LLENGUATGES DE PROGRAMACIÓ

BLOC 2 – SEMINARI 2

APUNTADORS II

MEMÒRIA DINÀMICA

APUNTADORS II

Apuntadors i arrays

La **relació entre apuntadors i arrays és molt estreta** → Podem utilitzar apuntadors per recórrer fàcilment tots els elements d'un array.

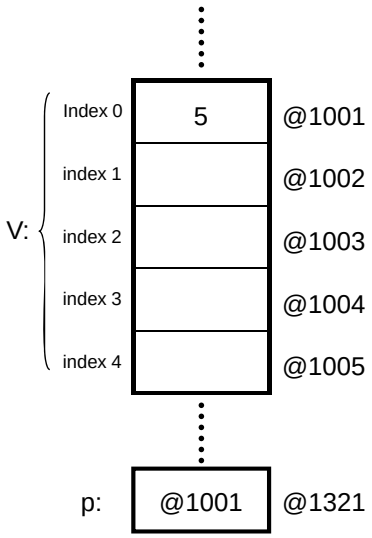
EXEMPLE

```
int V[5],*p;
```

```
p=&V[0];
*p=5;
```

De forma equivalent i més compacta, podem escriure:

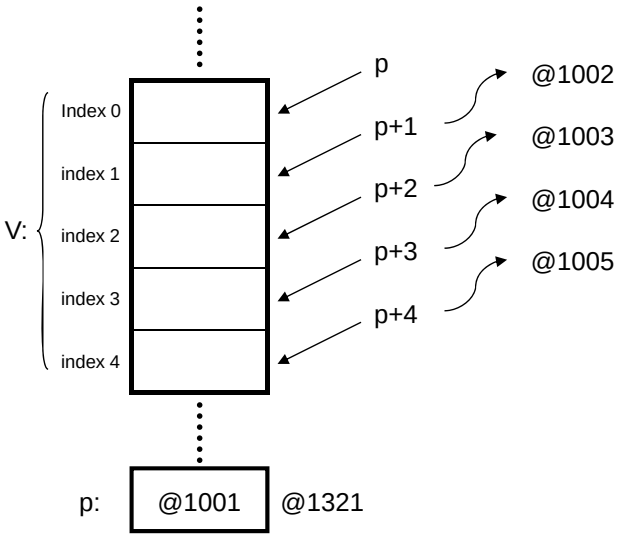
```
p=V;
```



Apuntadors i arrays

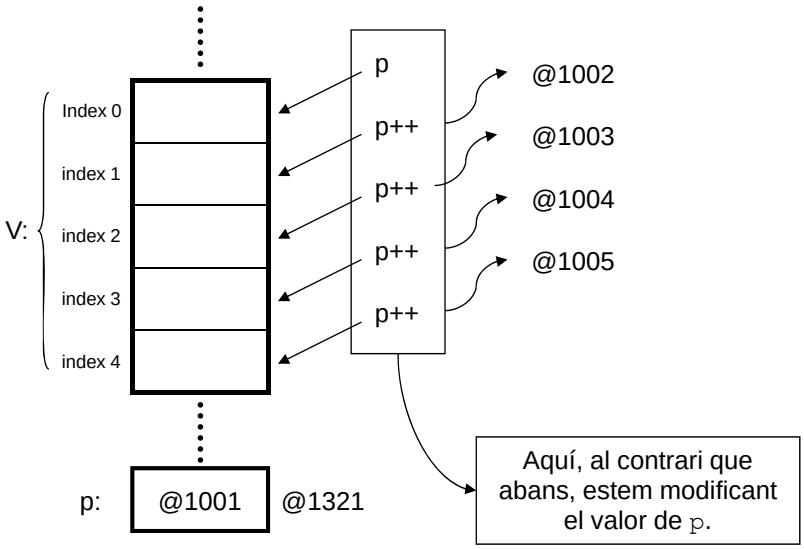
Una vegada tenim un apuntador al primer element, podem accedir a tots els elements de l'array. Ho podem fer de 3 formes diferents:

Operadors aritmètics → Podem sumar o restar a l'apuntador un determinat valor numèric (*p*+*n*), i això farà que l'apuntador avanci o retrocedeixi *n* posicions dins de l'array.



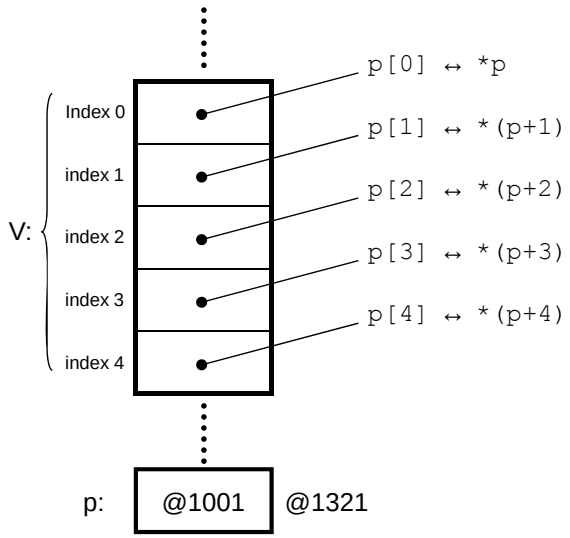
Apuntadors i arrays

Operadors d'autoincrement i autodedrecrement → Podem utilitzar amb l'apuntador els operadors d'autoincrement (*p*++) i autodecrement (*p*--). Això fa que l'apuntador passi a apuntar a l'element següent o anterior dins de l'array.



Apuntadors i arrays

Operador d'indexació [ ] → Podem utilitzar l'apuntador *p* com si fos un array i utilitzar l'expressió *p*[*n*]. D'aquesta manera estem accedint a la posició *n*, dins de l'array. És equivalent a utilitzar l'expressió *(*p*+*n*).



Apuntadors i arrays

PAS PER REFERÈNCIA D'ARRAYS UNIDIMENSIONALS

Crida a una funció amb un array com a paràmetre → No es passa una còpia de tot l'array, sinó simplement l'adreça del primer element de l'array.

Per tant, dins de la funció → L'array es tracta com un apuntador utilitzant els operadors.
Els canvis dins de la funció es reflecteixen a l'array original.

```
void LlegirVector (int v[], int n_elements)
```

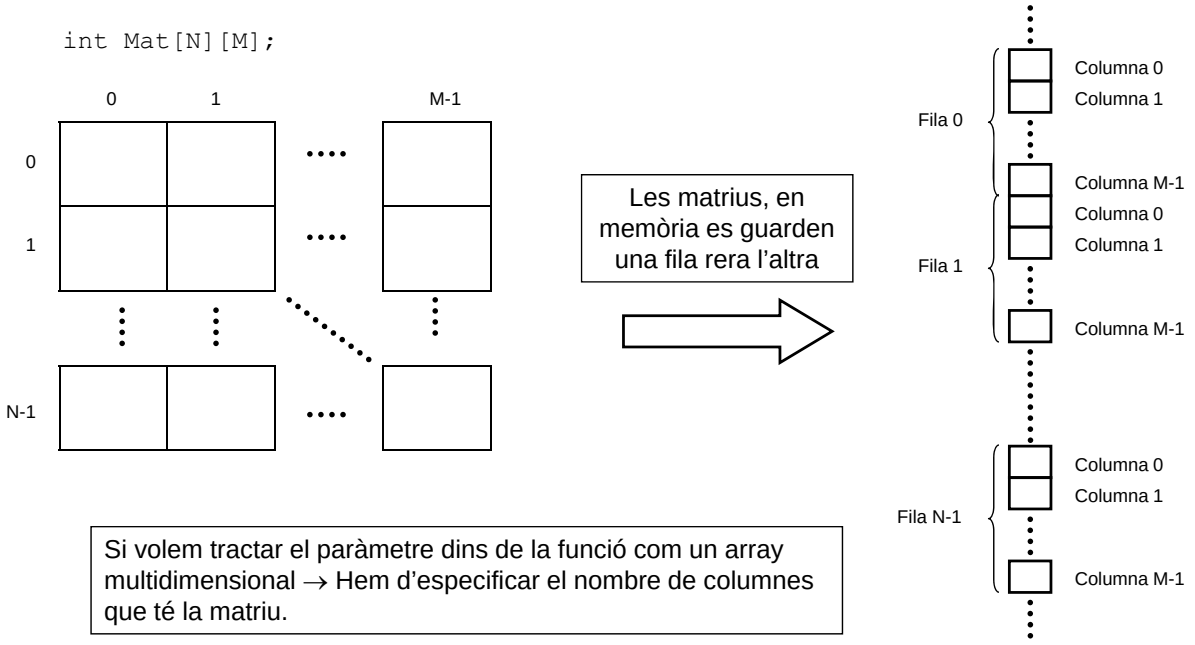
```
/* Programa que llegeix i suma
dos vectors */
#include <stdio.h>
#define MAX 5


void main()
{
  int v1[MAX], v2[MAX];
  int suma[MAX];
  int i;

  LlegirVector(v1,MAX);
  LlegirVector(v2,MAX);
  SumaVectors(v1,v2,suma,MAX);
  printf ("Suma dels vectors: ");
  for (i=0; i<MAX; i++)
    printf ("%d ", suma[i]);
}
```

```
void LlegirVector (int *v, int n_elements)
{
  int i;
  for (i=0; i<n_elements; i++)
  {
    printf ("Introdueix valor %d del vector: ", i);
    scanf ("%d", &v[i]);
  }

}

void SumaVectors (int *v1,int *v2,int *suma,int
n_elements)
{
  int i;
  for (i=0; i<n_elements; i++)
    suma[i] = v1[i] + v2[i];
}
```

Apuntadors i arrays

PAS PER REFERÈNCIA D'ARRAYS MULTIDIMENSIONALS

Crida a una funció amb un array multidimensional com a paràmetre → Passem l'adreça del primer element de l'array (l'element de la primera fila i primera columna).



Si volem tractar el paràmetre dins de la funció com un array multidimensional → Hem d'especificar el nombre de columnes que té la matriu.

Apuntadors i arrays

EXEMPLE

```
/* Programa que llegeix i suma dues
matrius */
#include <stdio.h>

#define N_FILES 2
#define N_COLUMNES 3

void main()
{
  int m1[N_FILES][N_COLUMNES];
  int m2[N_FILES][N_COLUMNES];
  int suma[N_FILES][N_COLUMNES];
  int i,j;

  LlegirMatriu(m1);
  LlegirMatriu(m2);

  SumaMatrius(m1,m2,suma);

  printf ("Suma de les matrius:\n");
  for (i=0; i<N_FILES; i++)
  {
    for (j=0; j<N_COLUMNES; j++)
      printf ("%d ", suma[i][j]);
    printf ("\n");
  }
}
```

```
void LlegirMatriu (int m[][N_COLUMNES])
{
  int i, j;

  for (i=0; i<N_FILES; i++)
    for (j=0; j<N_COLUMNES; j++)
    {
      printf ("Introdueix valor de la fila…
      %d i columna %d: ", i, j);
      scanf ("%d", &m[i][j]);
    }
}

void SumaMatrius (int m1[][N_COLUMNES], int
m2[][N_COLUMNES],int suma[][N_COLUMNES])
{
  int i, j;

  for (i=0; i<N_FILES; i++)
    for (j=0; j<N_COLUMNES; j++)
      suma[i][j] = m1[i][j] + m2[i][j];
}
```

Apuntadors a registres

Els apuntadors poden ser de qualsevol tipus. Fins ara n'hem tractat de tipus simples:

```
int *p;
float *pB;
```

etc.

Tot el que hem explicat fins ara d'apuntadors es pot aplicar igualment a **registres**:

```
typedef struct{
        <tipus1> <camp_1>;
                 ⋮
        <tipusN> <camp_N>;
}<UnTipusRegistre>;

<UnTipusRegistre> <Var>,*<Punter>;
```

Inicialtzem (d'alguna manera) <Var>.

```
<Punter>=&<Var>;
```

Per accedir als camps del registre mitjançant apuntadors, podem fer-ho:

```
(*<Punter>).<camp_i>
```

O, de forma equivalent, i més compacta:

```
<Punter> -> <camp_i>
```

Apuntadors a registres

EXEMPLE

```
/* Utilització d'apuntadors a registres */
#include <stdio.h>

typedef struct
{
  char nom[20];
  int grup;
  int nota_final;
} reg_alumne;

void main()
{
  reg_alumne alumne, *p_alumne;

  p_alumne = &alumne;
  printf ("introdueix el nom de l'alumne: ");
  gets (p_alumne->nom);
  printf ("Introdueix el grup de l'alumne: ");
  scanf ("%d", &p_alumne->grup);
  printf ("Introdueix la nota de l'alumne: ");
  scanf ("%d", &p_alumne->nota_final);
}
```

Apuntadors a registres

PAS DE REGISTRES COM A PARÀMETRES EN FUNCIONS

Els registres es poden passar com a paràmetres per valor a les funcions, però:

Normalment s'utilitza el pas per referència amb apuntadors per definir els paràmetres de tipus registre a les funcions

Motiu:

- Pas per valor → Còpia dels valors de la variable de la crida al paràmetre de la funció.
- Els registres acostumen a ser variables que ocupen molta memòria.

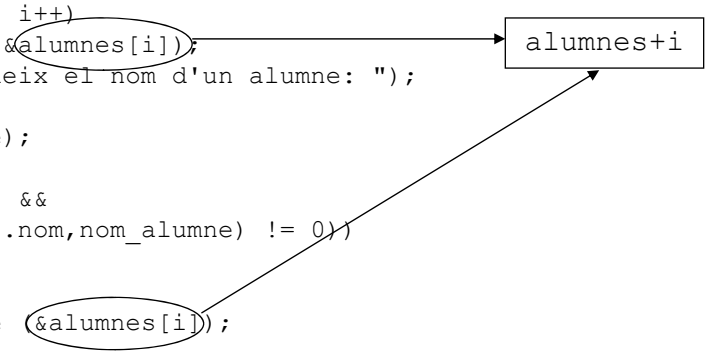
El pas per valor de registres suposa un cost en temps i espai de memòria que ens estalviem si fem el pas per referència

Apuntadors a registres

```
#include <stdio.h>
#include <string.h>
#define MAX 100

typedef struct
{
  char nom[20];
  int grup;
  int nota_final;
}reg_alumne;


void LlegirAlumne (reg_alumne *p_alumne)
{
  fflush (stdin);
  printf ("Introdueix el nom de l'alumne: ");
  gets (p_alumne->nom);
  printf ("Introdueix el grup de l'alumne: ");
  scanf ("%d", &p_alumne->grup);
  printf ("Introdueix la nota de l'alumne: ");
  scanf ("%d", &p_alumne->nota_final);
}
void EscriureAlumne (reg_alumne *p_alumne)
{
  printf ("Nom de l'alumne: %s\n", p_alumne->nom);
  printf ("Grup de l'alumne: %d\n", p_alumne->grup);
  printf ("Nota de l'alumne: %d\n", p_alumne->nota_final);
}
```

Apuntadors a registres

```
void main()
{
  reg_alumne alumnes[MAX];
  char nom_alumne[20];
  int i;

  for (i=0; i<MAX; i++)
    LlegirAlumne (&alumnes[i]);
  printf ("Introdueix el nom d'un alumne: ");
  fflush(stdin);
  gets (nom_alumne);
  i=0;
  while ((i < MAX) &&
(strcmp(alumnes[i].nom,nom_alumne) != 0))
    i++;
  if (i<MAX)
    EscriureAlumne (&alumnes[i]);
}
```

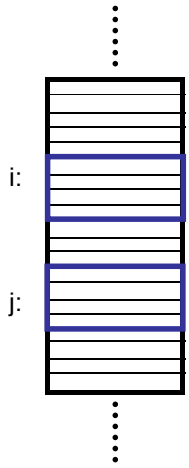


MEMÒRIA DINÀMICA

Memòria dinàmica

Memòria estàtica → Quan declareu variables dins una funció, el què fa el compilador és reservar memòria per a les variables en qüestió.

```
tipus_retorn Funcio(·····)
{
  int i,j;
    .
    .
    .
    .
}
```



Quan sortim de la funció, la memòria reservada s'ALLIBERA: es pot fer servir per a qualsevol altre ús.

Assumint que un enter ocupa 4 bytes... per emmagatzemar la variable entera `i` necessitem 4 bytes i per la `j`, 4 més.

Memòria dinàmica

Fins ara, la memòria que els programes necessiten, queda determinada per les variables que declareu ⇒ La reserva i alliberament de memòria no l'heu de fer vosaltres.

Potser alguna vegada heu pensat en fer:

```
void main(void)
{
  int Dim,i;

  printf("Entra la dimensió del vector");
  scanf("%d",&Dim);

  int Vect[Dim];

  for(i=0;i<Dim;i++)
    scanf("%d",&Vect[i]);
}
```

Evidentment això NO ÉS CORRECTE però LA IDEA SÍ.

Es tracta de reservar memòria en temps d'execució i segons les necessitats del programa → **MEMÒRIA DINÀMICA**

Memòria dinàmica

Per treballar amb memòria dinàmica → 2 funcions bàsiques

- **malloc** → Per reservar memòria
- **free** → Per alliberar memòria

malloc

```
<apuntador>=(tipus*)malloc(<tamany_de_memòria>);
```

- <tamany_de_memòria> → nombre de bytes que es volen reservar
- La funció `malloc`, busca una posició dins la memòria on hi hagi el nombre de bytes especificats. Quan la troba, en retorna l'adreça genèrica.
- Cal que especifiquem el tipus d'apuntador que volem
- Guardem l'adreça a un apuntador per poder treballar amb la memòria reservada.

free

```
free(<apuntador>);
```

- **IMPORTANT:** Quan un objecte dinàmic ja no és necessari → s'ha d'alliberar
- En general, per cada **malloc** hi ha d'haver un **free**

Memòria dinàmica

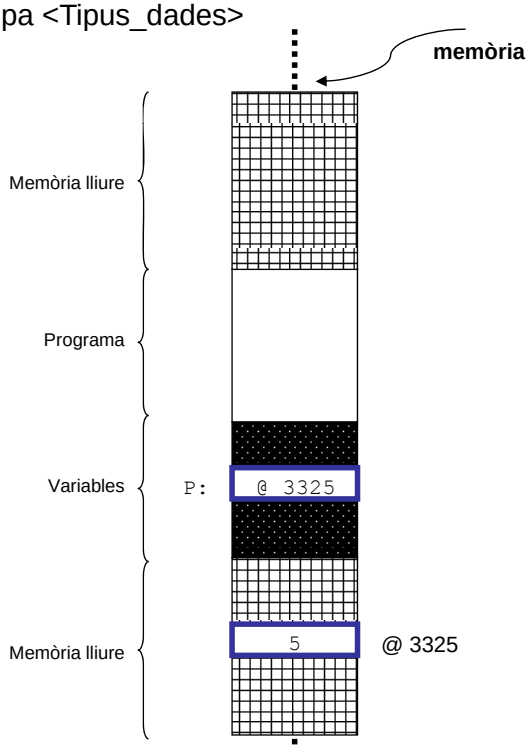
`sizeof()`→ Funció que serà de gran utilitat per reservar memòria

```
sizeof(<Tipus_dades>);
```

- La funció ens retorna el número de bytes que ocupa <Tipus_dades>

EXEMPLE

```
int *P;
.
.
P=(int*)malloc(sizeof(int));
.
.
*P=5;
.
.
free(P);
```



Memòria dinàmica

EXEMPLE

Tornem a l'exemple inicial que havíem posat per veure la necessitat de la memòria dinàmica:

```
void main(void)
{
  int Dim,i;
  int *Vect;

  printf("Entra la dimensió del vector");
  scanf("%d",&Dim);

  Vect=(int *)malloc(sizeof(int)*Dim);

  for(i=0;i<Dim;i++)
    scanf("%d",&Vect[i]);
       .
       .
       .

  free(Vect);
}
```

Memòria dinàmica

IMPORTANT: Les variables locals, es creen al començament d'una funció, i es destrueixen quan aquesta acaba. Per tant, si volem retornar un vector:

```
#include <stdio.h>
void main(void)
{
  int *vect;

  vect=Inicialitza_Vect();
  .
  .
}

int* Inicialitza_Vect (void)
{
  int V[10],i;

  for(i=0;i<10;i++)
    V[i]=0;

  return(V);
}
```

Com que hem alliberat la memòria del vector, no retornem cap adreça de memòria vàlida

Aquí es reserva la memòria de forma automàtica.

Aquí s'allibera la memòria de forma automàtica.

Memòria dinàmica

Forma correcta per retornar un vector des d'una funció:

```
#include <stdio.h>
void main(void)
{
  int *vect;

  vect=Inicialitza_Vect();
  .
  .
  free(vect);
}
```

```
int* Inicialitza_Vect (void)
{
  int *V,i;

  V=(int*)malloc(10*sizeof(int));

  for(i=0;i<10;i++)
    V[i]=0;

  return(V);
}
```

Dins vect, guardem l'adreça del primer element del vector per al que hem reservat memòria

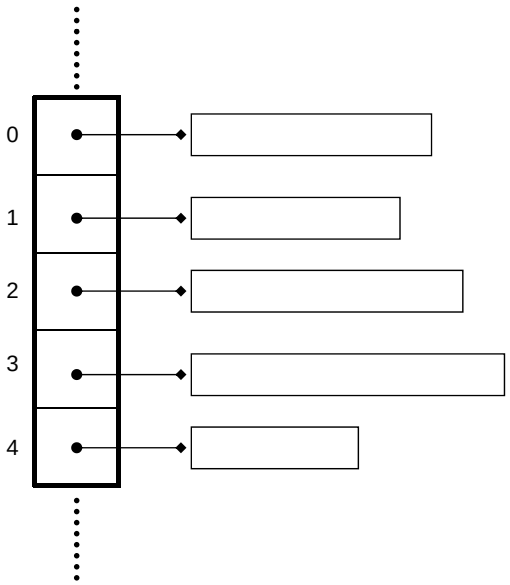
Ara reservem nosaltres mateixos la memòria que necessitem…

Finalment alliberem memòria quan calgui

Com que NO alliberem la memòria (encara)

Memòria dinàmica

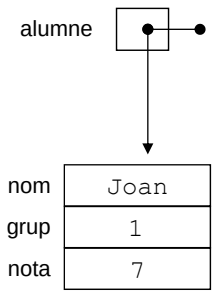
Els apuntadors són variables com qualsevol altra, amb la particularitat que emmagatzemen adreces de memòria. Per tant, també es poden declarar arrays d'apuntadors:




```
char* Linies[5];
   .
   .
   .
for (i=0; i<5; i++)
{
  printf("N° de caracters de la linia %d: ",i);
  scanf ("%d",&n);
  Linies[i]=(char*)malloc((n+1)*sizeof(char));
}
   .
   .
   .
for (i=0; i<5; i++)
  free(Linies[i]);
```

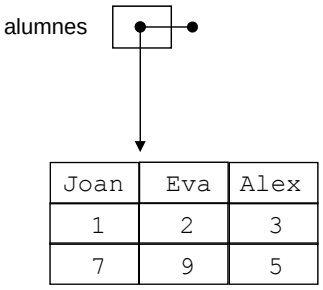
Memòria dinàmica

EXEMPLE




```
typedef struct registre_alumne{
  char nom[20];
  int grup;
  float nota;
}TipusAlumne;

   .
   .
TipusAlumne *alumne;
   .
   .
alumne=(TipusAlumne*)malloc(sizeof(TipusAlumne));
   .
   .
strcpy (alumne->nom,"Joan");
alumne->grup = 1;
alumne->nota = 7;
   .
   .
free(alumne);
```

EXEMPLE




```
typedef struct registre_alumne{
  char nom[20];
  int grup;
  float nota;
}TipusAlumne;

  .
  .
TipusAlumne *alumnes;
  .
  .
alumnes=(TipusAlumne*)malloc(3*sizeof(TipusAlumne));
  .
  .
for(i=0;i<3;i++)
  [ EMPLENEM ELS CAMPS D'ALGUNA MANERA ]
  .
  .
free(alumnes);
```