PUNTEROS (1 y 2):

Un puntero es una variable que contiene una direccion de memoria, por lo general esta apunta a la posicion de memoria de otra variable, 'apunta a otra variable'.

La manera de declarar un puntero es:

tipo\_de\_dato\* nombre\_variable;

Los operadores son:

&-> operador que retorna la direccion de memoria de una variable

\*-> retorna el valor de la variable apuntada por un puntero

Aritmetica de punteros:

Posición Actual a++ a-- a=a+2 a=a-3

char \*a 0xA080 0xA081 0xA07F 0xA082 0xA07D

a = a + num;

La posición a la que apunta a se incrementa en:

nueva dirección que contiene “a” = dirección que contiene “a” + (num \* sizeof(tipo))

Para la resta se decrementa de igual forma en:

nueva dirección que contiene “a” = dirección que contiene “a” - (num \* sizeof(tipo))

Vectores y punteros:

nombre del array sin índice es la dirección de comienzo del array para entrar al contenido, en notacion regular seria p[i] en notacion de puntero \*(p+i)

Es posible, definir un array de variables puntero int\* a[10];(10 de tipo int)

Tambien, es posible definir un puntero a un puntero(\*\*punteroPuntero (a través del puntero a puntero))

Comparación de punteros

Sobre las variables de tipo puntero es posible realizar operaciones de comparación,

Se debe recordar que en general, cuando se suma 1 a un puntero, el valor numérico de la dirección no se incrementa en 1, sino en:

sizeof(tipo)

Entonces, al sumar “i” a un puntero se tiene que la dirección obtenida al calcular “puntero+i” tiene el valor:

direccion\_actual\_del\_puntero + (i \* sizeof(tipo))

Cuando se usan los punteros a estructura la forma de acceder a un campo es usando el operador FLECHA (->)

AGREGAR ARRAY DE PUNTEROS

MEMORIA DINAMICA:

-Segmentos de memoria:

Segmento de codigo: se guardan instrucciones de nuestro programa

Segmento de memoria estatica: se guardan variables globales del programa

Segmento de Pila: se guardan llamados a funciones, parametros de funciones llamadas, variables locales, otra info. necesaria para el funcionamiento del programa

Segmento de Heap: se guardan las variables que se crearon dinamicamente en tiempo de ejecución

Funciones para reservar dinamicamente espacio en memoria:

malloc() -> ej: var=(int \*)malloc(sizeof(int))

aplicable a cualqueir tipo de dato casteandola al deseado (tipo \*).

calloc() -> ej: var=(int \*)calloc(num\_elementos,sizeof(int));

igual que malloc, solo que tambien inicializa todo en 0, y un parametro es la cantidad

de datos a los que le va a aplicar la funcion

Funciones para redimensionar espacio reservado en memoria:

realloc() -> ej: punteroAuxiliar = (char \*) realloc ( punteroA, 4 \* sizeof(char) );

\*TODAS DEVUELVEN PUNTEROS A NULL SI NO SE PUDIERON REALIZAR\*

Para liberar el espacio reservado en memoria dinámicamente, se debe utilizar la función: free(tipo\_de\_dato)

ARCHIVOS:

En C, todas las operaciones que se realizan sobre archivos son hechas a través de funciones.

Básicamente existen 2 categorías de funciones para trabajar con archivos y son las que usan

“buffer” y las que acceden directamente al archivo.

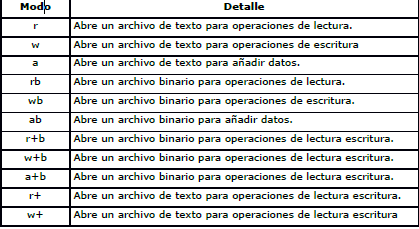
Algunas de las funciones usadas para trabajar con archivos son:

**FILE \* fopen (const char \*Nombre\_de\_archivo , const char \*Modo)->**

**Nombre\_de\_archivo**: Es una cadena de caracteres que representa el archivo , es decir se

pone la ruta y el nombre del archivo

**Modo**: Es una cadena de caracteres que determina el modo en el que será abierto el archivo.



**Modo escritura (w):** Si se abre un archivo para operaciones de escritura (w ,wb ,w+b ,w+) y

el archivo no existe se creará , pero si existe todos los datos del archivo serán borrados.

Si el archivo que se desea abrir tiene atributo de solo lectura o el disco esta lleno, etc. la

función ***'*fopen()**' devuelve error.

Cada vez que se abra un archivo en este modo, el indicador de posición se encuentra al

comienzo del archivo. Si se abre un archivo en el modo w o wb solamente se puede escribir. Si

se intenta leer datos del archivo no va a aparecer ningún error, simplemente lo que aparezca

como dato leido no va a reflejar la realidad.

**Modo lectura ( r ):** Si se abre un archivo para operaciones de lectura ( r , rb , r+b , r+ ) , si

el archivo no existe la función '**fopen()**' devuelve error.

Cada vez que se abra un archivo en este modo, el indicador de posición se encuentra al

comienzo del archivo.

Si se abre un archivo en el modo r o rb solamente se pueden realizar lecturas. No tiene ningún

efecto realizar operaciones de escritura sobre al archivo , es decir por mas que se intente

escribir sobre el archivo no se va a poder.

**Modo append ( a ):** Si se abre un archivo para agregar datos ( a , ab , a+b , a+ ) y el

archivo no existe se creará, caso contrario el indicador de posición del archivo queda

posicionado al final del mismo de forma de poder agregar datos.

Cada vez que se agregan datos se hace al final del archivo.

**int fclose(FILE \*fp); ->**

**Valor retornado:** Si el archivo es cerrado exitosamente se retorna un 0 , en caso contrario se

devuelve –1;

**int fcloseall(void);->**

cierra todos los archivos que se encuentran abiertos.

Si la operación es exitosa retorna la cantidad de archivos que se cerraron , en caso contrario

devuelve –1.

**int fread** ( **void** \* destino , size\_t tamaño , size\_t cantidad , FILE \*arch );->

• **destino:** Es un puntero al lugar donde se va a dejar el dato leido con fread

• **tamaño:** Es el tamaño en bytes del dato a leer

• **cantidad:** Es la cantidad de elementos de longitud tamaño que se van a leer

• **arch:** Es el puntero a la estructura FILE asociada al archivo desde el que se va a leer.

**Valor retornado:** Devuelve el número de datos leidos (cantidad). Si el valor retornado es

menor al que se indicó por medio de la variable

A diferencia de lo que ocurre en la escritura, se debe verificar que se realice la lectura mientras

no se haya llegado al final del archivo. Esta operación se realiza por medio de la función feof.

La función feof determina si se ha llegado al final de el archivo , el prototipo es:

**int feof** (FILE\* arch );

**int fwrite** ( **void** \* origen , size\_t tamaño , size\_t cantidad , FILE \*arch);->

• **origen:** Es un puntero al lugar desde donde se obtienen los datos para escribir en el

archivo

• **tamaño:** Es el tamaño en bytes del dato que se va a escribir

• **cantidad:** Es la cantidad de datos de longitud tamaño que se van a escribir

• **arch:** Es el puntero a FILE asociado al archivo

Nota: size\_t es un unsigned int definido en stdio.h

**Valor retornado:** Devuelve el número de datos escritos (cantidad). Si el valor retornado es

menor al que se indicó por medio de la variable cantidad , significa que hubo un error en la escritura.

**int feof(FILE \*f);->**

determina si se ha llegado al final de el archivo.

**int ferror(FILE \*fp);**

**void rewind(FILE \*fp);->**

Esta función permite llevar el indicador de posición al comienzo del archivo.

**int fseek(FILE \*fp,long num,int origen);->**

Esta función permite desplazar el indicador de posición del archivo a la posición que se le

indique.

• **arch:** Puntero a la estructura FILE asociada con el archivo

• **desplazamiento:** Es la cantidad de bytes que se desplazará el indicador de posición a

partir de origen

• **origen:** Es una constante que determina el punto de referencia a partir del cuál se

realiza el desplazamiento.

**long ftell(FILE \*f);->**

La función ftell me permite obtener la posición actual del indicador de posición.

**Valor retornado:** Si la operación es exitosa devuelve la cantidad de bytes que hay desde el

commienzo del archivo hasta el lugar en que se encuentra el indicador de posición del archivo ,

en caso contrario devuelve –1L (-1 como tipo long).

**int getc(FILE \*fp);**

**int putc(int ch,FILE \*fp);**

**char\* fgets(char \*str,int n,FILE \*fp);**

**int fputs(const char \*str,FILE \*fp);**

**int fscanf(FILE \*fp,const char \*formato[,dirección,...]);**

**int fprintf(FILE \*fp,const char \*formato[,argumento,...]);**