Listas

Algoritmos y Estructuras de datos

Prof. Ing. Gomez Pablo

29 de marzo de 2023

Introducción

Temario

- Lista
- Secuencia
- Asignación dinamica/estática de memoria
- Standart template library (STL) - Vectores
- Tipos abstractos de datos (TAD)
- Definición de TADs
- Especificación algebraica y aridad

Introducción

Lista , definición

Secuencia de elementos de un determinado tipo.

Secuencia , definición

Relación matemática entre los numeros naturales y los elementos a incluir.

1 $\Rightarrow$ dato1

2 $\Rightarrow$ dato2

3 $\Rightarrow$ dato3

...

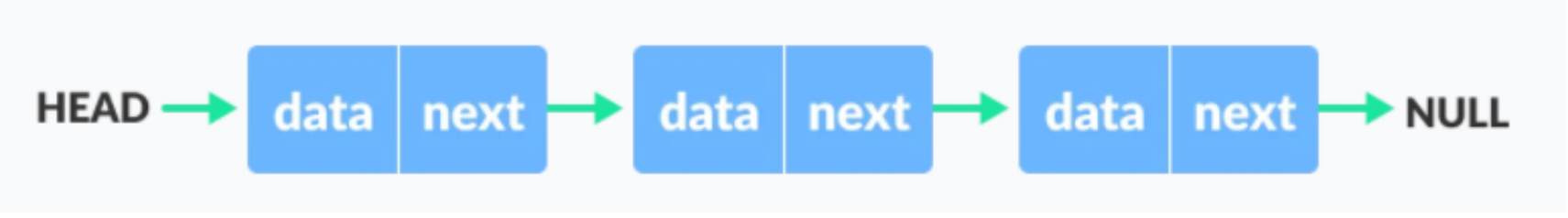
n $\Rightarrow$ dato n

Listas

Lista

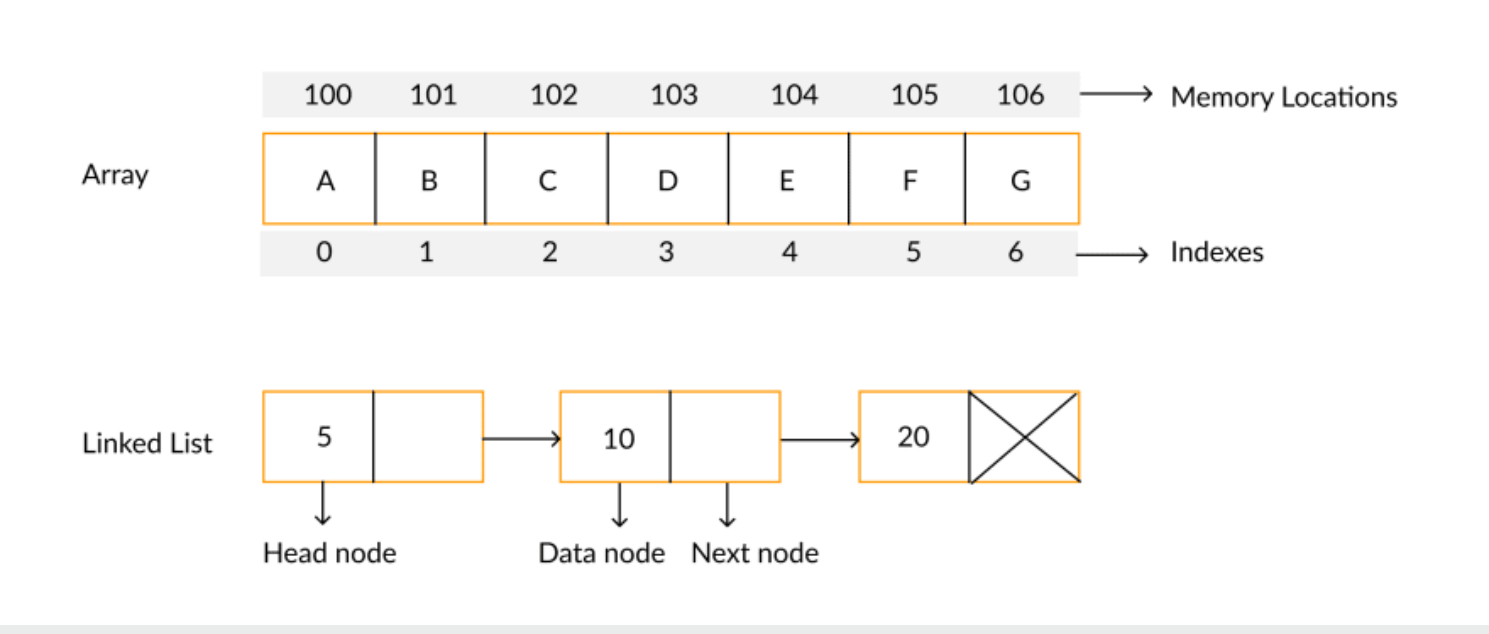
- No tiene un numero prefijado o limitado de elementos
- Su longitud es la cantidad de elementos que tiene #L. Si la longitud es cero se dice que la lista es VACIA
- Es un tipo de dato que contiene a otro. Por ej. una lista de enteros, una lista de strings, una lista de objetos de una determinada clase. No pueden mezclarse tipos de dato dentro de una lista.

Lista enlazada



Uso de memoria

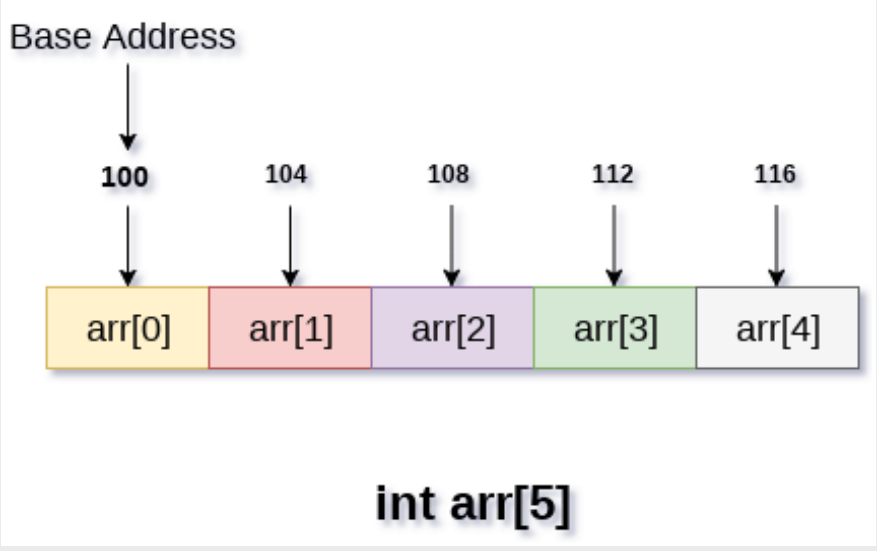
- Uso de memoria DINÁMICO
- En contrapartida con los arreglos , que tienen un numero predefinido de valores/elementos (Uso de memoria ESTÁTICO)



Arreglos vs Listas

Estático

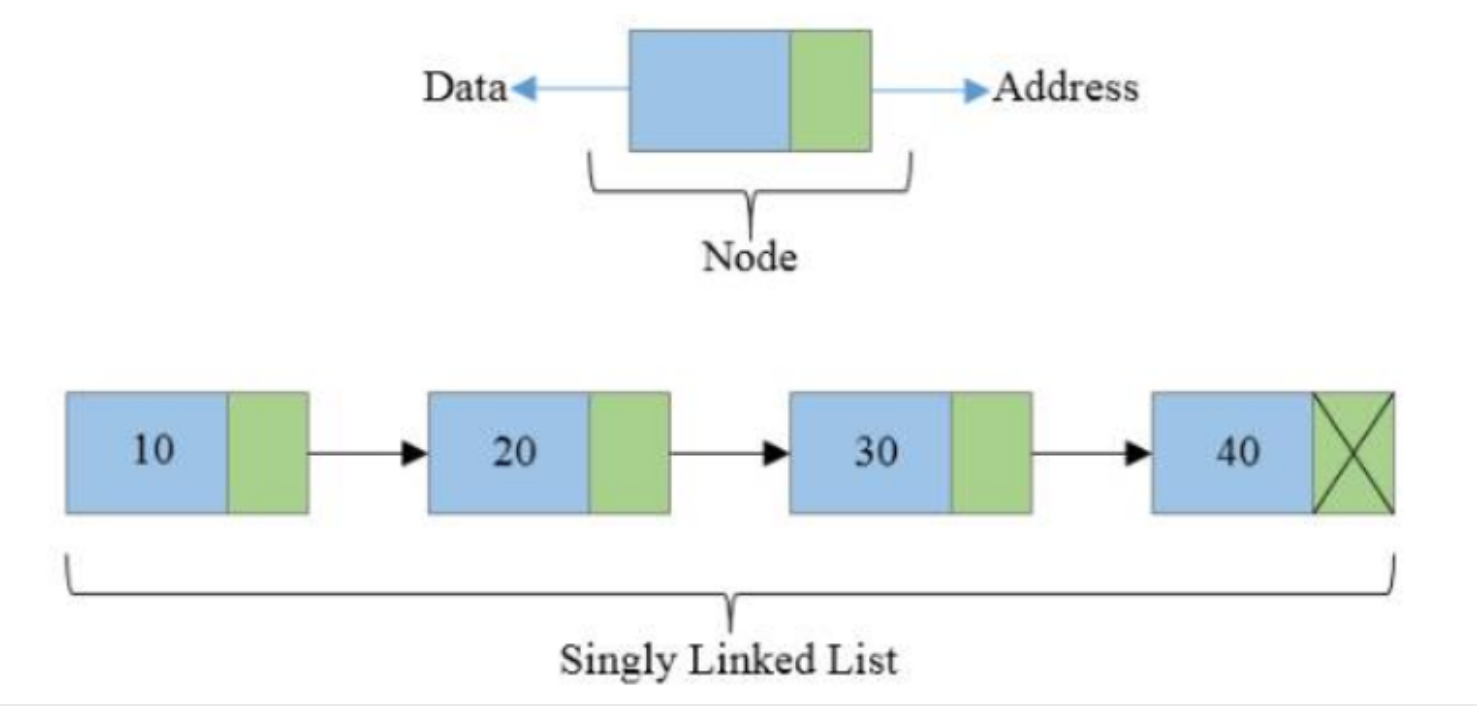
- Uso más eficiente para acceder a memoria. Acceso directo o indexado a cada dato.
- Numero fijo de elementos.
- Reservo memoria extra para garantizar cobertura hasta un máximo de datos.



Arreglos vs Listas

Dinámico

- Acceso secuencial a memoria.
- Sin límite en la cantidad de elementos.
- Se utiliza la memoria realmente necesaria. Se solicita cada vez que se incorpora un dato.



Listas

Estructura recursiva

- Acceder al elemento n $\Rightarrow$ siguiente elemento n-1
- Acceder al elemento n-1 $\Rightarrow$ siguiente elemento n-2
- ...
- Acceder al elemento 1 $\Rightarrow$ comienzo de la lista

Representación

- Lista de enteros: [2, 4, 6]
- Lista vacía: []
- Acceso al elemento de indice 1 de la lista a: a[1]

Vectores

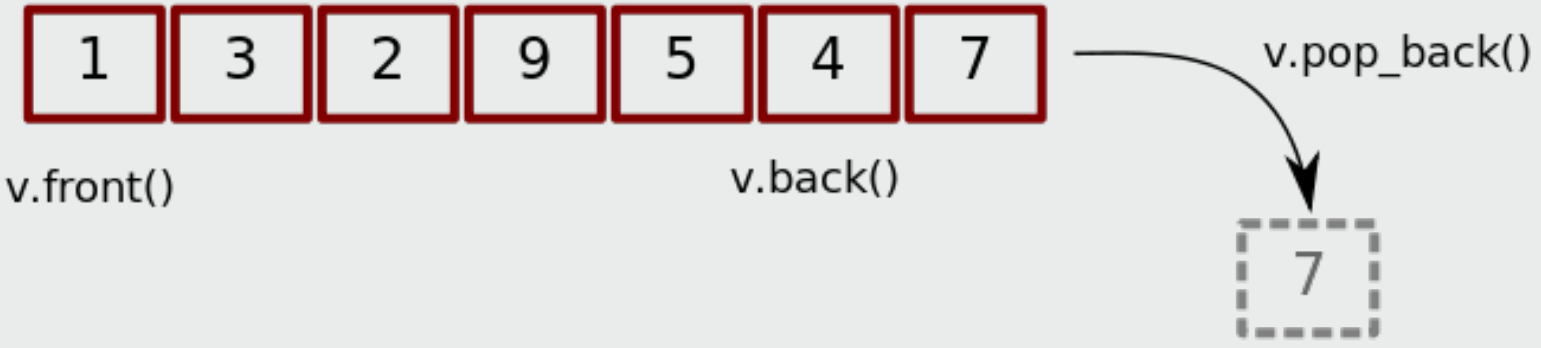
Estructura similar a un arreglo pero puede cambiar de tamaño

- Se encuentran definidos en la Standard Template Library (STL)
- Tiene funciones miembro de modificación, iteración y capacidad.
- Al extender la capacidad existen dos enfoques :
  Acceso mixto (seudo indexado). No se puede reservar memoria dinamicamente extendiendo el rango en la misma región de memoria. “Como una Lista de arreglos”
  Almacenar por ej. 10 posiciones y en vez de sumar otras diez reservaríamos 20 posiciones y debo copiar los valores del primer vector al segundo (necesito memoria para alojar ambos - suele no ser conveniente)

Vectores

Funciones miembro

- Modificación: insert() , erase()
- Iteración: begin( ), end( )
- Capacidad: size( ), empty( ), resize( )



Ejemplo uso vector

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main()
6  {
7      vector<int> g1;
8      for (int i = 1; i <= 5; i++)
9          g1.push_back(i);
10
11     cout << "Vector: ";
12     for (vector<int>::iterator i = g1.begin();
13             i != g1.end(); ++i)
14         cout << *i << " ";
15     cout << endl ;
```

Ejemplo uso vector

```
1        cout << "Tamanio"<< g1.size()<< endl;
2        cout << "Capacidad"<< g1.capacity()<< endl;
3        cout << "Tam. Max."<< g1.max_size()<< endl;
4
5        g1.resize(4);
6        cout << "Tamanio: " << g1.size() << endl;
7        cout << "Vector: ";
8
9        for (vector<int>::iterator i = g1.begin();
10              i != g1.end(); ++i)
11           cout << *i << " ";
12       cout << endl;
```

Ejemplo uso vector

```
1        if (g1.empty() == false)
2            cout << "Vector no vacio" << endl;
3        else
4            cout << "Vector vacio" << endl;
5
6        cout << "g1[2]=" << g1[2] << endl;
7        cout << "g1.at(3)=" << g1.at(3) << endl;
8        cout << "g1.front()=" << g1.front() << endl;
9        cout << "g1.back()=" << g1.back() << endl;
10
11       return 0;
12   }
```

Tipos abstractos de datos

Tipo de datos

Clasifica los objetos de los programas (variables, parámetros, constantes) y determina los valores que pueden tomar. También determina las operaciones que se aplican

- Entero: operaciones aritméticas enteras (suma, resta, . . .)
- Booleano: operaciones lógicas (y, o, . . .)

Tipo abstracto de datos

La manipulación de los datos sólo dependen del comportamiento descrito en su especificación (qué hace) y es independiente de su implementación (cómo se hace) Una especificación $\Rightarrow$ Múltiples implementaciones

Tipos abstractos de datos

Especificación de un TAD

Consiste en establecer las propiedades que lo definen. Debe ser:

- Precisa: sólo produzca lo imprescindible
- General: sea adaptable a diferentes contextos
- Legible: sea un comunicador entre especificador e implementador
- No ambigua: evite problemas de interpretación

Definición informal (lenguaje natural) o formal (algebraica)

Tipos abstractos de datos

Especificación algebraica

Especificación algebraica (ecuacional): establece las propiedades de un TAD mediante ecuaciones con variables cuantificadas universalmente, de manera que las propiedades dadas se cumplen para cualquier valor que tomen las variables. Pasos:

- Identificación de los objetos del TAD y sus operaciones
- Definición de la signatura (sintaxis) de un TAD (nombre del TAD y perfil de las operaciones)
- Definición de la semántica (significado de las operaciones)

Tipos abstractos de datos

Especificación algebraica

Es la definición de las funciones y el comportamiento de las funciones dentro del tipo que se esta definiendo. En este caso el tipo Lista de X (donde X también es un tipo de dato), por ej: Lista de Enteros.

Aridad de una función

Es la definición de la cantidad y tipos de datos que la función toma como parámetros y el tipo de dato que devuelve. Por ej. Suma es un operador binario, porque necesita 2 argumentos para poder realizar una suma.

Especificación algebraica - Ejemplo Naturales

Tipo: Natural

Operaciones

- cero: $\Rightarrow$ Natural
- sucesor: Natural $\Rightarrow$ Natural
- suma: Natural x Natural $\Rightarrow$ Natural

Variables

n1,n2 : Natural

Comportamiento

suma(n1,cero()) = n1
suma(cero(), n1) = n1
suma(n1, sucesor(n2)) = sucesor (suma(n1,n2))

Ejemplo Naturales - Extensión

Tipo: Natural

Operaciones

- cero: $\Rightarrow$ Natural
- sucesor: Natural $\Rightarrow$ Natural
- esCero: Natural $\Rightarrow$ Booleano
- igual: Natural x Natural $\Rightarrow$ Booleano
- predecesor: Natural $\Rightarrow$ Natural
- suma: Natural x Natural $\Rightarrow$ Natural
- mult: Natural x Natural $\Rightarrow$ Natural

Variables

n1,n2 : Natural

Ejemplo Naturales - Extensión

Comportamiento

esCero (cero()) = true
esCero (sucesor (n1)) = false
igual (cero(), n1) = esCero (n1)
igual (sucesor (n1), cero) = false
igual (sucesor (n1), sucesor (n2)) = igual (n1, n2)
suma(n1,cero()) = n1
suma(cero(), n1) = n1
suma(n1, sucesor(n2)) = sucesor (suma(n1,n2))
mult (cero(),n1) = cero()
mult (n1, cero()) = cero()
mult(sucesor(n1), n2) = suma(mult(n1, n2), n2)
pred(suc(n1)) = n1
pred(cero()) $\Rightarrow$ error

Listas

Tipo: Lista de Y

Aridad de funciones

- crearLista: $\Rightarrow$ Lista (*)
- agregar: Lista x Y $\Rightarrow$ Lista (*)
- cabeza: Lista $\Rightarrow$ Y
- resto: Lista $\Rightarrow$ Lista
- esVacia : Lista $\Rightarrow$ Bool

(*) Funciones constructoras

Listas - Especificación algebraica

Variables

y1,y2: Y
L1,L2: Lista

Comportamiento

esVacia(crearLista()) $\Rightarrow$ true
esVacia(agregar(y1,L1)) $\Rightarrow$ false

cabeza(agregar(y1,L1)) $\Rightarrow$ y1 (antepone los elementos)
resto(agregar(y1,L1)) $\Rightarrow$ L1

cabeza(crearLista()) $\Rightarrow$ error
resto(crearLista()) $\Rightarrow$ error
Inicialmente la lista está vacía

Listas Extensión

[2] $\Rightarrow$ agregar(2,creaLista())

[3,2] $\Rightarrow$ agr(3,agregar(2,crearLista()))

[4,1,7,10] $\Rightarrow$
agregar(4,agregar(1,agregar(7,agregar(10,crearLista()))))

[4,1] concat [7,10] $\Rightarrow$ [4,1,7,10]

resto(agr(2,creaLista())) = crearLista()

Listas Extensión

Tipo: Lista de Y

Aridad de funciones

- crearLista: $\Rightarrow$ Lista
- agregar: Lista x Y $\Rightarrow$ Lista
- cabeza: Lista $\Rightarrow$ Y
- resto: Lista $\Rightarrow$ Lista
- esVacia : Lista $\Rightarrow$ Bool

- concat:Lista x Lista $\Rightarrow$ Lista
- inicio:Lista x Nat $\Rightarrow$ Lista
- final:Lista x Nat $\Rightarrow$ Lista

Listas Extensión - Especificación algebraica

Variables

y1,y2: Y
L1,L2: Lista

Comportamiento

esVacia(crearLista()) $\Rightarrow$ true
esVacia(agregar(y1,L1)) $\Rightarrow$ false
cabeza(agregar(y1,L1)) $\Rightarrow$ y1
resto(agregar(y1,L1)) $\Rightarrow$ L1
cabeza(crearLista()) $\Rightarrow$ error
resto(crearLista()) $\Rightarrow$ error

concat(crearLista(),L1) $\Rightarrow$ L1
concat(agregar(y2,L2),L1) = agregar(y2,concat(L2,L1))

Listas Extensión - Especificación algebraica

concat(agregar(4,agregar(1,crearL())),agregar(7,agregar(10,creaL())))
concat([4,1], [7,10]) = [4,1,7,10]

agregar(4,concat(agregar(1,crearL()),agregar(7,agregar(10,crearL()))))
agregar ([4], (concat([1], [7,10])))

agregar(4, agregar(1, concat(crearL(), agregar(7, agregar(10, crearL()))))
agregar (4, agregar(1 , concat([ ] , [7,10])))

agregar(4,agregar(1,agregar(7,agregar(10,crearL()))))

$[4, 1, 7, 10]$

Listas Extensión - Especificación algebraica

final(inicio(lista,3),1)

lista= [a1,a2,a3,a4,a5,a6]

inicio(lista,3) = [a1,a2,a3]

final (ini..., 1) = [a2,a3]

insertar(L1,y,n) L:Lista, y:valor de tipo Y , n: posic

insertar(L1,y,n) = concat(inicio(L1,n-1), agr(y,final(L1,n-1)))

Implementación

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <sstream>
4  using namespace std;
5
6  class Nodo{
7  private:   int dato;
8             Nodo *next;
9  public: Nodo() {next=NULL;};
10         Nodo(int a) {dato=a; next=NULL;};
11         int get_dato() {return dato; };
12         Nodo *get_next() {return next; };
13         void set_next(Nodo *n) {next=n; };
14         bool esvacio() {return next==NULL; };
15 };
```

Implementación

```
1  class Lista{
2      private:
3        Nodo *czo;
4      public:
5        Lista() {czo=new Nodo();};
6        Lista(Nodo *n) {czo=n;};
7        //~Lista(void);
8        void agregar(int d);
9        bool esvacia();
10       int cabeza();
11       Lista *resto();
12       void imprimir();
13 };
```

Implementación

```
1  void Lista::agregar(int d){
2        Nodo *nuevo = new Nodo(d);
3        nuevo -> set_next(czo);
4        czo = nuevo;
5  }
6  void Lista::imprimir(){
7     Nodo *aux = czo;
8     cout << "Lista: ";
9     while(aux->get_next()!=NULL){
10          cout << aux->get_dato() << " ";
11          aux = aux->get_next();
12    }
13    cout << endl;
14 }
```

Implementación

```
1  bool Lista::esvacia(){
2       return czo -> esvacio();
3  }
4  int Lista::cabeza(){
5    if(this->esvacia()){
6       return -1;
7    }
8    return czo->get_dato();
9  }
10 Lista *Lista::resto(){
11         Lista *l=new Lista(czo->get_next());
12         return (l);
13 }
```

Implementación

```
1  int main()
2  {
3      Lista *l=new Lista();
4      l->imprimir();
5      cout << "Vacia: "<< l->esvacia() << endl;
6
7      l->agregar(532);
8      l->agregar(52);
9      l->agregar(22);
10     l->imprimir();
11     cout << "Vacia: "<< l->esvacia() << endl;
12     cout << "Cabeza: "<< l->cabeza() << endl;
13     cout << "Resto: " ;
14     l->resto()->imprimir();
15     return EXIT_SUCCESS;
16 }
```