Clase 2. Arreglos, memoria dinámica, lectura de archivos.

Apuntadores y memoria dinámica para uso en arreglos

Declaración de Apuntadores

Un apuntador es una variable que guarda una dirección de memoria. Normalmente decimos que el “apuntador” apunta a una variable tal o cual.

Posicion en la RAM	0x7fffffffab0	0x7fffffffabc	0x7fffffffac4	0x7fffffffabc
Datos alamacenados	2131895	0x7fffffffab0	3.282e-9	0x7fffffffab8
	int x;	**int** *y=&x;	**float** z;	**float** *y=&z;

- Aunque la memoria es requerida en el Stack, como comentamos el que se reserve esta memoria (el Stack), y el orden en que se requiere la memoria, no depende de C, sino del sitema operativo.

Declaración de Apuntadores

Un apuntador es una variable que guarda una dirección de memoria. Normalmente decimos que el “apuntador” apunta a una variable tal o cual.

Posicion en la RAM	0x7fffffffab0	0x7fffffffabc	0x7fffffffac4	0x7fffffffabc
Datos alamacenados	2131895	0x7fffffffab0	3.282e-9	0x7fffffffab8
	int x;	**int** *y=&x;	**float** z;	**float** *y=&z;

- Aunque la memoria es requerida en el Stack, como comentamos el que se reserve esta memoria (el Stack), y el orden en que se requiere la memoria, no depende de C, sino del sitema operativo.
- Aun así, es altamente probable, que la memoria que se declara en orden contiguo este muy cercana en el Stack.

Operadores * & y []

Como vimos el operador & devuelve la dirección de memoria de una variable.

Posicion en la RAM	0x7fffffffab0	0x7fffffffabc	0x7fffffffac4	0x7fffffffabc
Datos alamacenados	2131895	0x7fffffffab0	3.282e-9	0x7fffffffab8
	int x;	**int** *y=&x;	**float** z;	**float** *y=&z;

Para escribir sobre la variable x, podemos utilizar la variable en si:

x=5;

O la dirección de memoria (almacenada en y), con el operador *:

*y=5;

Que es equivalente a utilizar el operador []:

y[0]=5

Las instrucciones anteriores son equivalentes.

Apuntador NULL

Un apuntador usualmente apunta a una dirección de memoria, el otro posible valor es NULL.
NULL es una macro definida en varias librerias de C, en varias librerias: stddef.h, locale.h, stdio.h, stdlib.h time.h
La macro se utiliza para indicar que un apuntador no apunta a una dirección valida.
La mayoria de las implementaciones simplemente la consideran como 0. Pero lo único que se tiene que asegurar es que es una dirección no valida (que no es ninguna que si exista en la RAM y pueda ser asignada a otra cosa).
Muchas funciones la utilizan para indicar precisamente eso: que la memoria que se está indicando es no valida, por ejemplo:

- Cuando un archivo no se puede abrir fopen() regresa NULL.
- Cuando la memoria requerida no se puede reservar, se regresa NULL.
- En estructuras de datos, para indicar que un grafo llego asi nivel mas bajo, se puede asignar NULL al un apuntador en el último nivel.

Memoria dinámica

En el caso anterior los apuntadores podían ser usados para acceder a memoria estática. Y es la forma, también, de acceder a memoria dinámica.
Para manejar la memoria dinámica en C, se require:

1. **Declarar/definir una variable donde guardar la dirección de donde empieza la memoria.**
2. **Requerir la cantidad de memoria que ocupemos (con una función de C).**
3. **Utilizar la memoria.**
4. **Liberar la memoria.**

malloc

malloc

malloc es una función que permite requerir memoria dinámica, recordemos que esta se reserva en el **heap**. La función se encuentra stdlib.h, al igual que la funcion **free()**. Que permite liberarla.

- Prototipo: void* malloc (size_t size);
  Recibe el tamaño como un tipo size_t que es un entero sin signo de al menor 16 bits (nota, si se envia un negativo, se hará casting a entero sin signo). Devuelve el apuntador de donde inicia la memoria reservada.

malloc

malloc es una función que permite requerir memoria dinámica, recordemos que esta se reserva en el **heap**. La función se encuentra stdlib.h, al igual que la funcion **free()**. Que permite liberarla.

- Prototipo: void* malloc (size_t size);
  Recibe el tamaño como un tipo size_t que es un entero sin signo de al menor 16 bits (nota, si se envia un negativo, se hará casting a entero sin signo). Devuelve el apuntador de donde inicia la memoria reservada.
- El tamaño es en bytes (lo que devuelve size of).

malloc

malloc es una función que permite requerir memoria dinámica, recordemos que esta se reserva en el **heap**. La función se encuentra stdlib.h, al igual que la funcion **free()**. Que permite liberarla.

- Prototipo: void* malloc (size_t size);
  Recibe el tamaño como un tipo size_t que es un entero sin signo de al menor 16 bits (nota, si se envia un negativo, se hará casting a entero sin signo). Devuelve el apuntador de donde inicia la memoria reservada.
- El tamaño es en bytes (lo que devuelve size of).
- El sistema lleva el registro de la memoria requerida a partir de la posición devuelta.

malloc

malloc es una función que permite requerir memoria dinámica, recordemos que esta se reserva en el **heap**. La función se encuentra stdlib.h, al igual que la funcion **free()**. Que permite liberarla.

- Prototipo: void* malloc (size_t size);
  Recibe el tamaño como un tipo size_t que es un entero sin signo de al menor 16 bits (nota, si se envia un negativo, se hará casting a entero sin signo). Devuelve el apuntador de donde inicia la memoria reservada.
- El tamaño es en bytes (lo que devuelve size of).
- El sistema lleva el registro de la memoria requerida a partir de la posición devuelta.
- * El sistema no libera la memoria automáticamente (en sistemas modernos, se libera automáticamente al terminar el proceso).

malloc

malloc es una función que permite requerir memoria dinámica, recordemos que esta se reserva en el **heap**. La función se encuentra stdlib.h, al igual que la funcion **free()**. Que permite liberarla.

- Prototipo: void* malloc (size_t size);
  Recibe el tamaño como un tipo size_t que es un entero sin signo de al menor 16 bits (nota, si se envia un negativo, se hará casting a entero sin signo). Devuelve el apuntador de donde inicia la memoria reservada.
- El tamaño es en bytes (lo que devuelve size of).
- El sistema lleva el registro de la memoria requerida a partir de la posición devuelta.
- * El sistema no libera la memoria automáticamente (en sistemas modernos, se libera automáticamente al terminar el proceso).
- malloc requiere un bloque de memoria, es decir, que está continua.

malloc

malloc es una función que permite requerir memoria dinámica, recordemos que esta se reserva en el **heap**. La función se encuentra stdlib.h, al igual que la funcion **free()**. Que permite liberarla.

- Prototipo: void* malloc (size_t size);
  Recibe el tamaño como un tipo size_t que es un entero sin signo de al menor 16 bits (nota, si se envia un negativo, se hará casting a entero sin signo). Devuelve el apuntador de donde inicia la memoria reservada.
- El tamaño es en bytes (lo que devuelve size of).
- El sistema lleva el registro de la memoria requerida a partir de la posición devuelta.
- * El sistema no libera la memoria automáticamente (en sistemas modernos, se libera automáticamente al terminar el proceso).
- malloc requiere un bloque de memoria, es decir, que está continua.
- *Si el tamaño requerido es 0, puede o no devover un puntero NULL.

malloc

malloc es una función que permite requerir memoria dinámica, recordemos que esta se reserva en el **heap**. La función se encuentra stdlib.h, al igual que la funcion **free()**. Que permite liberarla.

- Prototipo: void* malloc (size_t size);
  Recibe el tamaño como un tipo size_t que es un entero sin signo de al menor 16 bits (nota, si se envia un negativo, se hará casting a entero sin signo). Devuelve el apuntador de donde inicia la memoria reservada.
- El tamaño es en bytes (lo que devuelve size of).
- El sistema lleva el registro de la memoria requerida a partir de la posición devuelta.
- * El sistema no libera la memoria automáticamente (en sistemas modernos, se libera automáticamente al terminar el proceso).
- malloc requiere un bloque de memoria, es decir, que está continua.
- *Si el tamaño requerido es 0, puede o no devover un puntero NULL.
- En C89 malloc regresaba un char*.

free()

free es la funcion que se utiliza para indicar que la memoria reservada por malloc o realloc, se ha liberado. Toda memoria reservada por malloc debe de ser liberada por free().

- prototipo: void free (void* ptr);
- Si ptr no apunta a una dirección valida (al inicio de memoria reservada por malloc) el comportamiento no está definido.

free()

free es la funcion que se utiliza para indicar que la memoria reservada por malloc o realloc, se ha liberado. Toda memoria reservada por malloc debe de ser liberada por free().

- prototipo: void free (void* ptr);
- Si ptr no apunta a una dirección valida (al inicio de memoria reservada por malloc) el comportamiento no está definido.
- Note que: ptr debe de ser el inicio de la memoria, si enviamos otra dirección aunque esa dirección este dentro de la memoria reservada, el puntero no es valido.

free()

free es la funcion que se utiliza para indicar que la memoria reservada por malloc o realloc, se ha liberado. Toda memoria reservada por malloc debe de ser liberada por free().

- prototipo: void free (void* ptr);
- Si ptr no apunta a una dirección valida (al inicio de memoria reservada por malloc) el comportamiento no está definido.
- Note que: ptr debe de ser el inicio de la memoria, si enviamos otra dirección aunque esa dirección este dentro de la memoria reservada, el puntero no es valido.
- free libera la memoria del bloque. No requiere el tamaño de la memoria, solo el inicio del bloque, el sistema, existe un registro (no disponible al programador) de la cantidad de memoria requerida/liberada.

Notas acerca de free y malloc

- Usualmente el retorno de malloc se maneja con un cast al tipo de apuntador donde lo recibimos. Este cast no es necesario en C (ya que los apuntadores son del mismo tamaño), pero como practica es deseable, para saber de que tipo recibimos la memoria. Lo bueno y malo, es que puede esconder un warning (acerca de que se almacenan datos de un tipo en otro), es malo si es por un error, es bueno si es porqué así lo deseamos. C++ si requiere el cast de manera forzosa.
- La implementación de malloc y free dependen del compilador y el sistema. Por lo cual, la memoria para almacenar puede no ser un multiplo exacto del tipo o dimensión que se requiere. Puede tener que ver con eficiencia de las arquitecturas (tamaños de memoria, tamaños de línea de caché, prefetchers).

Ejemplo malloc

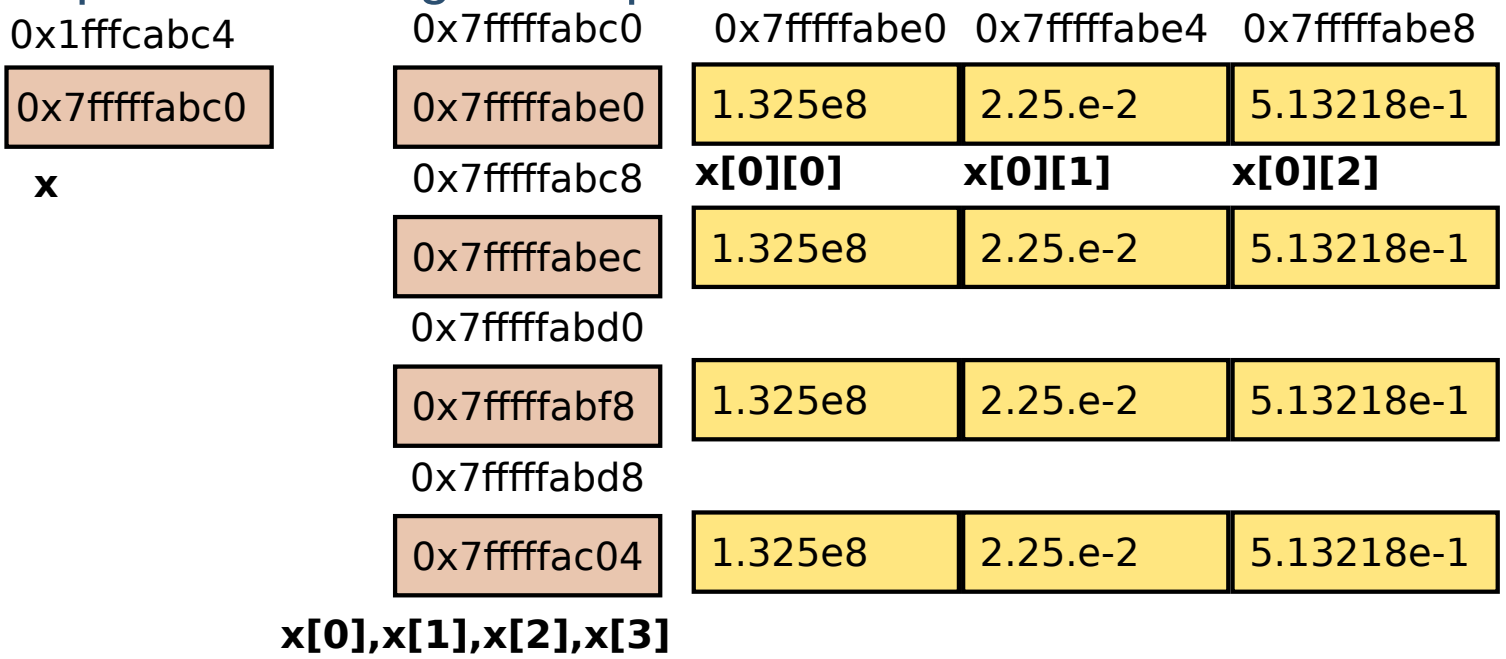
Un ejemplo completo de como requerir memoria dinámica con malloc es el siguiente:

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   int main()
5  {
     float *x;
7    int n;

9    printf("Inserte tamanio de la memoria requerida:\n");
     scanf("%d",&n);
11
     x=(float*)malloc(n*sizeof(float));
13
     free(x);
15
   return 0;
17 }
```

Arreglos dinámicos en 2 o mas dimensiones.

Para declarar un arreglo de dos o mas dimensiones, se requiere un arreglo de apuntadores.



Ejemplo malloc dos dimensiones

Un ejemplo de como requerir memoria dinámica con malloc es el siguiente:


```
1   float **x; //Variable tipo apuntador a apuntador
    int n=5,m=3;
3   printf("Inserte n y m:\n");
    scanf("%d %d",&n, &m);
5   //Memoria para un arreglo de apuntadores
    x=(float**)malloc(n*sizeof(float*));
7   //A cada uno de los apuntadores anteriores se les
      asigna un arreglo de flotantes
    for (int i=0; i<n; i++)
9       x[i]=(float*)malloc(m*sizeof(float));

11  ///Libero cada uno de los bloques (renglones) de
      flotantes
    for (int i=0; i<n; i++)
13      free(x[i]);
    //Libero el bloque de apuntadores
15      free(x);
return 0;
17  }
```

Sobre ejemplo anterior

Note:

- Cada bloque es continuo, sin embargo entre ellos puede ser que no lo sean.

Sobre ejemplo anterior

Note:

- Cada bloque es continuo, sin embargo entre ellos puede ser que no lo sean.
- Despues de llamar a la funcion free el apuntador no cambia (y no devuelve nada), pero hay que tener presente que en cuanto se libera la memoria esta puede ser reutilizada incluso por otro programa.

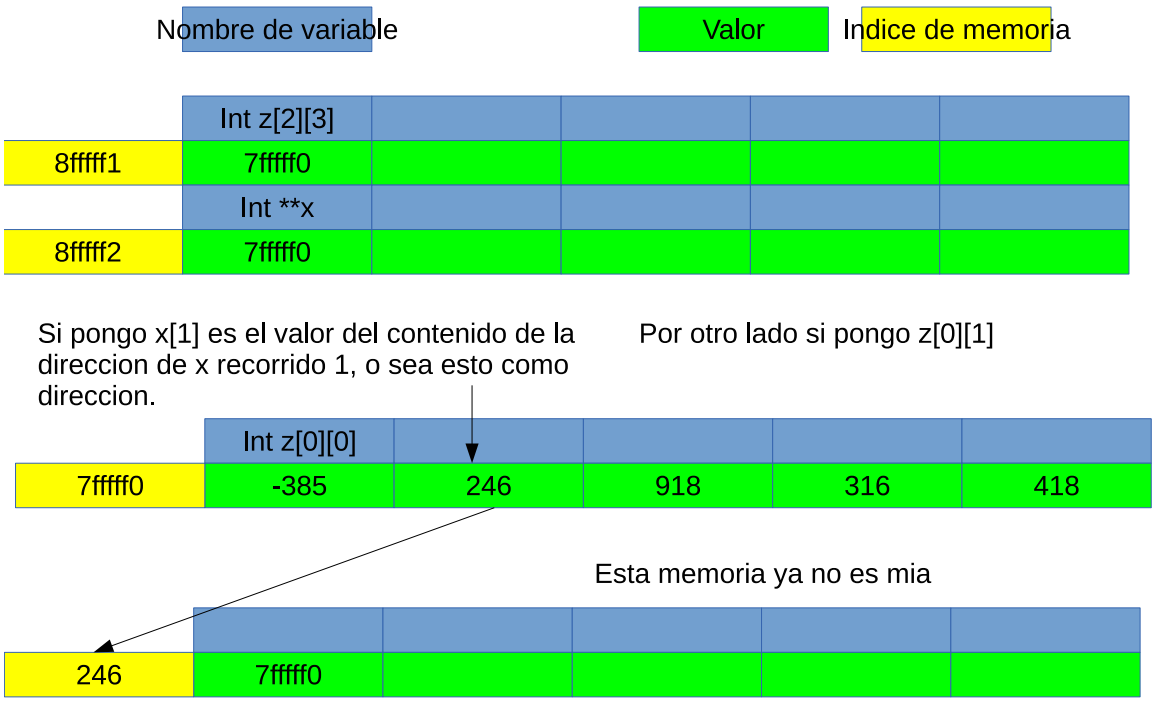
Sobre ejemplo anterior

Note:

- Cada bloque es continuo, sin embargo entre ellos puede ser que no lo sean.
- Despues de llamar a la funcion free el apuntador no cambia (y no devuelve nada), pero hay que tener presente que en cuanto se libera la memoria esta puede ser reutilizada incluso por otro programa.
- Los sistemas actuales, usualmente, liberan la memoria al salir del main, pero no es una garantía (por ejemplo los sistemas embebidos o para mobil podrían no hacerlo).

Diferencias entre arreglo estático y dinámico

Supongamos que igualo un apuntador (doble) int **x a la direccion de una mattriz estática int z[2][3]; x=z; x[1] es la dirección a la que apunta z recorrida un entero, que es z[0][1];



Errores comúnes


```c
#include <stdio.h>
#include <stdlib.h>
int fun1(int n)
{ //Memoria no liberada
  float *x=(float*)malloc
    (2*sizeof(float));
  for (int i=0; i<2; i++)
    x[i]=i;
return 0;
}

int main()
{
   for (int i=0; i<10000;
    i++)
      fun1(i);
return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int n=9,m=3;
   float **x=(float**)
    malloc(n*sizeof(float
    *));
  for (int i=0; i<n; i++)
      x[i]=(float*)malloc(
    m*sizeof(float));
  x[3]=92; //asignacion a
    direccion de memoria
  for (int i=0; i<n; i++)
   for (int j=0; j<n; j++)
    //limite del for
     x[i][j]=i*m+j;
return 0;
}
```

Aritmética de apuntadores

El avance de los apuntadores se define por el tipo de apuntador que es, ejemplo:

```
void main(){
    char *ptrChar=0;
    int *ptrInt=0;
    float *ptrFloat=0;
    double *ptrDouble=0;
    printf("1 char*=%p int*=%p float*=%p double*=%p\n",
     ptrChar++,ptrInt++,ptrFloat++,ptrDouble++);
    printf("2 char*=%p int*=%p float*=%p double*=%p\n",
     ptrChar--,ptrInt--,ptrFloat--,ptrDouble--);
    printf("3 char*=%p int*=%p float*=%p double*=%p\n",
     ptrChar,ptrInt,ptrFloat,ptrDouble);
}
```

Salida:

1 char*=(nil) int*=(nil) float*=(nil) double*=(nil)
2 char*=0x1 int*=0x4 float*=0x4 double*=0x8
3 char*=(nil) int*=(nil) float*=(nil) double*=(nil)

Ejemplo aritmética de apuntadores


```
1  int main(){
   char c; int i; float f; double d;
3  char *ptrC=&c; int *ptrI=&i; float *ptrF=&f;
   double *ptrD=&d; void *v;
5  v=ptrC;    v++;
   printf("ptrC=%p ptrC++=%p v=%p\n", ptrC, ptrC+1,v);
7  v=ptrI;    v++;
   printf("ptrI=%p ptrI++=%p v=%p\n", ptrI, ptrI+1,v);
9  v=ptrF;    v++;
   printf("ptrF=%p ptrF++=%p v=%p\n", ptrF, ptrF+1,v);
11 v=ptrD;    v++;
   printf("ptrD=%p ptrD++=%p v=%p\n", ptrD, ptrD+1,v);
13 }
```

Salida:

```
 ptrC=0x7fff424f10e7 ptrC++=0x7fff424f10e8 v=0x7fff424f10e8
ptrI=0x7fff424f10e0 ptrI++=0x7fff424f10e4 v=0x7fff424f10e1
ptrF=0x7fff424f10dc ptrF++=0x7fff424f10e0 v=0x7fff424f10dd
ptrD=0x7fff424f10d0 ptrD++=0x7fff424f10d8 v=0x7fff424f10d1
```

Accediendo a las variables con artimetica


```
1  int main(){
       float ff[3]={0.1,0.2,0.3};
3      double *ptrD=&d; void *v;
       printf("ff[0] ff[1] ff[2]=%f\n",f[0],f[1],f[2]);
5      v=ff;   v+=2*sizeof(float);  *((float*)v)=50.0;
       printf("ff[0] ff[1] ff[2]=%f\n",f[0],f[1],f[2]);
7  }
```

Salida:

```
 ff[0]=0.100000 ff[1]=0.200000 ff[2]=0.300000
ff[0]=0.100000 ff[1]=0.200000 ff[2]=50.000000
```

Preguntas/ejercicios sobre arreglos

- Hemos comentado que es mas eficiente que una matríz sea un solo bloque de memoria. ¿Como podríamos lograr eso con memoria dinámica (y accediendo con doble corchete x[i][j]).
- ¿Como requeririamos memoria para una matríz triangular?
- ¿Como podríamos requerir memoria para que los indices comenzaran desde 1 en lugar de 0?
- ¿Como podríamos requerir una locación de memoria extra para guardar el tamaño de un bloque, por ejemplo, para funciones que solo reciben el apuntador al bloque de memoria, y en la posición -1 tiene el tamño del arreglo.

Apuntadores y archivos

Apuntadores y archivos de texto

Cuando un archivo no existe o no puede ser abierto la función devuelve NULL.

```
1   if( fopen("archivo.txt","r")==NULL )
        printf("archivo.txt no existe");
3   if( fopen("archivo.txt","w")==NULL )
        printf("archivo.txt no pudo ser creado");
```

Leer archivos de texto hasta el final

EOF es una macro definida (en stdio.h), que indica el final de un archivo.
Puede ser usada para saber cuando hay que detener la lectura con, por ejemplo, fscanf.
fscanf sirve como función de alto nivel, pero solo lee palabras, se salta todos los espacios y saltos de línea. La siguiente clase veremos otras opciones.
Cuando un archivo se accede para lectura a través del apuntador es como leemos o escribimos:

```
FILE *in=fopen("archivo.txt","r");
int i=0;
//Considerando que la palabra mas larga es de 1024
   caracteres
char s[1024];
if (in)
   while(i++,fscanf(in,"%s",s)!=EOF)
         printf("cuenta=%i palabra=%s ",i,s);
printf("\n",i,s);
```

Comentarios sobre FILE

- FILE es un tipo compuesto (estructura) en C, con lo que hemos visto hasta ahora, es un tipo de variable definido por el usuario que puede alamacenar varios valores de diferente tipo.
- Esta variable almacena el tamaño del archivo, y la posición donde se está leyendo entre otras cosas.
- Su implementación depende principalmente del sistema operativo, pero también del compilador.

Tarea

prog2.7 Escriba funciones para requerir y devolver memoria dinámica para vectores y matrices tipo int, float y double. Verifique que las funciones realizadas en esta misma tarea para memoria estática funcionen adecuadamente con memoria dinámica (sino funcionan, comenten porque en su reporte).

Referencias

Uso de dll con MinGw.
Solo agregue la carpeta bin de MinGw a la variable de entorno pat
e hice todo desde consola.
http://www.transmissionzero.co.uk/computing/building-dlls-with-mi
Apuntadores
http://www.tutorialspoint.com/cprogramming/c_pointers.htm
http://www.cprogramming.com/tutorial/c/lesson6.html
http://pw1.netcom.com/~tjensen/ptr/ch1x.htm
Apuntador NULL
http://www.c4learn.com/c-programming/c-null-pointer/
malloc
http://www.cplusplus.com/reference/cstdlib/malloc/
free
http://www.cplusplus.com/reference/cstdlib/free/
ftell
http://www.tutorialspoint.com/c_standard_library/c_function_ftel

Temas pendientes

- Streams
- Completar apuntadores con estructuras de datos.
- Apuntadores a función.
- Funciones de manipulacion de archivos por medio de FILE.
- Otras funciones de lectura/escritura de archivos y salidas.

- Streams básicos.