l' adreça de memòria on es guarda aquest valor i que marcarà també l'inici del vector ja que tots els elements d'un vector estan guardats en memòria de forma consecutiva.
- Per tant *llista també tendrà el valor 10
- I:
  - Llista + 0 apunta a llista[0]
  - Llista + 1 apunta a llista[1]
  - Llista + 2 apunta a llista[2]
  - Llista + 3 apunta a llista[3]
  - Llista + 4 apunta a llista[4]
- *(llista + 4) té el valor 50

Arrays i punters

- Com hem dit el nom d'una array és un punter, però és una constant punter i no una variable punter. No es pot canviar el valor del nom d'un array:

```
int x=3, llista[5] = {10,20,30,40,50};
llista = &x;
```

- Llista és un punter constant, és a dir sempre a punta a la mateixa adreça de memòria:

```
char *const punter_constant;
```

- Per contra amb un punter si que ho podem fer:

```
int x, *ptr, llista[5] = {10,20,30,40,50};
ptr = llista;
printf("%d",ptr+2);  //mostra l'adreça de memòria de llista +8
                     //(adreça de llista + (2xsizeof(int))
ptr=&x
printf("%d",ptr);  //mostra l'adreça de memòria de x!!
```

- Aquesta flexibilitat fa que alguns programadors declaren vectors per la seva facilitat de declaració però per manipular-los utilitzen punters per la seva flexibilitat.

Arrays i punters

- Que els arrays són constants punters també explica que no podem copiar 2 cadenes fàcilment i que necessitem *strcpy* o *strcmp* per comparar:

```
char cad1="Hola", cad2 ="Adeu";
cad1=cad2;   //incorrecte
if(cad == cad2)  //incorrecte
```

- Amb punters si ho podríem fer:

```
char cad1="Hola", cad2 ="Adeu", char *ptr_cad1, char *ptr_cad2;
ptr_cad1 = cad1;
ptr_cad2 = cad2;
ptr_cad1 = ptr_cad2;   //correcte
if(ptr_cad1 == ptr_cad2) //correcte
```

Altres usos dels punters

- **Pas de paràmetres per referència**
- Ja hem vist que en C els punters s'utilitzen per a passar paràmetres a funcions per referència i poder modificar els valors dins de la funció (si els passem per valor la seva modificació no té efecte fora de la funció).
- Així la capçalera d'una funció que utilitza paràmetres per referència indicarà que aquests paràmetres són punters:

```
void funcio( int *operador1, int *operador2){
   *operador1 = 4;
   *operador2 = 5;
}
```

- I per cridar aquesta funció haurem de passar no la variable sinó la direcció de memòria on es troben les variables:

```
main(){
   int a=0;
   int b=2;
   printf("a: %d i b: %d",a,b);
   funcio(&a,&b);
   printf("a: %d i b: %d",a,b);
}
```

Altres usos dels punters

- **Punters a estructures**
- També hem vist que els punters poden apuntar a estructures i que quan ho fan hem d'utilitzar l'operador “->”per accedir als camps de l'estructura.

```
struct alumne alumne1;
struct alumne *ptr_alumne;

ptr_alumne = &alumne1;

ptr_alumne->no_matricula = 12321;
strcpy(ptr_alumne->dadesAl.nom, “Lluis”);
```

Altres usos dels punters

ARRAY DE PUNTERS

- Els punters poden estructurar-se en arrays com qualsevol altre tipus de dades. Cal indicar el tipus i el nombre d'elements. La seua utilització posterior és igual que els arrays que hem vist anteriorment, amb la diferència que s'assignen direccions de memòria.

```
Tipus *nom[núm. Elements];
```

- La seva assignació és realitza:

```
nom_array[índex]=&variable;
```

- Per utilitzar els elements:

```
*nom_array[índex];
```

- Un ús molt comú dels arrays de punters és als arguments de la funció main:

```
void main(int argc, char *argv[ ]) // o void main(int argc, char **argv)
```

- Per a `./paraules cafe chocolate`

```
argc = 3
argv[ 0 ] = "paraules"
argv[ 1 ] = "cafe"
argv[ 2 ] = "chocolate"
```

Altres usos dels punters

Punters a funcions

- De mode semblant a com el nom d'un array en C és un punter, també el nom d'una funció és un punter. Aquesta funció pot ser interessant per què permet passar com a argument a una funció el nom d'una altra funció. Per exemple, si **pfunc és** un punter a una funció que torna un sencer i té dos arguments que són punters, aquesta funció pot declarar-se de la manera següent:

```
int (*pfunc)(void *, void *);
```

  - El primer parèntesi és necessari, la declaració següent:

```
int *pfunc (void *, void *); // incorrecte
```

- **Correspon a una funció anomenada *pfunc* que torna un punter a sencer**. Considere's el següent exemple per a cridar d'una manera alternatiu a les funcions ***sin()*** i ***cos(x)***:

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

void main(void){
     double (*pf)(double);

     *pf = sin;
     printf("%lf\n", (*pf)(3.141592654/2));
     *pf = cos;
     printf("%lf\n", (*pf)(3.141592654/2));
}
```

- Observe's com la funció definida per mitjà del punter reb el mateixos paràmetresi torna el mateix tipus que les funcions sinus i cosinus. L'avantatge està en que per mitjà del punter ***pf*** les funcions sinus i cosinus podrien ser passades indistintament com a argument a una altra funció. Aquesta útilitat resulta especialment important en grans projectes amb multituds de funcions.