ESTRUCTURA DE DATOS

Muchos algoritmos requieren una representación apropiada de los datos para lograr ser eficientes.

Esta representación junto con las OPERACIONES permitidas se llama ESTRUCTURA DE DATOS.
Las estructuras de datos varían, fundamentalmente, según como permitan el acceso a los datos.
Algunas permiten operaciones de borrado, otras permiten acceso sólo al elemento más reciente insertado, etc.
Algunas de las estructuras de datos más comunoes son:

Listas en general	Pilas	Colas
Listas enlazadas	Árboles	Árboles binarios
Tablas hash,	etc.	

Definición.

Una estructura de datos es una representación de datos junto con las operaciones permitidas sobre dichos datos.

Los estructuras de datos nos permiten lograr un importante objetivo de la programación orientada a objetos: la reutilización de componentes.
Esto significa que, una vez implementada, puede ser utilizada una y otra vez.

La implementación es parte del paradigma de la programación orientada a objetos: el usuario de la estructura de datos no necesita ver la implementación, sólo las operaciones disponibles.

Listas general.

Diefiniciones.

Una lista L una secuencia de elementos: $a_1, a_2, a_3, ... a_{n-1}, a_n$
con n >= 0
El tamaño de L es n.

Existe la lista vacía o lista nula y su tamaño es 0

Para cualquier lista, excepto la lista nula, se dice que
a_{i+1} es sucesor de a_i con $1 <= i < n$
a_{i-1} es predecesor de a_i con $1 < i <= n$
a_1 es el primer elemento de la lista
a_n es el último elemento de la lista

Asociado a estas definiciones existe un conjunto de operaciones que nos permiten manipular los elementos de una lista.

Ejemplo de algunas operaciones más comunes.

Anula(L), anular una lista. Hacer que la lista L quede vacía. Retorna la lista L vacía.

Vacía (L), determinar si la lista está o no vacía. Retorna verdadero o falso según el caso.

Vaciar(L), sacar los elementos de la lista uno a uno hasta dejar la lista vacía.

Primero(L), retorna el primer elemento de la lista.

Ultimo(L), retorna el último elemento de la lista.

Imprimir(L), imprime todos los elementos de la lista en un orden especificado.

Casilla N° 23 A	Casilla N° 24 B	Casilla N° 25 C	Casilla N° 26 D	Casilla N° 27 E	Casilla N° 28 F	Casilla N° 29 G	Casilla N° 30 H

Casillas del correo de Temuco

Definición:

Ubicación de un elemento en la lista: Número entero que indica el número de orden del elemento en la lista.

Posición de un elemento en la lista: Número entero que indica el lugar físico de en elemento en la lista

El elemento B esta la en la ubicación 2, es decir ocupa el segundo lugar en la lista.

El elemento B está en la posición 24, es decir la casilla rotulada N° 24

Otras funciones comunes para el manejo de listas.

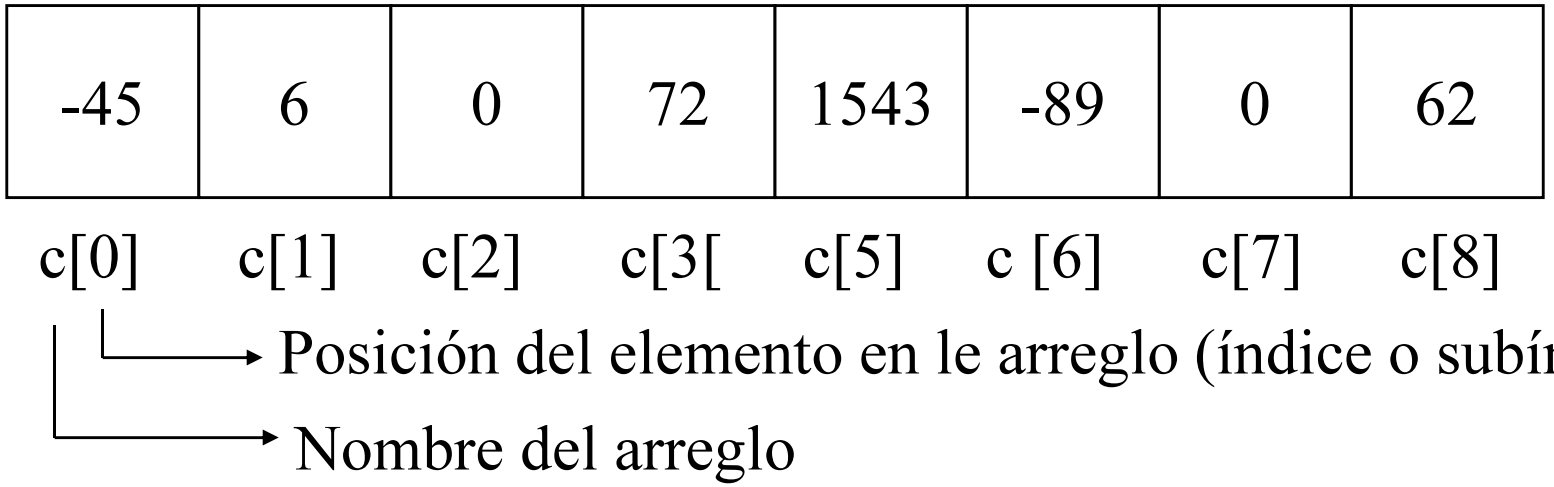
Dada una lista L, una posición p de L y un elemento x.

Inserta(x,p,L), agrega x a la lista L en la posición p.
Suprime(p,L), elimina el elemento de la lista L que está en la posición p.
Localiza(x,L), retorna la posición de la primera ocurrencia de x en la lista L.
Recupera(p,L), retorna el elemento que está en la posición p de la lista L.
Siguiente(p,L), retorna la posición siguiente de p en L.
Anterior(p,L), retorna la posición anterior de p en L.

En general la interpretación de qué es adecuado para una función es decisión del programador. Lo mismo ocurre con el manejo de ciertos casos espaciales.

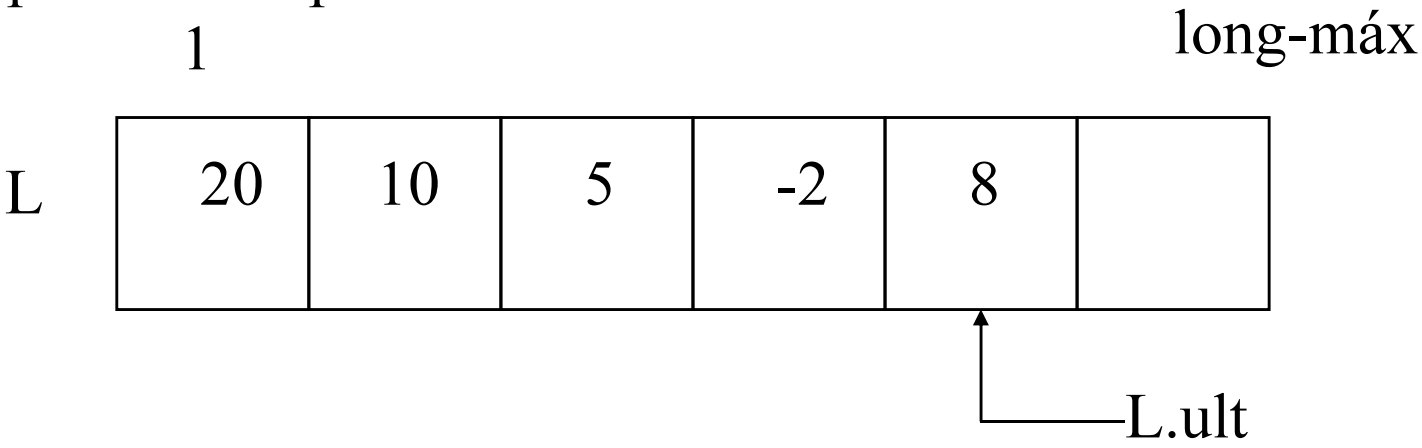
Implementación de listas con arreglos.

Un arreglo es un grupo consecutivo de localidades de memoria que tienen el mismo nombre y el mismo tipo.
Para hacer referencia a un elemento de un arreglo, especificamos el nombre del arreglo y la posición del elemento en el arreglo.



En general, la referencia al elemento i de un arreglo c es c[i – 1]
El índice debe ser entero o una expresión entera
Ej. Si a = 3 y b = 4, c[a+b] +=2 suma 2 al elemento c[7]

En el caso particular de un arreglo los términos ubicación y posición se pueden confundir.



El elemento 10 se encuentra en la ubicación 2 y en la posición 2.

Este caso puede ocurrir si, por ejemplo, el arreglo comienza con el índice 1.

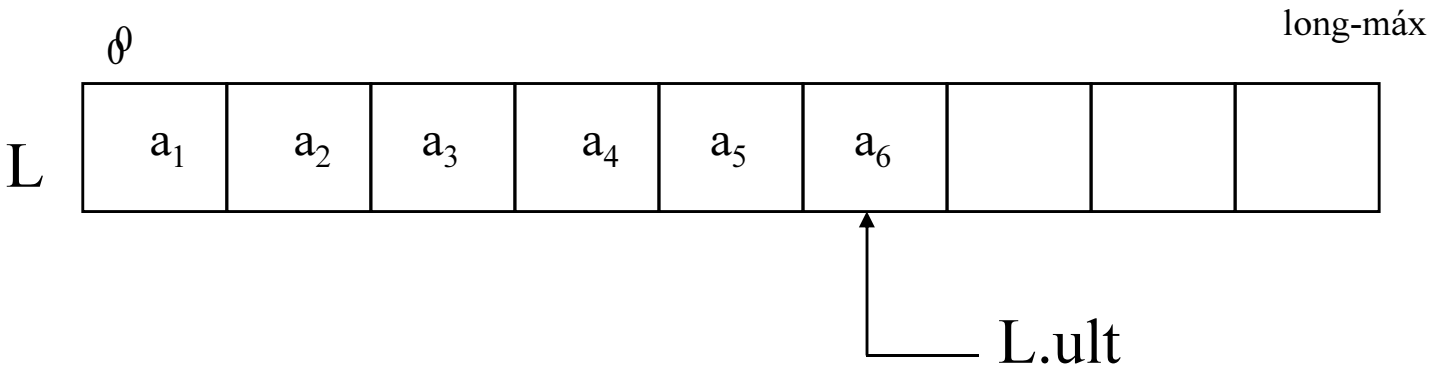
Declaración de arreglos:

tipo nombreAreglo [tamañoArreglo];

```
int L[8];    // ordena al compilador que reserve 8 elementos para
             // para un arreglo de enteros llamado L
```

```
int main()
        {
          int L[ 10 ];  // L es un arreglo de 10 enteros

          // inicializa los elementos del arreglo L en 0
          for ( int i = 0; i < 10; i++ )
            L[ i ] = 0;   // establece el elemento de la ubicación i en 0

          cout << "Elemento" << setw( 13 ) << "Valor" << endl;


          // despliega el contenido de un arreglo L en dos columnas
          for ( int j = 0; j < 10; j++ )
            cout << setw( 7 ) << j << setw( 13 ) << L[ j ] << endl;

          return 0;  // indica terminación exitosa

        } // fin de main
```

Formato de una lista implementada en arreglos.



Algunas consideraciones para la implementación de operaciones más comunes.

1.- Una lista puede crecer como máximo el tamaño del arreglo. Es una limitación importante. Se dice en este caso que la memoria disponible es estática.

2.- Una lista está vacía si y sólo si L.ult = -1

3.- La lista esta cargada a la izquierda y se supone que entre a_i y a_{i+1} no hay celdas vacías. Es decir la lista está compactada a la izquierda.

Definición:

Dada una lista L se define fin(L) como la posición siguiente a la posición n en una lista de n elementos.
Si la lista está vacía retorna 0.
Si long-máx = L.ult entonces fin(L) no existe.

Implementación de operaciones.

1. Anula (L), deja la lista L vacía (drásticamente), retorna la lista L vacía.
   L.ult ← -1
   T(n) = cte, cualquier-caso.
   ➔Anula(L) es O(1)

2. Vacía(L), determina si la lista L está vacía. Retorna V o F según el caso.
   if (L.ult = -1) retorna V
   T(n) = cte, cualquier-caso.
   ➔ Vacía(L) es O(1)

3. Inserta(x,p,L), inserta x en la posición p de la lista L.
   Desplaza los elementos de la posición L.ult hasta p en una lugar a la derecha. Muchas referencias pueden quedar obsoletas.
   Se puede insertar si y sólo si p < = L.ult
   Si p = fin(L) agrega al final de la lista. Si L esta vacía agrega el primer elemento.
   Requiere actualizar L.ult
   T(n) = n para el peor-caso
   T(n) = (n+1)/2 caso-promedio
   ➔ Inserta(x,p,L) es O(n)

4. Suprime(p,L), elimina el elemento de la lista L que está en la posición p.
   Desplaza los elementos de la posición p + 1 hasta L.ult en un lugar a la izquierda. Muchas referencias pueden quedar obsoletas.
   Se puede eliminar si la lista no está vacía.
   Requiere actualizar L.ult
   T(n) = n - 1 para el peor-caso
   T(n) = (n+1)/2 caso-promedio
   ➔ Suprime(L) es O(n)

5. Localiza(x,L), retorna la posición de la primera ocurrencia de x en la lista L.
   Si x no está en la lista retorna fin(L).
   T(n) = n para el peor-caso
   T(n) = (n+1)/2 caso-promedio
   ➔ Localiza(x,L) es O(n)

6. Recupera(p,L), retorna el elemento que está en la posición p de la lista L.
   T(n) = cte, cualquier-caso.
   ➔ Recupera(p,L) es O(1)

7. Siguiente(p,L), retorna la posición siguiente de p en L.
   p+1 es la posición siguiente
   Si p = L.ult retorna fin(L).
   Si p = fin(L) indefinida
   T(n) = cte, cualquier-caso.
   ➔ Siguiente(L) es O(1)

8. Anterior(p,L), retorna la posición anterior de p en L.
   p-1 es la posición anterior
   Si p = fin(L) retorna L.ult .
   Si p = primera posición, indefinida
   T(n) = cte, cualquier-caso.
   ➔ Anterior(p,L) es O(1)

9. Imprime(L) imprime los elementos de L en un orden especificado.
   T(n) = n, cualquier-caso.
   ➔ Imprime(L) es O(n)

10. Cambia(x,p,L), Cambia el elemento de la posición p por x y retorma la lista L modificada
    T(n) = cte, cualquier-caso
    ➔ Cambia(x,p,L) es O(1)

11. Vaciar(L) suprime los elementos de L uno por uno hasta dejar la lista vacía.
    Se debe especificar una estrategia para suprimir, por ejemplo:

```
Vaciar(L)
while no está vacía(L)
      suprime(L.ult, L)
```

Ejemplo:
¿Qué hace el siguiente algoritmo?

```
int p,q = 0;
p = primera(L);

while ( p <> fin(L) )
  {
     q = siguiente(p,L);
     while  (q <> fin(L) )
       {
          if sonIguales(recupera(p,L), recupera(q,L))
             suprime(q,L);
          else
             q = siguiente(q,L);
       }
      p = siguiente(p,L);
  }
```