Nombre: Gerson Ronaldo Ortiz Marin_____**No. de Matrícula.: _20171176**_____

Materia: _____Fundamentos de Programacion_____ _____ Grupo**:** _____ Turno**:** _Matutno__

Carrera: ___Desarrollo de softwere_____

Tema: __Tema 17_____ **No: inv 5** ____

Fecha propuesta: _____ **Fecha de Entrega:** _____

Escuela: ___**Amerike**_____ **Plantel** ___**Guadalajara**_____

Calle: ___**Montermorelos**_____ **No:** ___3503___ **Colonia:** _Rinconada de la calma_____ **C.P.:** __45080__

Teléfono: _____ **Ciudad:** _____

RonaldoO.

_____ _____

Firma del alumno (a) **Firma de revisión fecha**

Qué se evalúa:	10 pts.	7 pts.	4 pts.	
Entrega electrónica	Es en tiempo y forma al iniciar la clase. (1 pts.)	Después de 20 minutos de iniciada la clase. (.7 pts.)	Al minuto 30. (Posteriormente ya no se reciben). (.4pts.)	
Del formato.	Cumple con todos los elementos solicitados. (1 pts.)	No cumple con dos elementos solicitados. (.7 pts.)	No cumple con tres o más elementos solicitados. (.4pts.)	
La ortografía.	Tiene dos errores ortográficos. (1 pts.)	Tiene de tres a cuatro errores ortográficos. (.7 pts.)	Tiene cinco o más errores ortográficos. (.4pts.)	
Del tema.	La teoría y ejemplos corresponden al tema tratado. (1 pts.)	La teoría o ejemplos no corresponden al tema tratado. (.7 pts.)	La teoría y ejemplos no corresponden al tema tratado. (.4pts.)	
El resumen.	Es congruente con el (los) tema (s) y aporta conceptos propios del alumno. (1.5pts.)	Es congruente con el (los) tema (s) y no aporta conceptos propios del alumno. (1 pts.)	No es congruente con el (los) tema (s) y no aporta conceptos propios del alumno. (.4pts.)	
Conocimientos.	Responde acertadamente las preguntas del tema tratado que se le formulan oralmente. (1.5pts.)	Responde acertadamente algunas preguntas del tema tratado que se le formulan oralmente. (.7 pts.)	No responde acertadamente las preguntas del tema tratado que se le formulan oralmente. (.4 pts.)	
Las preguntas.	Todas las preguntas formuladas son acordes con su nivel de estudio, cuentan con cálculos matemáticos y su respectiva respuesta. (1 pts.)	Una o dos preguntas formuladas no son acordes con su nivel de estudio o no cuentan con cálculos matemáticos, o su respectiva respuesta. (.7 pts.)	Tres o más preguntas formuladas no son acordes con su nivel de estudio o no cuentan con cálculos matemáticos o su respectiva respuesta. (.4pts.)	
Presentación y archivo electrónico.	Es congruente con el (los) tema (s) presenta una secuencia lógica y no tiene más de dos errores ortográficos. (1 pts.)	Es congruente con el (los) tema (s) presenta una secuencia lógica y no tiene más de tres a cuatro errores ortográficos. (.8 pts.)	No es congruente con el (los) tema (s) no presenta una secuencia lógica y tiene más de cinco errores ortográficos. (.4pts.)	
Bibliografía.	Es acorde al (los) tema (s) tratado (s) y está completa (.7 pts.)	Es acorde a algún (os) tema (s) tratado (s), le falta algún elemento que la conforman (.7 pts.)	No es acorde al (los) tema (s) tratado (s), le faltan 2 elementos que la conforma (.4pts.)	
Fuentes de consulta.	Es acorde al (los) tema (s) tratado (s) (.3 pts.)	Es acorde a algún (os) tema (s) tratado (s) (.3 pts.)	Es acorde a algún (os) tema (s) tratado (s) (.4 pts.)	

Índice:

Contenido del tema:

Listas en programación:

Es una estructura de datos muy importante ya que representa una colección de elementos ordenados, los elementos pueden ser repetidos, cada elemento de la lista tiene un indice que lo ubica de la misma.

Operaciones:

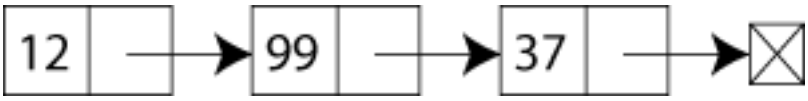
En general las listas proveen operaciones como, construir una lista, obtener el tamaño de la lista, verificar si está vacía, obtener el primer elemento de la lista llamado head o cabeza, agregar un nuevo elemento para un indice dado etc.

Ejemplo de una lista con seudónimo código:

tipo abstracto Lista<T>
operación crearLista() -> Lista
operación tamaño(Lista lista) - > int
operación vacia(lista lista) - > Boolean
operación cabeza(Lista<T> lista) - > T
operación agregar (lista<T> lista, T elemento)
operación elementoEn (lista<T> lista, una indice) - > T
operación removerElementoEn(lista lista, int indice)

Lista Enlazada:

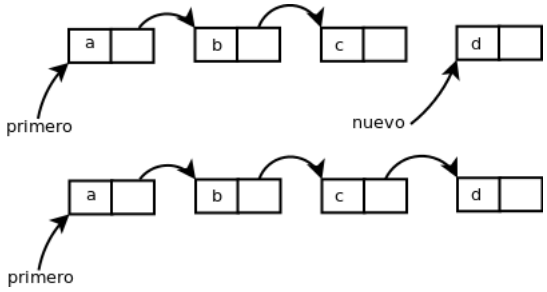
En esta lista es donde se representa internamente como nodos que referencias al siguiente



Así la lista tiene una referencia al primer nodo (salvo que esté vacía)
Cada nodo tiene el dato en la posición y una referencia al siguiente nodo.

En las listas enlazadas, agregar un elemento es fácil, o poco costoso en términos de alocución de memoria.
Simplemente debemos de colocar espacio para un nuevo nodo, y actualizar el ultimo nodo para que apunte de nuevo, como “siguiente”.

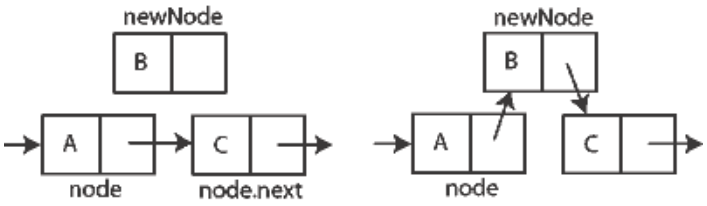
Por el contrario en una lista basada en array, agregar un elemento que involucre redimensionar el array podría ser costoso a nivel de memoria.



Como agregar un electo es necesario primero ubicar el ultimo nodo, recorrer todos los nodos hasta llegar al final esto hace que sea una operación con complejidad lineal, ya que entre más elementos tenga la lista mas tiempo va a demorar.

Por esto es que en general se implementa una variante, donde la lista conoce el primer y ultimo nodo

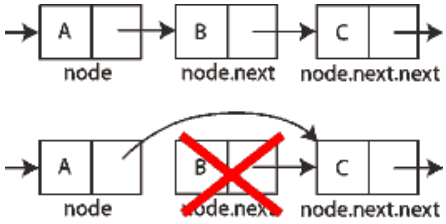
Agregar un elemento en medio, esta operación requiere la misma cantidad de Memoria, es decir un nuevo nodo. Lo que debemos hacer es posicionarnos en el nodo anterior y “meter” el nuevo nodo, actualizando las referencias del anterior al nuevo y del nuevo al que previamente era el siguiente.



Remoción de un elemento.

Al igual que el agregar, el remover un ítem es también simple en términos operativo. Se libera el nodo pero primero se “atan” entre si los nodos anterior y siguiente:

No se requiere re-ordenar o re-colocar memoria.



Problemas o desventajas.

La lista enlazada tiene las siguientes desventajas:

Acceso aleatorio: Si necesitamos acceder a una posición específica, digamos el Ítem con indice 23, la estructura no posee una forma de acceder directamente a este elemento por lo cual deber ir recorriendo todos los nodos hasta llegar al 23.

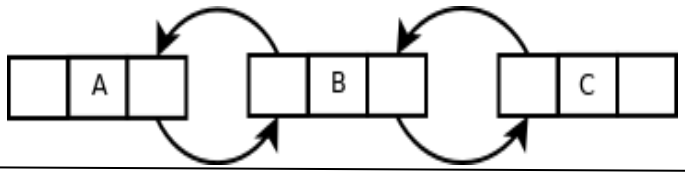
Localidad: La localidad espacial en la memoria es “mala” ya que los nodos pueden estar dispersos y “alejados” en la memoria.

Recorrido reverso: Por consecuencia del hecho que el acceso aleatorio es “malo” también sucederá que en mi programa necesito recorrer la lista en sentido contrario, esto va a ser muy eficiente, por que cada elemento va a recorrer los nodos.

Lista Doblemente Enlazada:

Para resolver el problema de recorridos inversos / reverso, existe una variante a la lista enlazada simple.
La idea es que cada nodo ademas de conocer al siguiente, conozca al anterior.

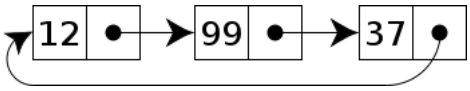
Es un poco mas costoso a nivel de complejidad de código, por que ahora vamos a tener que mantener un puntero mas y en cada operación como agregar o eliminar etc… deberemos actualizarlo y por otro lado, también requería más memoria (un puntero mas). Pero hará mas recorrido inverso(pasamos de complejidad sum(0..N) a O(N))



Lista Circular:

Otra variante de la lista enlazada es aquella en la que el ultimo electo conoce al primero y viceversa si es doblemente enlazada. De ahí su nombre “circular” ya que se puede representar como un circulo de nodos.

Obviamente esta lista no tiene fin.

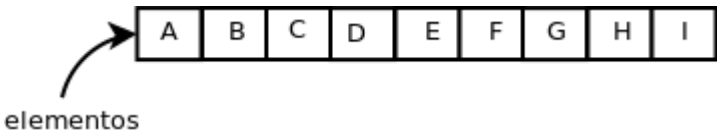


Lista basada en array:

Esta implementación si que es bastante diferente a la enlazada (a diferencia de la doblemente enlazada y la circular). Justo a la enlazada, son las dos implementaciones mas comunes en los lenguajes de programación.

La idea es bien simple:

- Tendremos un tipo de lista(struct en C, clase en Python)
- Este almacenara los datos internamente como un array.
- En C tendremos un puntero.



Agregar un elemento.

Adicionar un elemento a la lista de tipo array tiene varios problemas:

- Por un lado es costoso a nivel memoria, ya que debemos hacer crecer el array una posición, pidiendo memoria = cantidadActual +1.
- Por otro lado también podrá ser costoso a nivel de tiempo, porque se deben de copiar los elementos a la nueva posición (en caso de re-colocación).

Como ya vimos, a veces el sistema operativo no nos puede dar la posición siguiente de memoria, por lo que tiene que buscar otro sector de memoria, por lo que tiene que buscar otro sector de memoria que tenga la nueva capacidad y luego copiar los elementos.

Resumen:

Las listas es una forma de guardar datos en sectores acomodados en los cuales podremos tener acceso para poder moverlos, agregar o eliminar.

Fuentes de consulta:
https://sites.google.com/site/programacioniiuno/temario/unidad-3---estructuras-de-datos-comunes-y-colecciones/la-estructura-lista