	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 1 de 12



TITULO: INVESTIGACIÓN.

PROGRAMA EDUCATIVO: LENGUAJE DE PROGRAMACIÓN.

DOCENTE: JOSÉ LUIS LIRA TURRIZA.

INTEGRANTES:


COL CAUICH ROSA ITZEL-5674

ESCOBAR TURRIZA JUAN DAVID SALVADOR-5678

GONZALEZ MENDEZ LUIS ENRIQUE

MARTINEZ SALAZAR EDGAR ALBERTO-7116

FECHA DE ENTREGA: 30 DE ABRIL DEL 2020

	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 2 de 12

En el presente ensayo se hablará acerca de un tema muy interesante en la asignatura de lenguaje de programación es sobre los diferentes tipos de estructura de datos, estos tienen varias aplicaciones muy interesantes haciendo referencia al lenguaje de C++ y son muy importantes en los sistemas de computadora por su caracterización en la organización y en diferentes operaciones

Se va a abordar de manera concisa cada tipo de estructura acompañado de un ejemplo para concretar y explicar lo esencial de cada tipo, las referencias se encontrarán al final para consultar más información.

Un arreglo también conocido como matriz, vector o array, es un conjunto limitado y ordenado de elementos iguales. Consta de una propiedad “ordenado” ya que cada elemento puede ser identificado sin ningún problema de manera directa mediante un índice o posición.

Para crear un arreglo en C ++, debe cumplir, el tipo de elemento (por ejemplo, int, char, double, bool o tipo definido por el programador), el nombre del arreglo y para crear una matriz unidimensional, el formato es el siguiente:

<Tipo de elemento> <Nombre de matriz> [<Tamaño de primera dimensión>]

Ejemplo:

```
/*Escribe un programa que defina un vector de números y calcule la suma de sus elementos.*/
```


```
#include<iostream>
```

```
#include<conio.h>
```

```
using namespace std;
```

```
int main(){
```

```
    int numeros[5] ={1,2,3,4,5};
```

	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 3 de 12

```

int suma=0;


for(int i=0;i<5;i++){
    suma += numeros[i]; //Utilizamos una suma iterativa
}

cout<<"La suma de los elementos del arreglo es: "<<suma<<endl;
getch();
return 0;
}

```

Como se puede observar en el ejemplo se creó una suma de elementos de un arreglo con los números que se querían ya que el ejercicio no especificaba.

Las pilas son unas estructuras de datos que trabajan la información de una manera especial dando a entender a una estructura de datos como una lista, dando a entender que al ingresar la información se llevara una pila donde el primer dato ingresado se direccionara a la parte inferior y consecuentemente al ingreso de datos se llevara a cabo la asignación de una manera ascendente de esta manera hasta el último elemento correspondiente, cuando se quiera obtener la información derivada de la pila no se podrá acceder a ningún dato con excepción al último, ya que para comenzar se debe utilizar el primer dato y así consecuentemente con los datos adquiridos, a las pilas se les define como datos tomando en cuenta el último en llegar y el primero en salir, de esta manera son utilizados para analizadores de léxico y expresiones por ejemplo una formula matemática, estos datos se almacenan en una pila donde uno puede llamar una función y este mismo almacena los datos de la misma y hace una función derivada del segundo dato ingresado en el mismo, a la hora de liberar los espacios en la función se hace uso de la primera función que se tomó en cuenta en la captura de los datos, correspondiente a ello en la cola el primer dato que llega es el primero que se procesa, y su proceso es consecutivo,

	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 4 de 12


esto se usa en la cola de impresiones o llamadas de servidores, atender peticiones de base de datos y transacciones, por ejemplo si hay cinco personas conectadas a los atm esta misma entregara resultados a la primera persona que llevo a cabo la conexión y así consecuentemente.

Ejemplo pila:

```
{
Pila p,t;
Int dato, opc, elemento, flag0;
p.tope=0;
do
{
clrscr();
printf("\nMENU-PILA");
printf("\ninsertar elemento");
printf("\nEliminar elemento");
```

Ejemplo cola:

```
class Cola_Lista{
    class Nodo{
        String nombre;
        Nodo enlace;
        Nodo (String n){
            nombre=n;
            enlace = null;
        }
    }
    Nodo frente;
    Nodo fin;
    Cola_Lista(){
        frente=null;
        fin=null;
    }
    public boolean vacia(){
        return(frente==null);
    }
    public boolean llena(){
        return false;
    }
}
```

	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 5 de 12

Se le conoce como lista lineal a un grupo de elementos de un tipo de dato, este puede variar en número. Los elementos pueden tener únicamente un predecesor (antes) y un sucesor (después), exceptuando al primero y al último de la lista.

La forma más común en que se almacenan los datos de una lista lineal es uno dentro de otro, es decir, continuos, de lugares consecutivos dentro de la memoria.


Se le denomina almacenamiento secuencial a la manera en que se guarda una lista lineal, ya sea en una memoria de computadora, donde se almacena en la memoria principal en posiciones sucesivas. O en una cinta magnética, en la que los datos se encuentran sucesivos en la cinta.

Hay distintas operaciones que se pueden hacer con las listas lineales, por ejemplo: insertar, eliminar o localizar un elemento; determinar el tamaño (número de elementos) de la lista; recorrer la lista para localizar un determinado elemento; clasificar los elementos de la lista en orden ascendente o descendente; unir dos o más listas en una sola; dividir una lista en varias sublistas; copiar la lista; borrar la lista.

Una lista lineal se procesa como un array unidimensional. El acceso a algún elemento de la lista y la adicción de nuevos elementos es sencillo; por otra parte, para la inserción o eliminación se requiere de un desplazamiento de lugar de los elementos siguientes, es decir, el diseño de algoritmo. A los arrays hay que darles un tamaño suficiente para contener los posibles elementos de la lista, esto para poder realizar operaciones con listas como array.

Ahora, para comprender de mejor manera, se presenta un ejemplo a continuación.

El algoritmo requiere conocer el número de elementos de la lista (su longitud, L). Los pasos para dar son:

	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 6 de 12

1. conocer longitud de la lista L.

2. si $L = 0$ visualizar «error lista vacía».

si_no comprobar si el elemento j-ésimo está dentro del rango permitido de elementos $1 \leq j \leq L$; en este caso, asignar el valor del elemento $P(j)$ a una variable B; si el elemento j-ésimo no está dentro del rango, visualizar un mensaje de error «elemento solicitado no existe en la lista».

3. fin.

El pseudocódigo correspondiente sería:

procedimiento acceso(E lista: P; S elementolista: B; E entero: L, J)

inicio

 si $L = 0$ entonces

 escribir('Lista vacia')

 si_no

 si $(j \geq 1) \text{ y } (j \leq L)$ entonces

$B \leftarrow P[j]$


 si_no

 escribir('ERROR: elemento no existente')

 fin_si

fin_si

fin

	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 7 de 12

Por otra parte, se le conoce como lista ligada a un conjunto de estructuras, a las que se les llama nodos, éstas están conectadas por medio de ligas, de allí el nombre lista ligada. Para ingresar al primer nodo de una lista es necesario de un apuntador. El acceso a los nodos subsiguientes es por medio del miembro liga almacenado en cada uno de los nodos. El final de una lista se señala con el apuntador liga del ultimo nodo, el cual se establece en NULL.


“Un arreglo puede declararse para que contenga más elementos que los esperados; sin embargo, esto puede desperdiciar memoria. Las listas ligadas proporcionan una mejor utilización de memoria, en estas situaciones.” (Deitel, 2004)

Existen dos tipos de estructura de datos, las lineales y las no lineales. Las vistas anteriormente eran las lineales, a continuación, se hablará de una no lineal.

La conocida como estructura de árbol es una no lineal de dos dimensiones, que además tiene propiedades especiales.

Hay árboles que pueden tener nodos con hasta dos ligas, de las cuales una, dos o ninguna pueden ser NULL.

El primer nodo del árbol recibe el nombre de nodo raíz, cada liga de este nodo hara referencia a un “hijo”. Al primer nodo del subárbol izquierdo se le conoce como hijo izquierdo. De igual forma pasa con el lado derecho, el hijo derecho es el primer nodo del subárbol derecho. El termino hermanos se aplica a los hijos de un nodo. Si un nodo no tiene hijos se le llamara nodo hoja.

	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 8 de 12

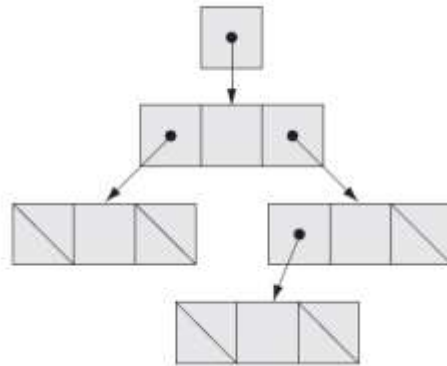


Ilustración 1 Representación gráfica de un árbol binario.


Ahora, para comprender de mejor manera, se presenta un ejemplo a continuación.

El programa genera 10 números aleatorios e inserta cada uno en el árbol, con excepción de los valores duplicados.

```

1  /* Figura 12.19: fig12_19.c
2     Crea un árbol binario y lo recorre en
3     preorden, inorden, y en postorden */
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <time.h>
7
8  /* estructura autorreferenciada */
9  struct nodoArbol {
10     struct nodoArbol *ptrIzq; /* apuntador al subárbol izquierdo */
11     int dato; /* valor del nodo */
12     struct nodoArbol *ptrDer; /* apuntador al subárbol derecho */
13 }; /* fin de la estructura nodoArbol */
14
15 typedef struct nodoArbol NodoArbol; /* sinónimo de la estructura nodoArbol */
16 typedef NodoArbol *ptrNodoArbol; /* sinónimo de NodoArbol* */
17
18 /* prototipos */
19 void insertaNodo( ptrNodoArbol *ptrArbol, int valor );
20 void inOrden( ptrNodoArbol ptrArbol );
21 void preOrden( ptrNodoArbol ptrArbol );
22 void postOrden( ptrNodoArbol ptrArbol );
23
24 /* la función main comienza la ejecución del programa */
25 int main()


```


	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 9 de 12

```

26 {
27     int i; /* contador para el ciclo de 1 a 10 */
28     int elemento; /* variable para almacenar valores al azar */
29     ptrNodoArbol ptrRaiz = NULL; /* árbol inicialmente vacío */
30
31     srand( time( NULL ) );
32     printf( "Los numeros colocados en el arbol son:\n" );
33
34     /* inserta valores al azar entre 1 y 15 en el árbol */
35     for ( i = 1; i <= 10; i++ ) {
36         elemento = rand() % 15;
37         printf( "%3d", elemento );
38         insertaNodo( &ptrRaiz, elemento );
39     } /* fin de for */
40
41     /* recorre el árbol en preorden */
42     printf( "\n\nEl recorrido en preorden es:\n" );
43     preOrden( ptrRaiz );
44
45     /* recorre el árbol en inorden */
46     printf( "\n\nEl recorrido inorden es:\n" );
47     inOrden( ptrRaiz );
48
49     /* recorre el árbol en posorden */
50     printf( "\n\nEl recorrido en posorden es:\n" );
51     posOrden( ptrRaiz );
52
53     return 0; /* indica terminación exitosa */
54
55 } /* fin de main */
56
57 /* inserta un nodo dentro del árbol */
58 void insertaNodo( ptrNodoArbol *ptrArbol, int valor )
59 {
60
61     /* si el árbol está vacío */
62     if ( *ptrArbol == NULL ) {
63         *ptrArbol = malloc( sizeof( NodoArbol ) );
64
65         /* si la memoria está asignada, entonces asigna el dato */
66         if ( *ptrArbol != NULL ) {
67             ( *ptrArbol )->dato = valor;
68             ( *ptrArbol )->ptrIzq = NULL;
69             ( *ptrArbol )->ptrDer = NULL;
70         } /* fin de if */
71         else {
72             printf( "no se inserto %d. No hay memoria disponible.\n", valor );
73         } /* fin de else */
74
75     } /* fin de if */
76     else { /* el árbol no está vacío */
77
78         /* el dato a insertar es menor que el dato en el nodo actual */
79         if ( valor < ( *ptrArbol )->dato ) {
80             insertaNodo( &( ( *ptrArbol )->ptrIzq ), valor );


```

	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 10 de 12

```

81     } /* fin de if */
82
83     /* el dato a insertar es mayor que el dato en el nodo actual */
84     else if ( valor > ( *ptrArbol )->dato ) {
85         insertaNodo( &(amp; ( *ptrArbol )->pntDer ), valor );
86     } /* fin de else if */
87     else { /* ignora el valor duplicado del dato */
88         printf( "dup" );
89     } /* fin de else */
90
91 } /* fin de else */
92
93 } /* fin de la función insertaNodo */
94
95 /* comienza el recorrido inorden del árbol */
96 void inOrden( ptrNodoArbol ptrArbol )
97 {
98
99     /* si el árbol no está vacío, entonces recórralo */
100    if ( ptrArbol != NULL ) {
101        inOrden( ptrArbol->pntIzq );
102        printf( "%3d", ptrArbol->dato );
103        inOrden( ptrArbol->pntDer );
104    } /* fin de if */
105
106 } /* fin de la función inOrden */
107
108 /* comienza el recorrido preorden del árbol */
109 void preOrden( ptrNodoArbol ptrArbol )
110 {
111
112     /* si el árbol no está vacío, entonces recórralo */
113     if ( ptrArbol != NULL ) {
114         printf( "%3d", ptrArbol->dato );
115         preOrden( ptrArbol->pntIzq );
116         preOrden( ptrArbol->pntDer );
117     } /* fin de if */
118
119 } /* fin de la función preOrden */
120
121 /* comienza el recorrido postorden del árbol */
122 void postOrden( ptrNodoArbol ptrArbol )
123 {
124
125     /* si el árbol no está vacío, entonces recórralo */
126     if ( ptrArbol != NULL ) {
127         postOrden( ptrArbol->pntIzq );
128         postOrden( ptrArbol->pntDer );
129         printf( "%3d", ptrArbol->dato );
130     } /* fin de if */
131
132 } /* fin de la función posOrden */


```

	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 11 de 12

Imágenes recuperadas del libro “Como programar C/C++ y java de Harvey M. Deitel y Paul J. Deitel

“No establecer en NULL las ligas de los nodos hoja de un árbol, puede ocasionar errores de ejecución.” (Deitel, 2004)

Como se ha visto en el trabajo sobre los tipos de estructuras de datos que se usan en la programación de lenguaje C++, son variados y cada uno tiene características diferentes por lo cual su uso va dependiendo de lo que el usuario quiera desarrollar. Por ejemplo, la estructura de árbol tiene como objetivo agrupar los nodos jerárquicamente de los cuales se distingue uno que es la raíz y en consecuencia se van desplazando los demás nodos así también se puede hacer un recorrido para manipular la información (bastante útil si quieres ordenar con prioridad y después buscar información para después modificarla). En cambio, la clase Lista, se puede obtener un elemento destacado, crearla vacía e irle agregando elementos, eliminar elementos de cualquier posición eh incluso mostrar todos los elementos de la lista. Todo lo contrario, sucede con la estructura de Pilas, porque solo se puede eliminar o insertar elementos desde el extremo de la estructura que habitualmente se le llama cima, ósea que si quieres modificar el inicio de tu programa tendrás que ir modificando desde el extremo hasta llegar a la zona que desees modificar. (este tipo de estructura es lineal). La estructura Cola como su nombre lo dice se van insertando datos y el borrado se hace desde el principio de esta por eso el nombre. Cada estructura tiene sus aplicaciones y características estas las distinguen una de otra, el estudiante sabrá cuando es necesario usar cada tipo de estructura para facilitar su trabajo.

	DIRECCIÓN ACADÉMICA	Código:
		Revisión: 3
	INVESTIGACIÓN	Vigente desde: 14-09-2016
		Página: 12 de 12

Bibliografía

Aguilar, L. J. (2008). *Fundamentos de programación, cuarta edición*. Aravaca (Madrid): McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S. A. U.

AHO, A. V., HOPCROFT, J. E., & ULLMAN, J. D. (1998). *Estructuras de datos y algoritmos*. México: Addison Wesley.

BRASSARD, G., & BRATLEY, P. (1997). *Fundamentos de Algoritmia*. Madrid: Prentice-Hall.

Castell, R. F. (s.f.). *Algoritmo y estructura de datos 1*. España: Valencia: universidad de valencia.

COLLADO MACHUCA, M., MORALES FERNÁNDEZ, R., & MORENO NAVARRO, J. J. (1987). *Estructuras de datos. Realización en Pascal*. Madrid: Ediciones Díaz de Santos.

Deitel, H. M. (2004). *Como programar en C/C++ y java (4° ed.)*. México: PERSON EDUCACIÓN.

Flores, L. I. (2008). *Manual de Programación en Lenguaje C++*. Obtenido de <https://paginas.matem.unam.mx/pderbf/images/mprogintc++.pdf>

GARCÍA MOLINA, J. J., MONTOYA DATO, F. J., FERNÁNDEZ ALEMÁN, J. L., & MAJADO ROSALES, M. J. (2005). *Una introducción a la programación. Un enfoque algorítmico*. Madrid: Thomson-Paraninfo.

Velázquez, J. L. (Octubre de 2010). *Lenguaje de Programación: C++*. Obtenido de Arreglos: https://www.cimat.mx/~pepe/cursos/lenguaje_2010/slides/slide_31.pdf