Programación Avanzada (TC2025)

Tema 1. Programación en lenguaje C

Tecnológico de Monterrey. Campus Santa Fe
Departamento de Computación
Dr. Vicente Cubells (vcubells@tec.mx)

Temario

- Apuntadores
- Aritmética de apuntadores
- Trabajo con memoria dinámica
- Arreglos
- Relación que existe entre arreglos y apuntadores
- Aritmética de apuntadores para trabajar con arreglos y matrices
- Ejemplos

Punteros o apuntadores…

- Variable que puede almacenar la dirección de otra variable
- Pueden existir punteros de punteros
- Declaración:
  - `int *iptr`
  - `float *fptr`
  - `char * string`
- Operador de dirección &
- Operador de indirección *
- Son complementos entre sí

Punteros o apuntadores…

- Ejemplos de uso de los operadores & y *

```
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 */
```

Punteros o apuntadores…

- Algunos errores comunes

```
p = &34;          /* las constantes no tienen dirección */
p = &(i+1);       /* las expresiones no tienen dirección */
&i = p;           /* las direcciones no se pueden cambiar */
p = 9879;         /* habría que escribir p = (int *)9879 */
```

- El puntero indefinido

```
int *p;
float *q;
void *r;
p = q;            /* Ilegal */
p = (int *)q;     /* Legal */
p = r = q;        /* Legal */
```

Punteros o apuntadores…

- Aritmética de punteros
  - No se pueden dividir ni multiplicar
  - Si se pueden sumar y restar

```
int *p;          /* p es un puntero a int */
double *q;       /* q es un puntero a double */
p++;             /* se le añade a p el tamaño de un int */
q += 1;          /* se le añade a q el tamaño de un double */
```

  - Si se restan dos punteros se encuentra la diferencia entre las direcciones de las variables que apuntan, no en bytes sino en el número de datos de su tipo

Punteros o apuntadores

- Aritmética de punteros: Ejercicio para ver como cambian las variables

```
int a, b, c;
int *p1, *p2;
void *p;

p1 = &a;
*p1 = 1;
p2 = &b;
*p2 = 2;
p1 = p2;
*p1 = 0;
p2 = &c;
*p2 = 3;
p = &p1;
*p = p2;
*p1 = 1;
```

Trabajo con la memoria dinámica…

```
/* Asignar memoria dinámica */
void *malloc(size_t size);

int *i;
i = (int *)malloc(sizeof(int));

/* Asignar espacios a matrices */
void *calloc(size_t nmemb, size_t size);

/* redimensiona el espacio asignado a un puntero */
void *realloc(void *ptr, size_t size);

/* Libera memoria */
void free(void *ptr);
```

Es un operador que regresa el tamaño en bytes

Entero con signo y sin signo definido en <stddef.h>

Trabajo con memoria dinámica

- Programar un ejemplo

Arreglos

- Arreglos de tipos numéricos
- Arreglos de caracteres
- Direccionamiento de una matriz en memoria
  - Para una matriz de N filas x M columnas
  - e[i,j] = Pos(e[0,0]) + i*M + j

Arreglos y la aritmética de apuntadores

```
int vector[10], matriz[3][5], *p;

p = &vector[0];
printf("%d \n", *(p+2));        // Se imprime vector[2]
p = &mat[0][0];
printf("%d \n", *(p+2));        // Se imprime matriz[0][2]
printf("%d \n", *(p+4));        // Se imprime matriz[0][4]
printf("%d \n", *(p+5));        // Se imprime matriz[1][0]
printf("%d \n", *(p+12));       // Se imprime matriz[2][2]
```

Arreglos y apuntadores

- El nombre de un arreglo es un apuntador al primer elemento del arreglo
  - double vector[10]; // vector es un puntero a vector[0]
  - Dado que vector apunta a vector[0], vector+i apuntará a vector[i]
- A un puntero se le pueden poner *índices*
  - Si p = &vector[0], entonces:
    - p[3] equivale a vector[3]
- Resumiendo, si p = vector, entonces se cumple:
  - *p equivale a vector[0], *vector y a p[0]
  - *(p+1) equivale a vector[1], *(vector+1) y a p[1]
  - *(p+2) equivale a vector[2], *(vector+2) y a p[2]

Variantes para sumar números de un vector

```
#define N 10
int a[N], suma, i, *p;

for(i=0, suma=0; i < N; ++i)
        suma += a[i];

for(i=0, suma=0; i < N; ++i)
        suma += *(a+i);

for(p=a, i=0, suma=0; i < N; ++i)
        suma += p[i];

for(p=a, suma=0; p < &a[N]; ++p)
        suma += *p;
```

Matrices y punteros…

```
int m[5][3], **p, *q;
```

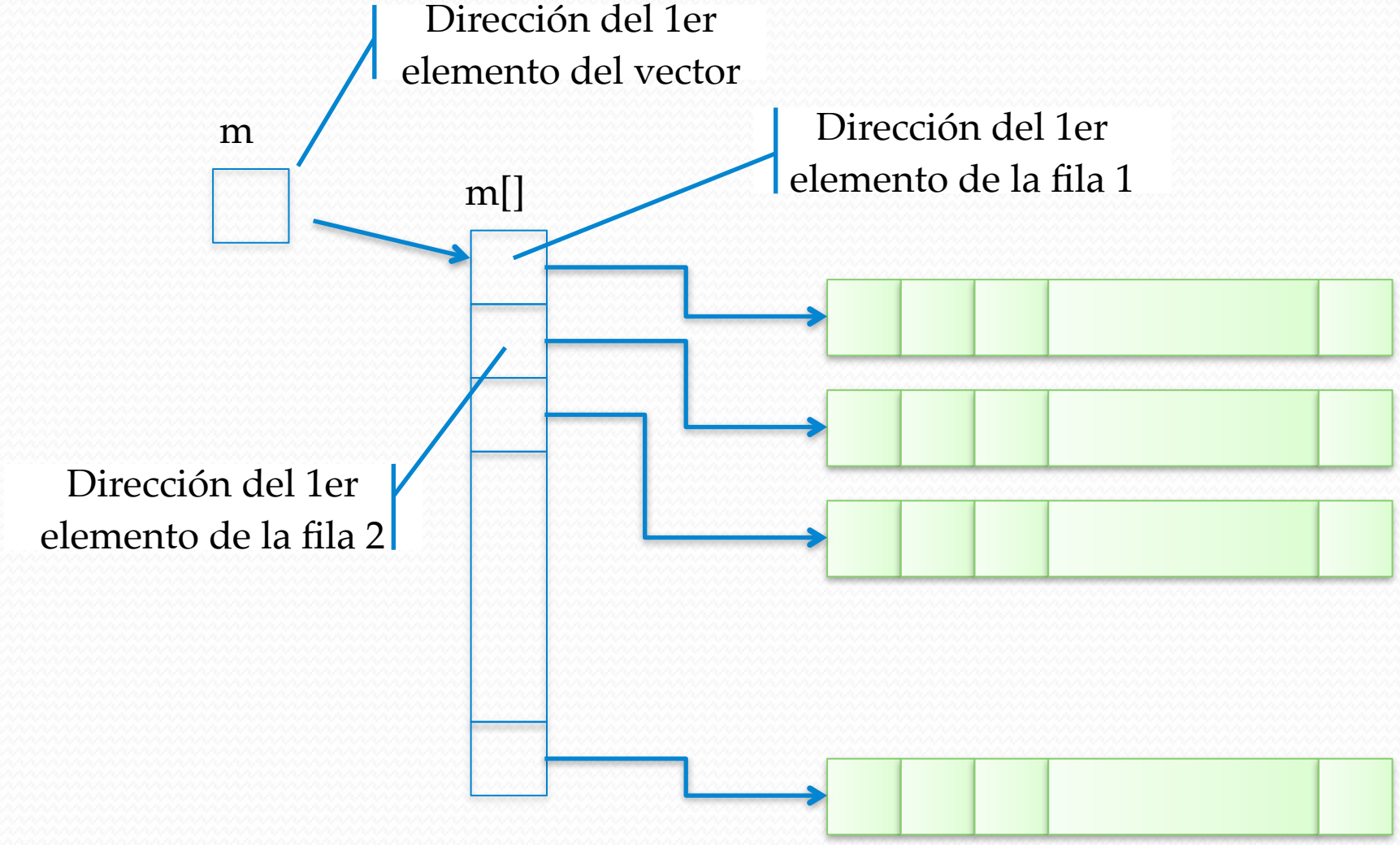
`m` es un puntero al primer elemento de un vector de punteros `m[]` cuyos elementos contienen las direcciones del primer elemento de cada fila

`m` es un *puntero a puntero*

```
m es igual a &m[0]
m[0] es &m[0][0]
m[1] es &m[1][0]

Si hacemos **p = m:
       *p     es m[0]        **p            es m[0][0]
       *(p+1) es m[1]        **(p+1)        es m[1][0]
                             *(*(p+1)+1)    es m[1][1]
```

Matrices y punteros



Matrices y punteros

- Realice un programa que imprima los elementos de una matriz utilizando aritmética de apuntadores
  - Si hacemos `q = &matriz[0][0]`
    - `*(q + M*i + j)`
    - `*(mat[i] + j)`
    - `(*(mat + i))[j]`
    - `*((*(mat + i)) + j)`

Resumiendo

- Los punteros son variables que almacenan la direción de memoria de otra variable
- Un puntero puede inicializarse con 0, NULL o una dirección
- Existen punteros a punteros
- La aritmética de punteros es de gran utilidad al trabajar con arreglos y matrices
- Los punteros permiten el uso de memoria dinámica
- Siempre que se utilice malloc() hay que utilizar free()
- Las cadenas, al ser arreglos de caracteres, pueden ser tratadas con la aritmética de apuntadores