Curso C

Memoria Dinámica

Memoria dinámica

- Consiste en asignar la cantidad de memoria necesaria para almacenar un objeto durante el tiempo de ejecución, NO de compilación.
  - Nos devuelve un puntero al bloque de memoria asignado, por lo que necesitaremos un puntero para almacenar dicha dirección.
- Funciones para manejo de memoria:
  - void* malloc (size_t nbytes) -> (esta función devuelve un puntero a void siempre que haya espacio en memoria, sino devuelve NULL).

```
int *p= (int*) malloc(nbytes);
if(p==NULL) printf("Memoria Insuficiente");
```

  - void * calloc (size_t num, size_t size);

->(hace lo mismo que malloc, pero inicializa la memoria reservada)

Memoria dinámica

- void * realloc (void * ptr, size_t size);-> (esta función reasigna el bloque de memoria apuntado por *ptr*, *cambiándolo al tamaño size*
- void free (void * ptr);->(Libera el bloque de memoria apuntado por ptr, dejandolo libre para futuros usos)

¡¡¡¡¡ OJO, toda memoria reservada con malloc/calloc debe ser liberada con free !!!!!!

Matrices dinámicas

- Utilizando memoria dinámica podemos decidir durante la ejecución el tamaño de nuestras matrices.

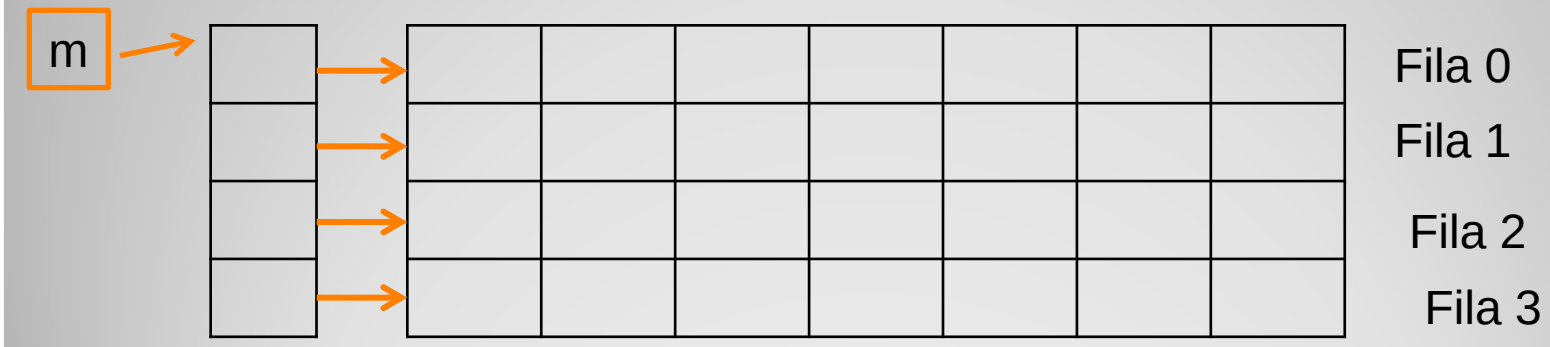
```
int *p=NULL
if( (p = (int*) malloc(100*sizeof(int)))==NULL){
    printf("memoria insuficiente");
    return -1;
}
```

  - Este ejemplo reserva un vector de 100 elementos de tipo int. Lo mismo se podria hacer con calloc, inicializando la memoria a 0.

```
int *p=NULL
if( (p = (int*) calloc(100, sizeof(int)))==NULL){
    printf("memoria insuficiente");
    return -1;
}
```

Matrices dinámicas – dos dimensiones

- Para asignar memoria matriz 2 dimensiones, proceso en 2 partes.
  - Asignar memoria para una matriz de punteros cuyos elementos referenciaran cada una de las filas
  - Asignar memoria para cada una de las filas



Matrices bidimensionales

```
int**m;
m=(int**)malloc(nfilas*(sizeof(int*));
for(f=0;f<nfilas;f++)
   m[f]=(int*)malloc(ncol*sizeof(int));
```

- Para liberar la memoria, el proceso es el inverso

```
for(f=0;f<nfilas;f++)
   free(m[f]);

free(m);
m=NULL;
```

Matrices de cadenas de caracteres

- char nombre[FILAS][COLUMNAS].
  - Esto define una matriz ESTATICA de caracteres, donde todos las filas nombre[i] tienen el mismo numero de caracteres.
- Usando memoria dinámica, podremos tener cadenas de diferente longitud.
  - char**nombre=(char**)malloc(FILAS*(sizeof(char*));
  - A diferencia de las matrices numericas, no reservamos memoria para las filas, hasta que no se lean.

```
char cad[90];
gets(cad);
nombre[i]=(char*)malloc(strlen(cad)+1))
```

Programa que lea n filas cadenas, las ordene y las imprima por pantalla

Punteros a estructuras

- Se declaran de igual forma que a otros tipos básicos.
  - Para acceder a los miembros de la estructura usaremos el operador '->' en vez del '.'

```
Typedef struct {
  int dd;
  int mm;
  int aa;
}fecha;
//MAIN
fecha *hoy=NULL
hoy = (struct fecha*) malloc(sizeof(struct fecha));
printf("Introducir fecha (dd-mm-aa)\n");
scanf("%d-%d-%d",&hoy->dd, &hoy->mm, &hoy->aa);
```

Ver libro ceballos, pag 305

Más sobre funciones:

- Ojo, en matrices multidimensionales, cuando se pasan por parámetro, hay que especificar la segunda y restantes dimensiones. (todas menos la primera)
  - float a[FILAS][COLS], c[FILAS][COLS];
    CopiarMatriz(c,a);

```
void CopiarMatriz(float destino[][COLS], float orig[][COLS]){
...
}
```

Más sobre funciones:

- Con matrices dinámicas:

```
float **m=(float**)malloc(filas*sizeof(float*));
 for(f=0;f<filas;f++)
    m[f]=(float*)malloc(cols*sizeof(float));

 Void Visualiza(float **x, int filas, int cols){
 }
```

- Pasar PUNTERO como argumento a una Función.

PIZARRA

- Retornar un PUNTERO al bloque de datos.

PIZARRA

Argumentos en la Línea de comandos

- Muchas veces, cuando invocamos un programa desde el sistema operativo necesitamos escribir uno o más argumentos tras escribir el nombre del programa, separados por espacios.
  - Ej: dir /p
- Quien recibe estos argumentos?
  - La función main

- El prototipo completo del main es:
  - int main(int argc, char* argv[]);
  - Donde: argc indica el número de argumentos pasados a través de la línea de ordenes, incluido el nombre del programa
  - argv es una matriz de punteros a cadenas de caracteres. Cada elemento de la matriz, contienne un argumento.
    - argv[0] contiene el nombre del programa
    - argv[1] el primer argumento, etc.