**Estructura:** agrupaciones de datos con organización interna y operaciones propias. Esto es ventajoso para acceder a la información y manejar los datos. Los lenguajes de programación permiten manipular una estructura completa como si se tratara de un dato individual. Pueden clasificarse en:

• Contiguas: son aquellas que se van a representar en la memoria en posiciones adyacentes.

• Enlazadas: aquellas en que no hay obligación de situarse de forma contigua en la memoria. Los datos se relacionan unos con otros mediante punteros.

* Estáticas: su tamaño ocupado en memoria se define con anterioridad a la ejecución del programa y durante este no puede modificarse. Las estructuras estáticas siempre se representan en la memoria de forma contigua.
* Dinámicas: pueden crecer o decrecer durante la ejecución, adquiriendo o liberando posiciones de memoria y la única limitación respecto a tamaño es la memoria de la computadora; son representadas con la ayuda de punteros.

**Árbol:** es una ED no lineal de estructura jerárquica en la que cada nodo puede apuntar a uno o más nodos. Mismos tipos de nodo que en árbol binario.

**Árbol binario:** Es un tipo especial de árbol en el que de cada nodo solo pueden colgar 2 subárboles. Es una estructura de datos no lineal jerárquica que consiste en un campo para un dato entero y dos punteros que indican la posición de sus hijos. Cumple que el subárbol izquierdo de cada nodo, si no está vacío, contiene un valor menor a dicho nodo, y el subárbol derecho, si no está vacío, contiene uno mayor.

Se tienen varios tipos de nodos: hijo (cualquiera de los nodos apuntado por uno de los nodos del árbol), padre (nodo que apunta a otro nodo), raíz (nodo que no tiene padre), y hoja (nodo que no tienen hijos).

Existen distintas formas de recorrerlos: Pre-orden (el nodo raíz se visita a lo primero), In-orden (el nodo raíz está en el medio del recorrido) y Post-orden (el nodo raíz se visita a lo último).

**Listas:** es un conjunto de datos del mismo tipo (simple o estructurado) más un puntero. En las listas simplemente enlazadas, cada nodo tiene un único predecesor (excepto el primero) y un único sucesor (excepto el último). Es una ED dinámica, por lo que su número de elementos es variable. También hay un puntero externo a la lista que apunta al primer elemento de la misma llamado cabecera. Y el último elemento apunta a NULL. Las listas son estructuras autoreferenciadas, es decir que dentro de su definición aparece la misma estructura. Tipos de lista:

* Pila:Es una estructura lineal a cuyos datos sólo se puedeacceder por un solo extremo, denominado tope o cima.Sólo se pueden efectuar dos operaciones: apilar y desapilar. Si se meten varios elementos en la pila y después se sacan de ésta, el último elemento en entrar será el primero en salir. [LIFO (last in firstout)] Lo que varía es solo la posición del tope de la pila.
* Cola: Es una estructura lineal a cuyos datos entran por la partede atrás y salen por la de adelante.Se realizan de encolar y desencolar un elemento.Si se meten varios elementos en la cola y después sesacan de ésta, el primer elemento en entrar será elprimero en salir. [FIFO (first in firstout)]

**Punteros:** Un puntero es una variable que contiene la dirección de memoria de un dato o de otra variable que contiene al dato en un arreglo. Esto quiere decir, que el puntero apunta al espacio físico donde está el dato o la variable. Un puntero puede apuntar a un objeto de cualquier tipo, como por ejemplo, a una estructura o una función. Como cualquier variable, un puntero está almacenado en una dirección de la memoria, que además es distinta de aquella a la que apunta. Pasajes: por valor trabaja con una copia y pasa el VALOR en la variable y por referencia pasa la DIRECCION de memoria, accediendo directamente al dato. Uso doble puntero cuando quiero modificar el dato y simple si NO quiero modificarlo.

**Asignación dinámica de memoria:** lo hacemos utilizando la función malloc (memory allocation). Ésta devuelve una dirección de memoria libre, si no hay memoria suficiente duevuelve NULL.

**Crear una** **estructura** alumno para almacenar: nombre, apellido, fecha de ingreso (estructura del tipo fecha cuyos campos son día, mes año), cantidad de materias aprobadas y promedio.

typedef struct alumno

{

char nombre[30];

char apellido[30];

fecha ingreso;

int aprobadas;

float promedio;

}alumno;

**Cargar estructura**

void cargar

{

alumno aux;

printf(“Ingrese el nombre”);

scanf(“%s”, &aux->nombre);

printf(“Ingrese el apellido”);

scanf(“%s”, &aux->apellido);

printf(“Ingrese la fecha de ingreso (d-m-a)”);

scanf(“%d-%d-%d”, &aux->ingreso->dia, &aux->ingreso->mes, &aux->ingreso->año);

printf(“Ingrese la cdad de materias aprobadas”);

scanf(“%d”, &aux->aprobadas);

printf(“Ingrese el promedio”);

scanf(“%f”, &aux->promedio);

}

Sobre **punteros**:

int a, \*ptr;

a=7;

ptr=&a;

int \*b[2];

int \*ptr2;

int c[2] = {1, 2, 3};

ptr2=&c[0];

Imprimir <&a>, <ptr>, <\*&ptr> da la dirección de memoria de a y <a>, <\*ptr> muestran “7” (el contenido de a). Y b es un array de 3 punteros a int. Si se le suma 1 a ptr2 (\*(ptr2+1);) apuntará a la siguiente dirección de memoria, es decir a c[1] ya que los vectores almacenan sus datos en direcciones de memoria contiguas. Si se hace ptr2++ apunta al mismo lugar.

int x[100],b,\*pa,\*pb;

//...

x[50]=10;

pa=&x[50];

b = \*pa+1; *//Esto es como decir el valor que tiene el array de x[50] sumarle 1.*

*//Esto es igual a: b=x[50]+1; => Su valor sería igual a 11.*

b = \*(pa+1); *//Esto primero pasa a la siguiente dirección de memoria y luego lo referencia*

*//El resultado es: b = x[51];*

pb = &x[10]; *//al puntero pb se le asigna la dirección de x[10]*

\*pb = 0; *//Al valor que tiene el puntero se le asigna 0*

*//Esto es igual que decir: x[10] = 0*

\*pb += 2; *//El valor del puntero se incrementa en dos unidades, es decir x[10] += 2*

(\*pb)--; *//El valor del puntero se decrementa en una unidad.*

x[0] = \*pb--; *//A x[0] se le pasa el valor de x[10] y el puntero pb, pasa a apuntar a x[9]*

*//recuerda, que -- es post-decremento, primero asignará y luego restará.*

**Mostrar un registro leído**

(fread(&reg, sizeof(producto),1,pf); lee el reg)

printf(“Código: %d Stock: %d Borrado: %d\n”, reg.codigo, reg.stock, reg.borrado);