Facultad de Ingeniería Ingeniería y de Sistemas

Algoritmia y Estructura de Datos: Punteros y estructuras dinámicas

Ing. Juan Carlos Sotelo, M.Sc.

¿Qué es un puntero?

Definición

Es una variable que contiene la dirección de memoria de otra variable.
Permiten el pase de parametros por referencia a una función.
Sirven para flexibilixar la programación y permiten crear estructuras de datos dinámicas

Importante

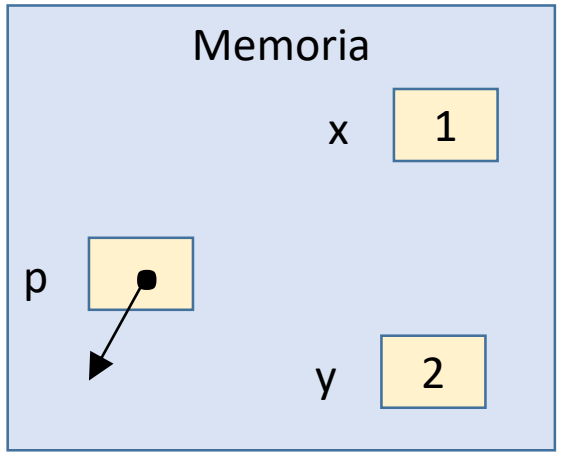
Operador **&** permite obtener la dirección de memoria de una variable
Operador * permite acceder al contenido de una dirección apuntada por una variable puntero

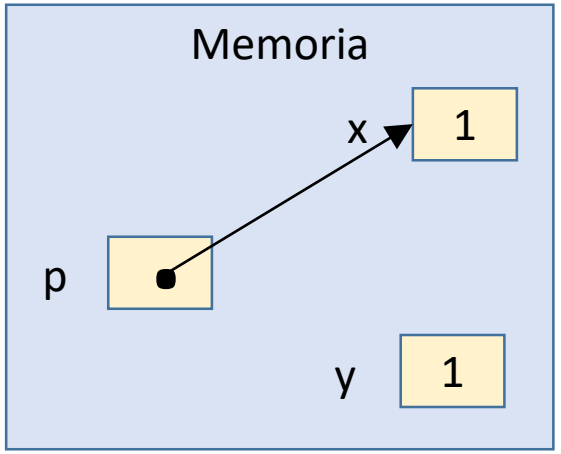
```
int x=1,y=2;
int *p;
//p es un puntero a int
```

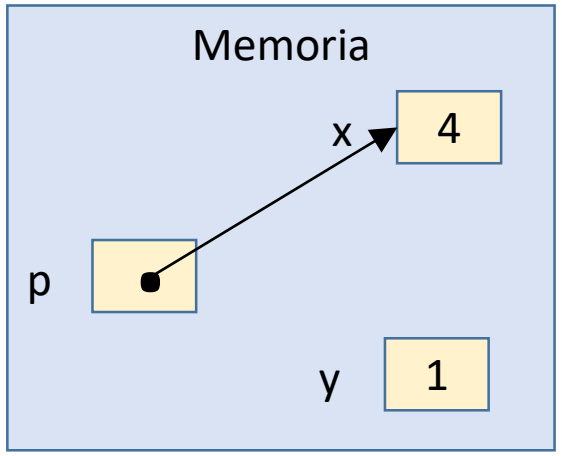
```
p=&x;  //p apunta a x
y=*p;
//y toma el valor de
//lo apuntado por p
```

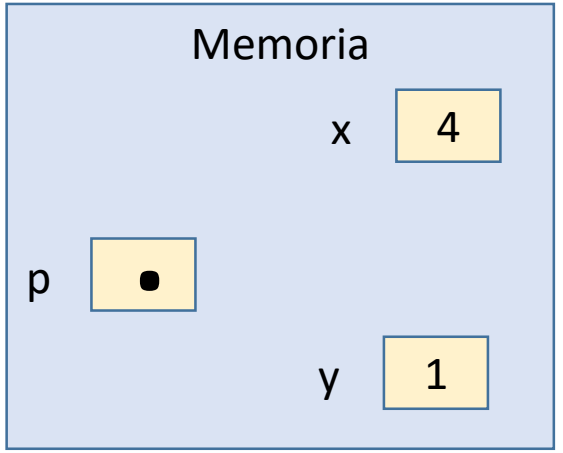
```
*p=*p+3;
//incrementa en 3 lo
//apuntado por p
```

```
p=NULL;
//p no apunta a nada
```









Arreglos y punteros en C++

Programa que recorre un arreglo de enteros con un puntero a entero

```
#include <iostream>
#include <fstream>
using namespace std;
main(){
   int i,x[5]={1,2,3,4,5};
   int *p;
   p=&x[0];
   while (p!=NULL){
      cout<<*p<<endl;
      if (*p==5)
        p=NULL;
      else
          p++;
   }
}
```

p es un puntero a entero

p toma la dirección del primer elemento del arreglo x

Muestra el contenido apuntado por el punter p

p toma el valor de nada

El puntero p apunta al siguiente element del arreglo x

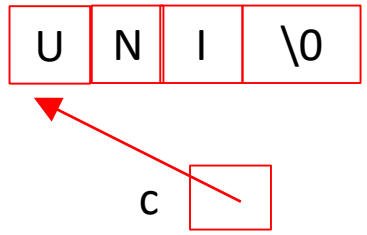
Flexibilidad: puntero en el pase de parámetros por referencia

```
#include <iostream>
using namespace std;
int cuentavocales(char *c){
  int cont=0;
  while(*c) { //mientras no llegue al nulo (0)
    switch(toupper(*c)){
      case 'A':
      case 'E':
      case 'I':
      case 'O':
      case 'U':cont++;
    }
    c++;
  }
  return cont;
}
main(){
  char cadenax[80],cadenay[50];
  cout<<"Ingrese primera cadena "; gets(cadenax);
  cout<<"Ingrese segunda cadena "; gets(cadenay);
  cout<<cadenax<<" = " <<cuentavocales(cadenax)<<" vocales"<<endl;
  cout<<cadenay<<" = " <<cuentavocales(cadenay)<<" vocales"<<endl;
}
```

c es un puntero a caracter y recibe la dirección del inicio de la cadena

*c es contenido apuntado por el puntero c

c++ Avanza el punter al siguiente caracter de la cadena



El ejemplo anterior pero con string

```
#include <iostream>
using namespace std;
int cuentavocales(char *c){
  int cont=0;
  while(*c) {            //mientras no llegue al nulo
    switch(toupper(*c)){
      case 'A':
      case 'E':
      case 'I':
      case 'O':
      case 'U':  cont++;
    }
    c++;
  }
  return cont;
}
main(){
  string cadenax,cadenay;
  cout<<"Ingrese primera cadena "; getline(cin,cadenax);
  cout<<"Ingrese segunda cadena "; getline(cin,cadenay);
  cout<<cadenax<<" = " <<cuentavocales(&cadenax[0])<<" vocales"<<endl;
  cout<<cadenay<<" = " <<cuentavocales(&cadenay[0])<<" vocales"<<endl;
}
```

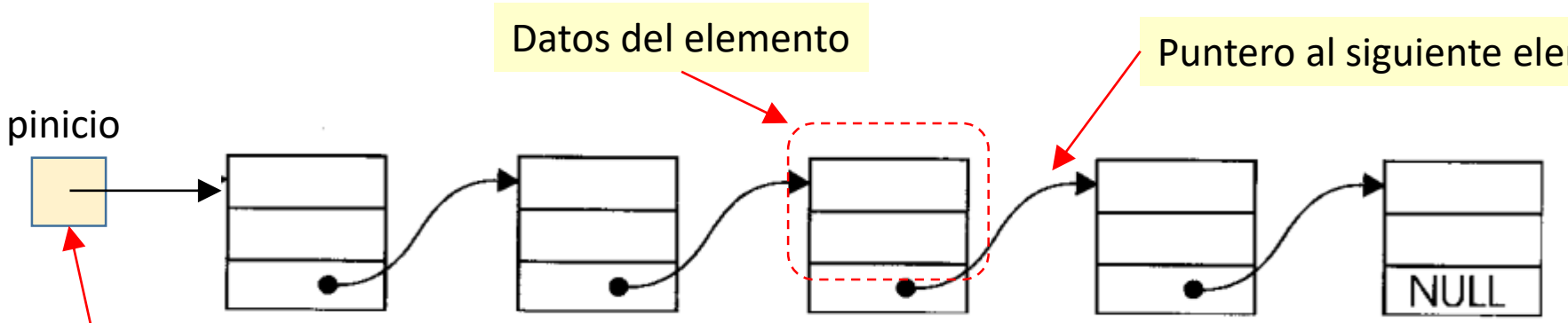
c es un puntero a caracter y recibe la dirección del inicio de la cadena

cadenax y cadenay son strings

&cadenax[0] es la direcci’on de memoria del primer character del string cadenax

Estructura dinámica: Lista enlazada

Una **lista** es una estructura que inicialmente está vacia y no ocupa lugar en memoria y luego a medida que se ejecuta la lógica del algoritmo va incorporando elementos o los va eliminado.

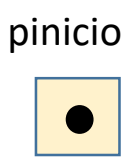


pinicio es el puntero que apunta al primer elemento de la **lista**

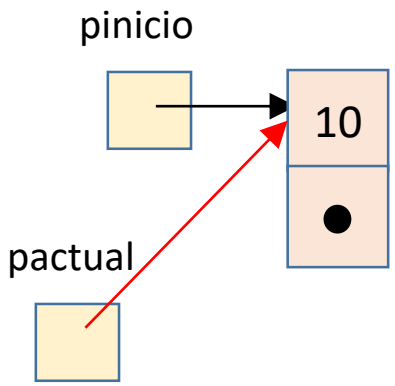
Todo elemento de la lista contiene datos y un puntero al siguiente elemento. El elemento es un registro,

El último elemento apunta a nada

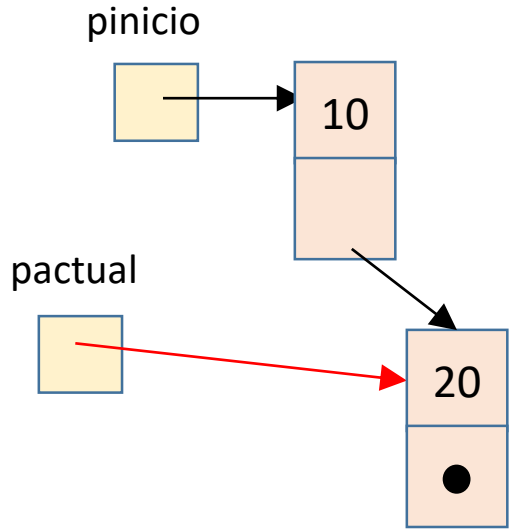
Construción de una lista enlazada



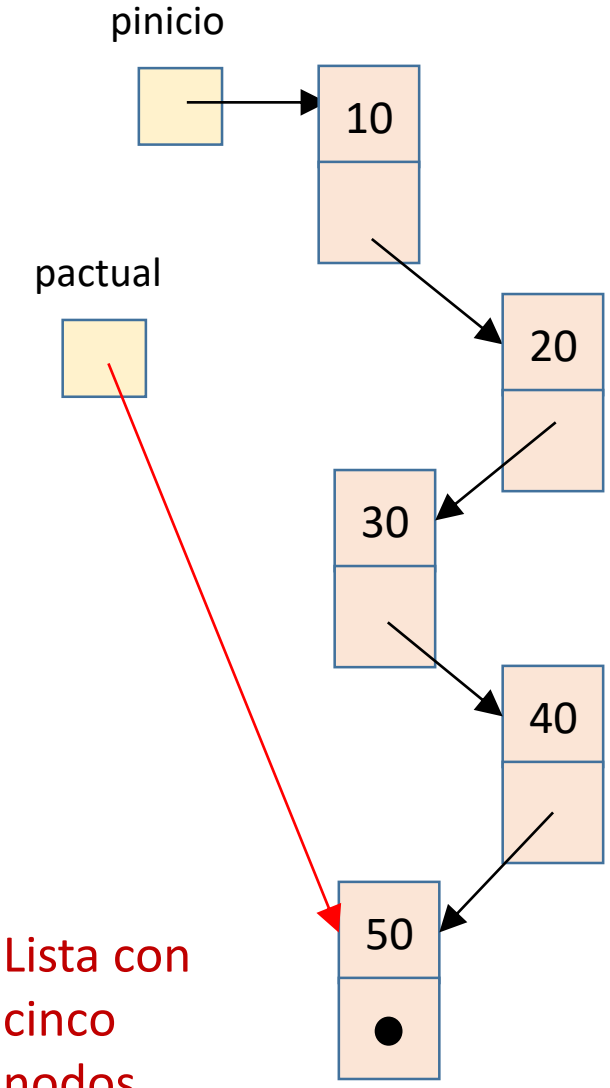
Lista vacia



Lista con
un nodo



Lista con
dos nodos



Lista con
cinco
nodos

Construción de una lista enlazada

```
struct reg_nodo{
   int numero;
   reg_nodo *psigue;
} *pinicio,*pactual,*pante,*paux;
pinicio = NULL;
int x[5]={10,20,30,40,50},i=0;
// agregar nodos a la lista con los valores de x
while (i<5){
   if (pinicio==NULL) {
      pinicio=new reg_nodo;
      pactual=pinicio;
   }
   else {
      pactual->psigue=new reg_nodo;
      pactual=pactual->psigue;
   }
   pactual->numero=x[i];
   pactual->psigue=NULL;
   i++;
}
```

Declara registro que representa al nodo de la lista

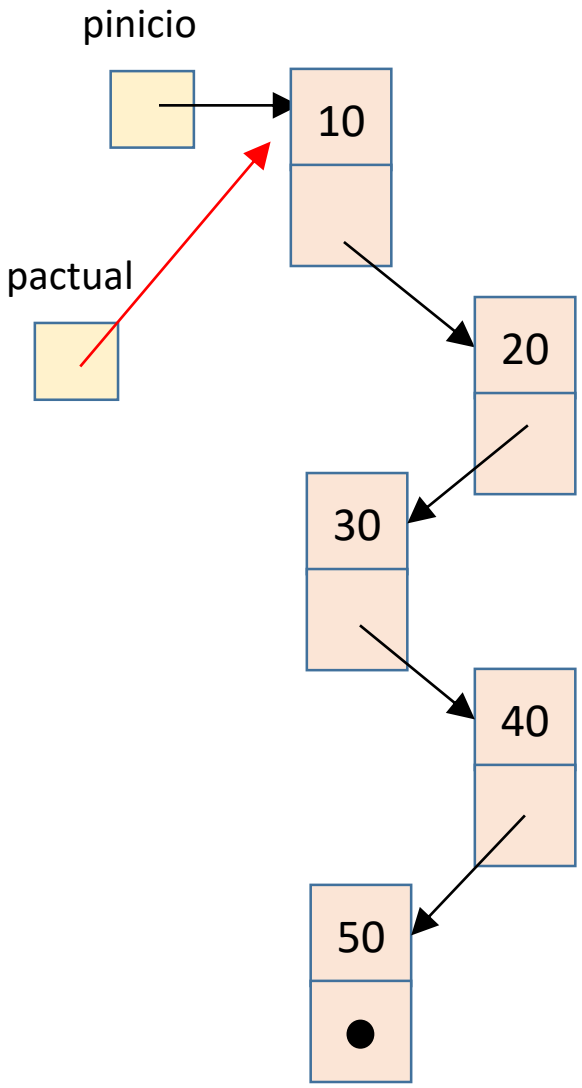
Declara punteros a nodo de la lista

Crea el primer nodo de la lista

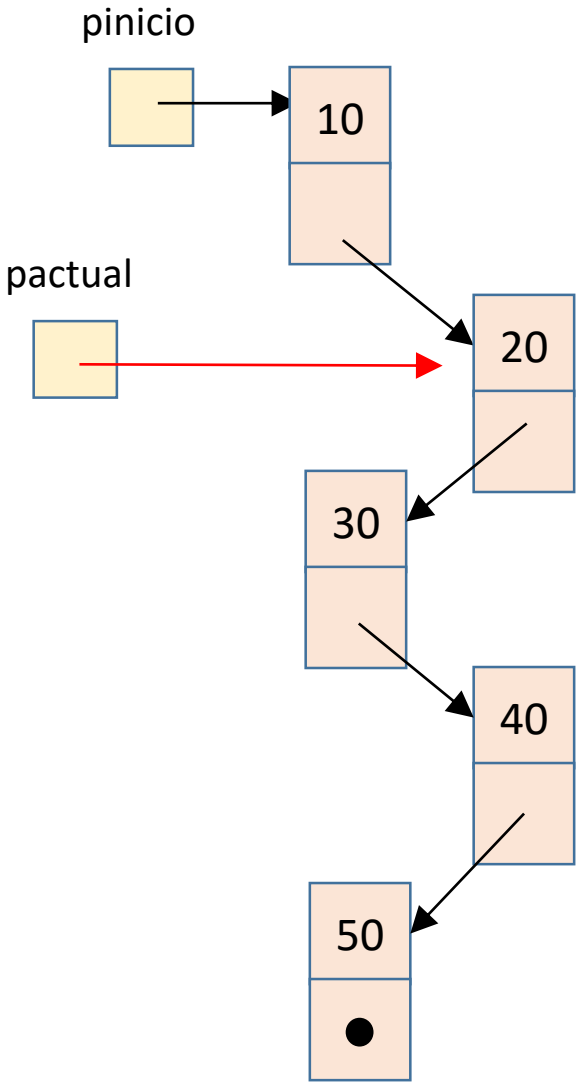
Agrega un nodo a la lista

Da valor al nodo y hace que el punter al siguiente sea nulo

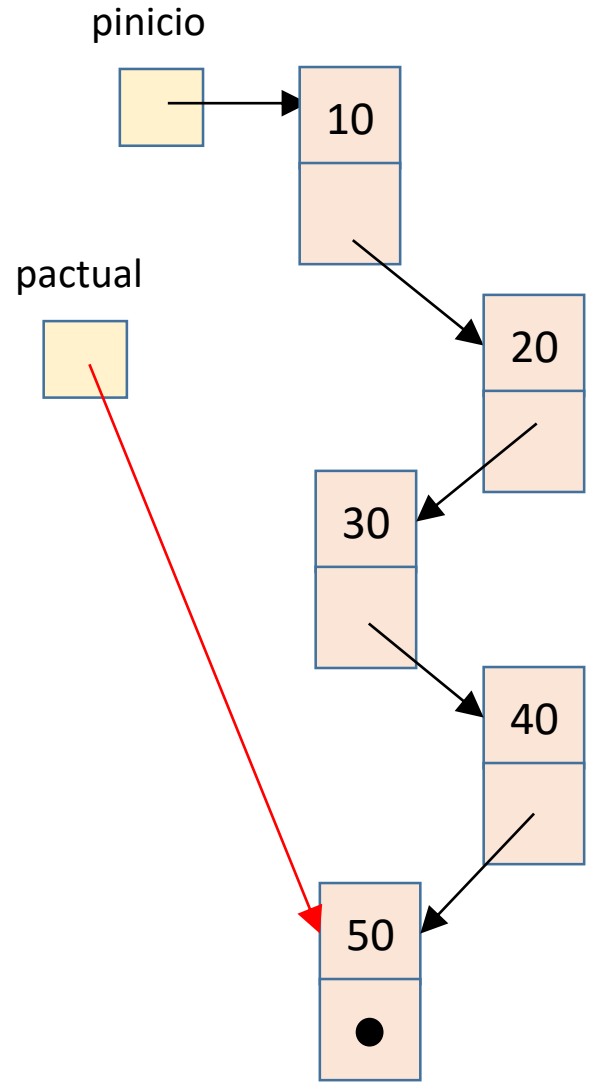
Recorrido de una lista enlazada



Contenido del primer nodo



Contenido del siguiente nodo



Contenido del ultimo nodo

Recorrido de una lista enlazada

```
struct reg_nodo{
   int numero;
   reg_nodo *psigue;
} *pinicio,*pactual,*pante,*paux;

...
...
...

// recorrido de la lista creada
pactual=pinicio;
while (pactual!=NULL){
   cout<<pactual->numero<<endl;
   pactual=pactual->psigue;
}
```

Declara registro que representa al nodo de la lista

Declara punteros a nodo de la lista

Apunta con el punter pactual al nodo inicial de la lista

Muestra el contenido del nodo

Apunta la siguiente nodo de la lista

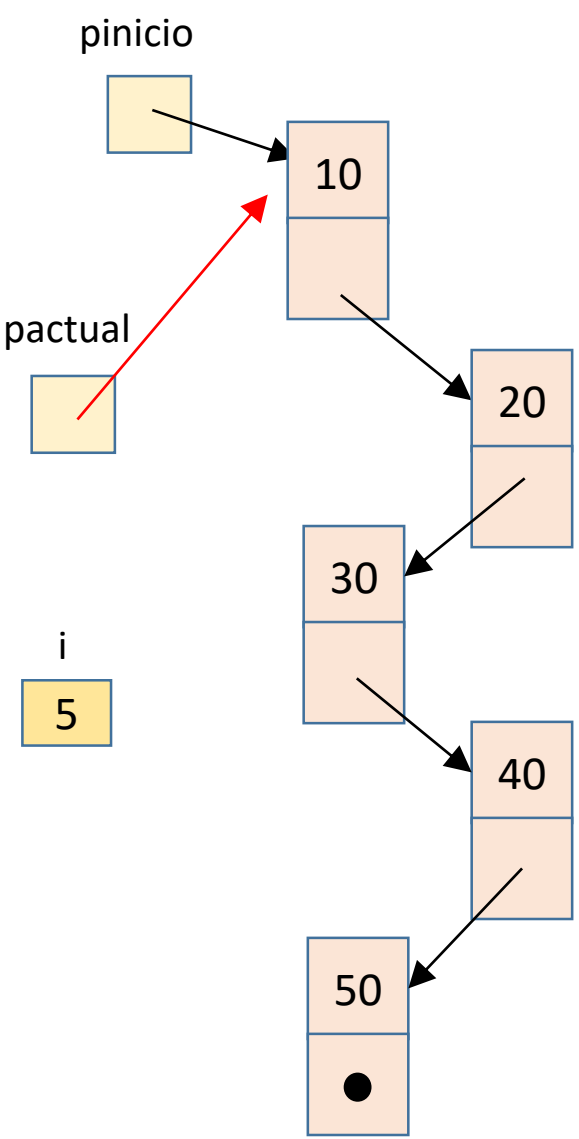
Insertar un nodo en la lista enlazada

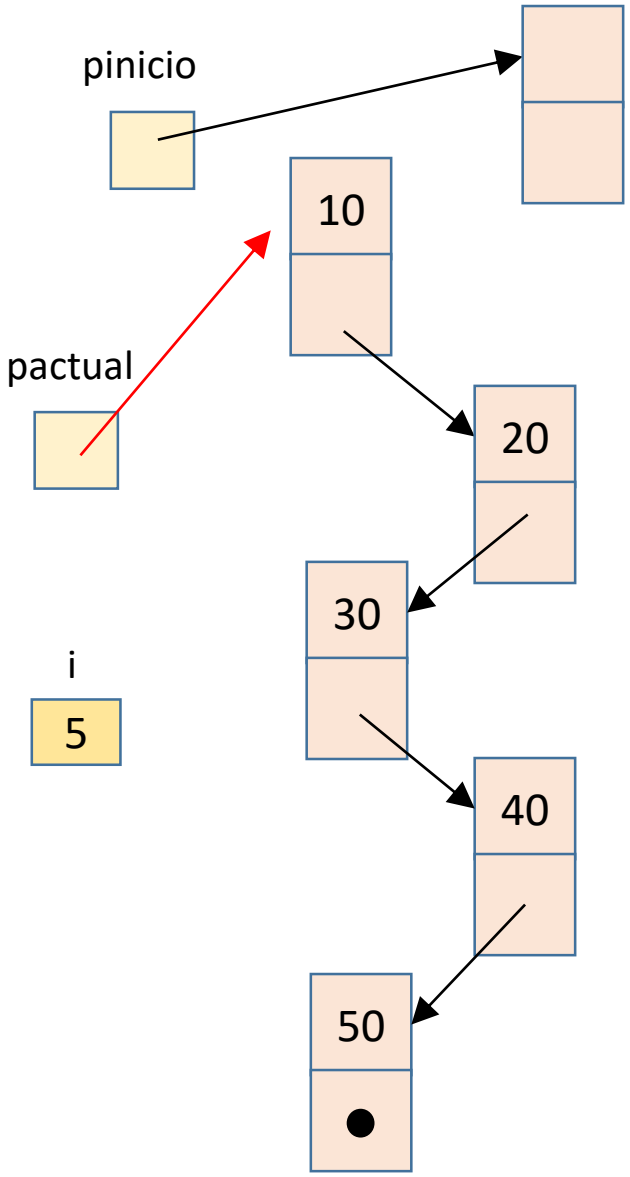
Para ilustrar la inserción, se tiene una lista de números ordenada de menor a mayor y insertaremos un nuevo número de forma tal que la lista siga ordenada

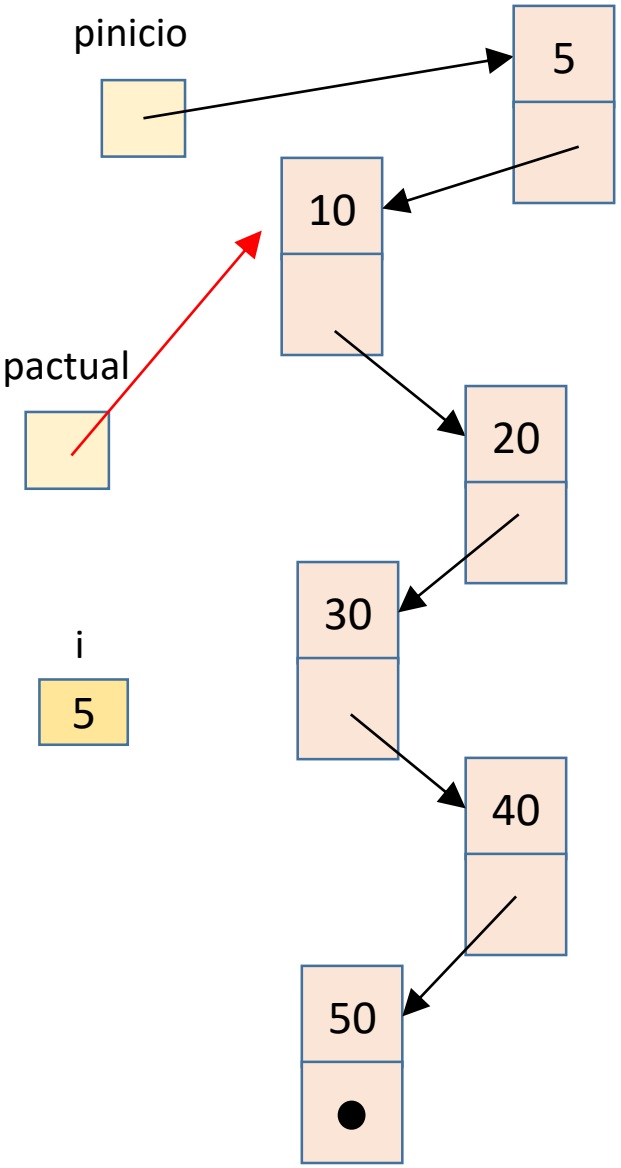
Se presentan 3 posibilidades

Insertar antes del primer nodo
Insertar entre dos nodos
Insertar despues del último nodo

Insertar antes del primer nodo de la lista







Insertar antes del primer nodo de la lista

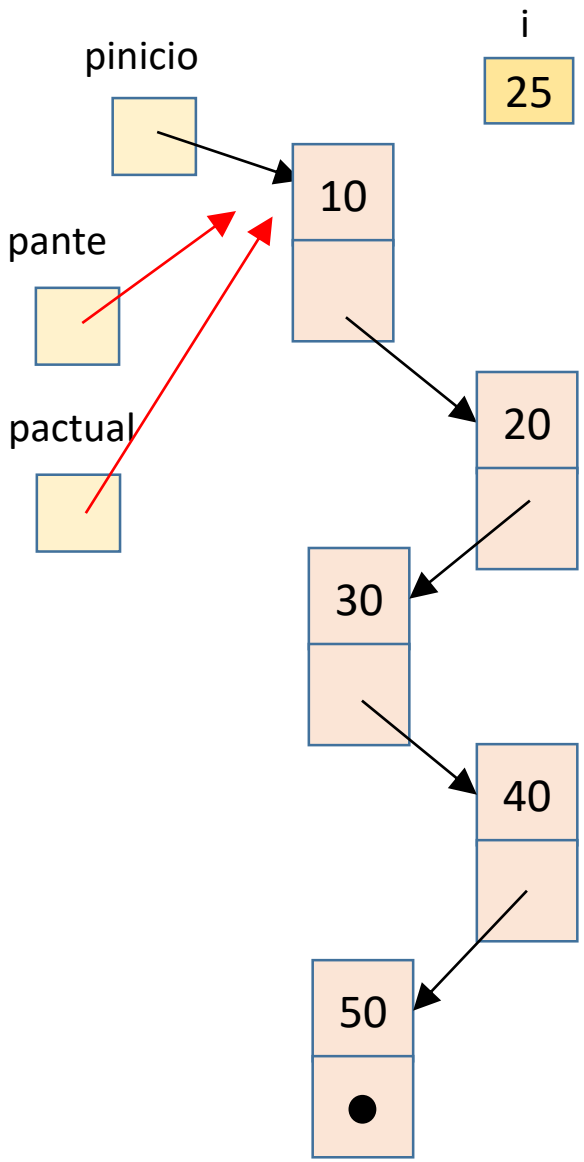
```
// insertando un valor en la lista ordenada
bool insertado=false;
pactual=pinicio;
cout<<"Ingrese el numero a insertar: ";
cin>>i;
if (i<=pinicio->numero){
  pinicio=new reg_nodo;
  pinicio->psigue=pactual;
  pinicio->numero=i;
  insertado=true;
}
else {
  ...
  //insertar entre nodos o al final
  ...
}
```

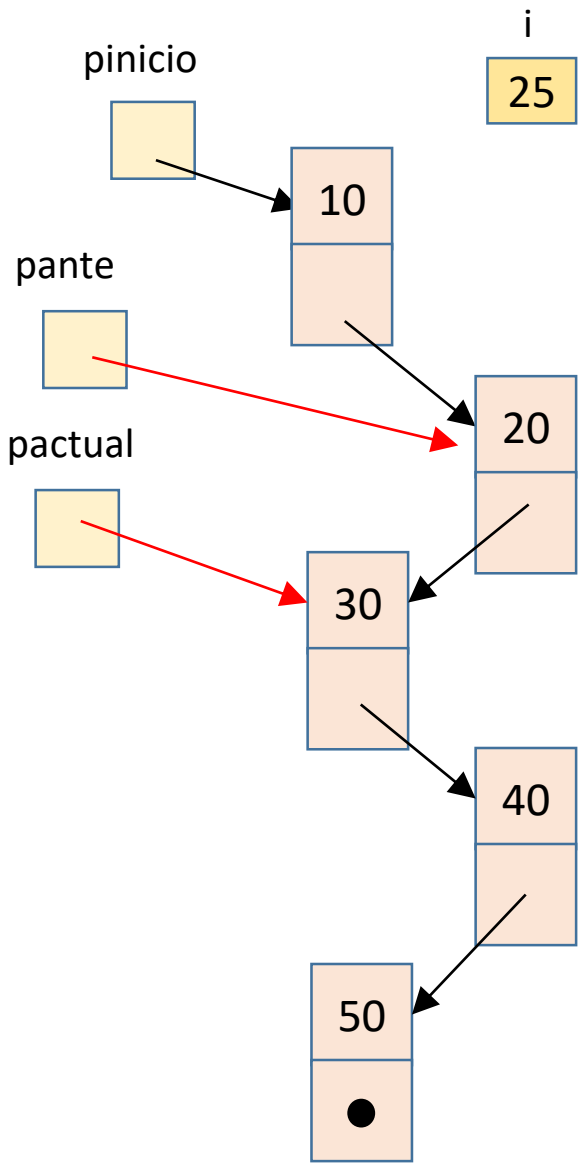
pactual apunta al nodo inicial

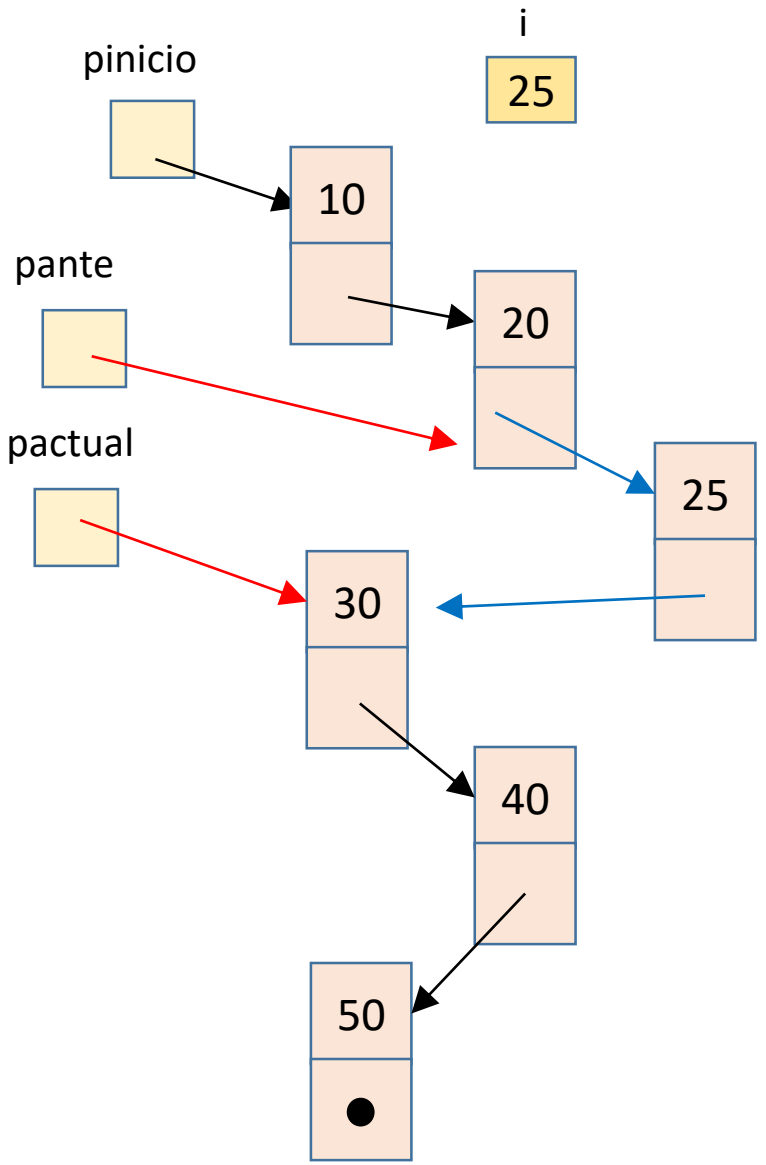
Verifica que el valor a inserter es menor al contenido del nodo inicial

Inserta antes del nodo inicial

Insertar entre nodos de la lista







Insertar entre nodos de la lista

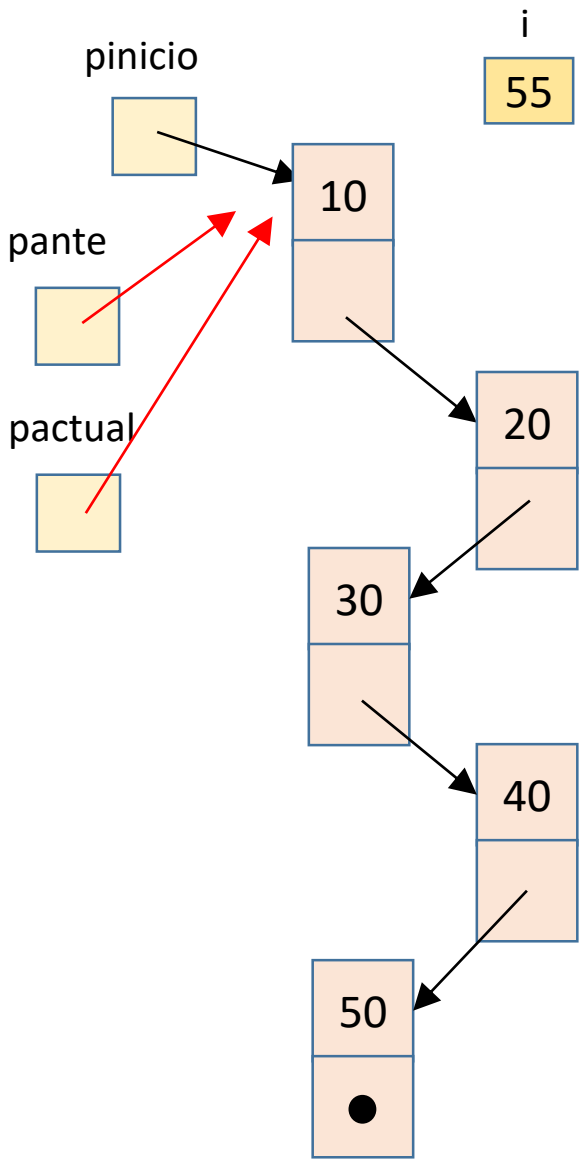
```
// insertando un valor en la lista para que siga ordenada
bool insertado=false;
cout<<"Ingrese un numero a insertar en la lista: "; cin>>i;
pactual=pinicio;
if (i<=pinicio->numero){
  //insertar antes del nodo inicial
}
else {
  while (!insertado && pactual!=NULL){
    if (i<=pactual->numero){
      pante->psigue=new reg_nodo;
      pante->psigue->numero=i;
      pante->psigue->psigue=pactual;
      insertado=true;
    }
    if (!insertado && pactual->psigue==NULL){
       // inserta despues del ultimo nodo
    }
    pante=pactual;
    pactual=pactual->psigue;
  }
}
```

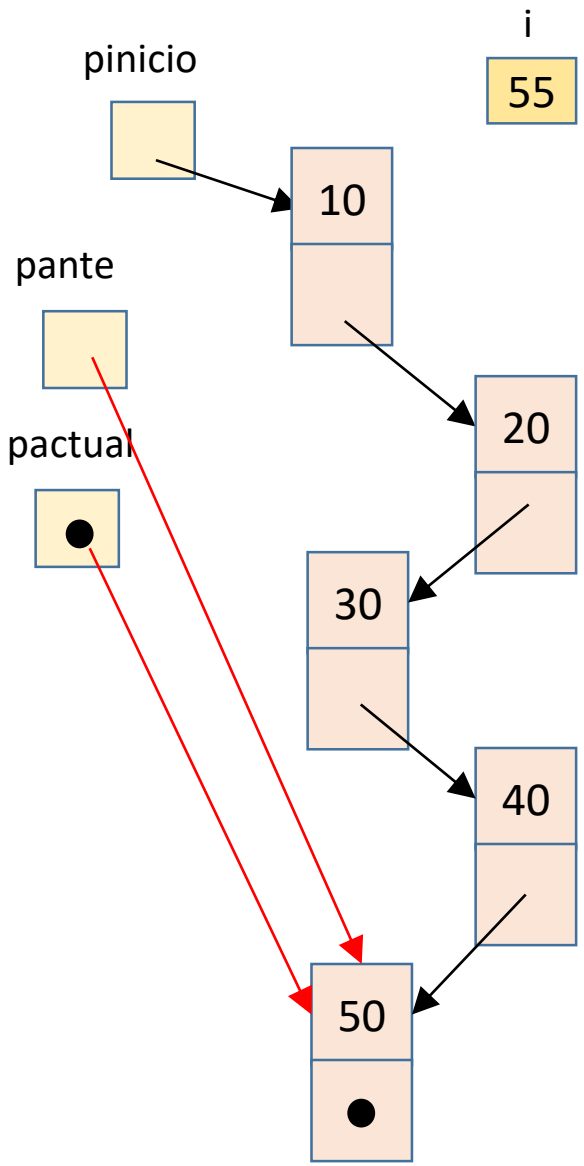
pactual apunta al nodo inicial para empezar el recorrido de la lista

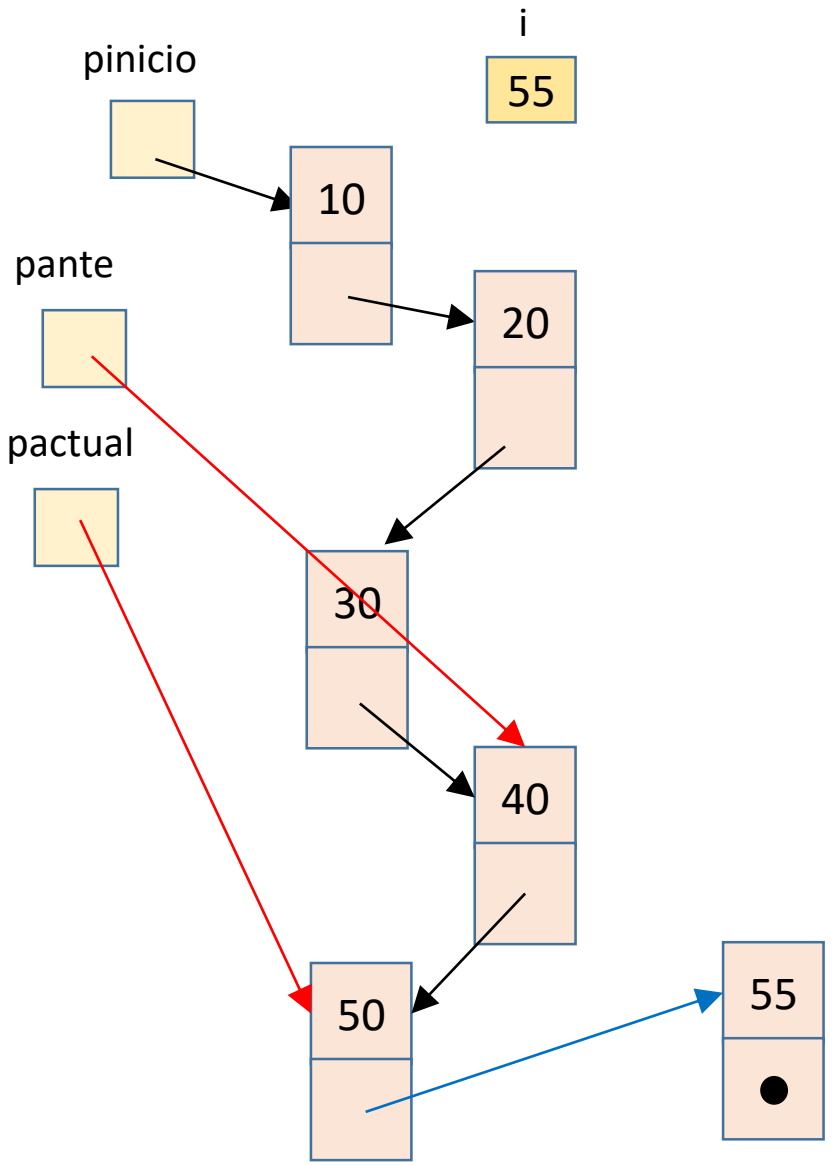
Inserta entre los nodos apuntados por pante y pactual

Avanza pante y pactual

Insertar al final de la lista







Insertar al final de la lista

```
// insertando un valor en la lista para que siga ordenada
bool insertado=false;
cout<<"Ingrese un numero a insertar en la lista: "; cin>>i;
pactual=pinicio;
if (i<=pinicio->numero){
  // inserter antes del primer nodo
}
else {
  while (!insertado && pactual!=NULL){
    if (i<=pactual->numero){
      //inserter entre nodos
    }
    if (!insertado && pactual->psigue==NULL){
      pactual->psigue=new reg_nodo;
      pactual->psigue->numero=i;
      pactual->psigue->psigue=NULL;
      insertado=true;
    }
    pante=pactual;
    pactual=pactual->psigue;
  }
}
```

pactual apunta al nodo inicial para empezar el recorrido de la lista

Detecta que tiene que insertar después del ultimo nodo

Inserta despues del ultimo nodo

Avanza pante y pactual

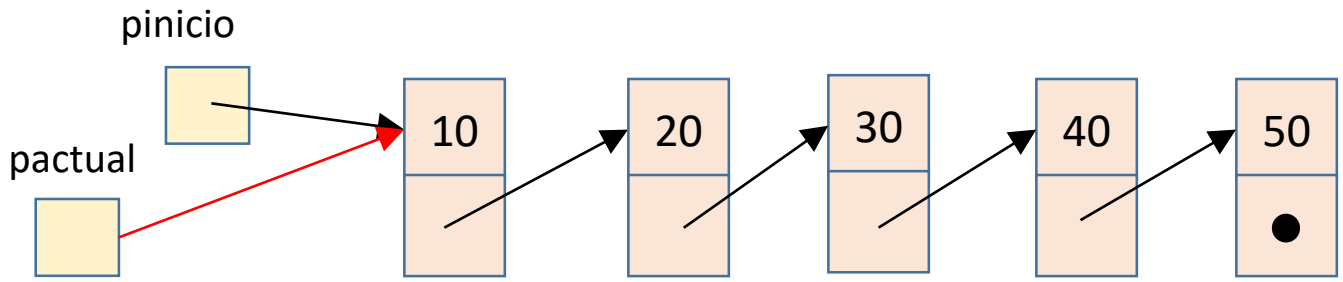
Eliminar el primer nodo de la lista

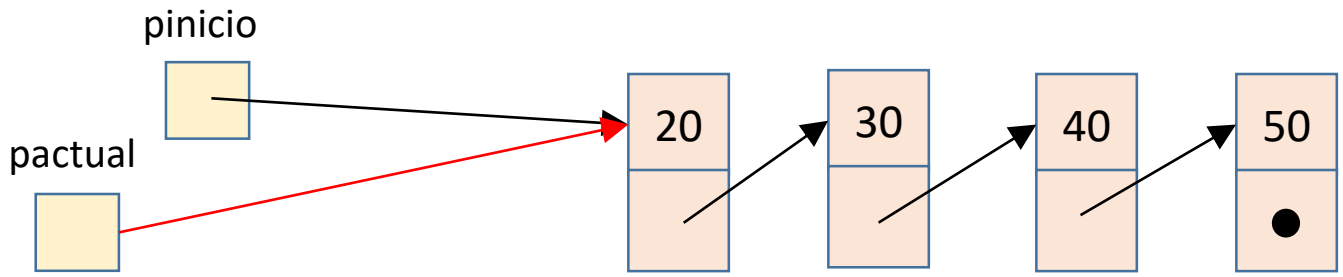
```
// eliminar el primer elemento de la lista
pactual=pinicio;
if (pinicio!=NULL) {
   pactual=pactual->psigue;
   delete pinicio;
   pinicio=pactual;
}
```

pactual apunta al nodo inicial

pactual apunta al nodo siguientel

Se libera el espacio de memoria apuntado por pinicio

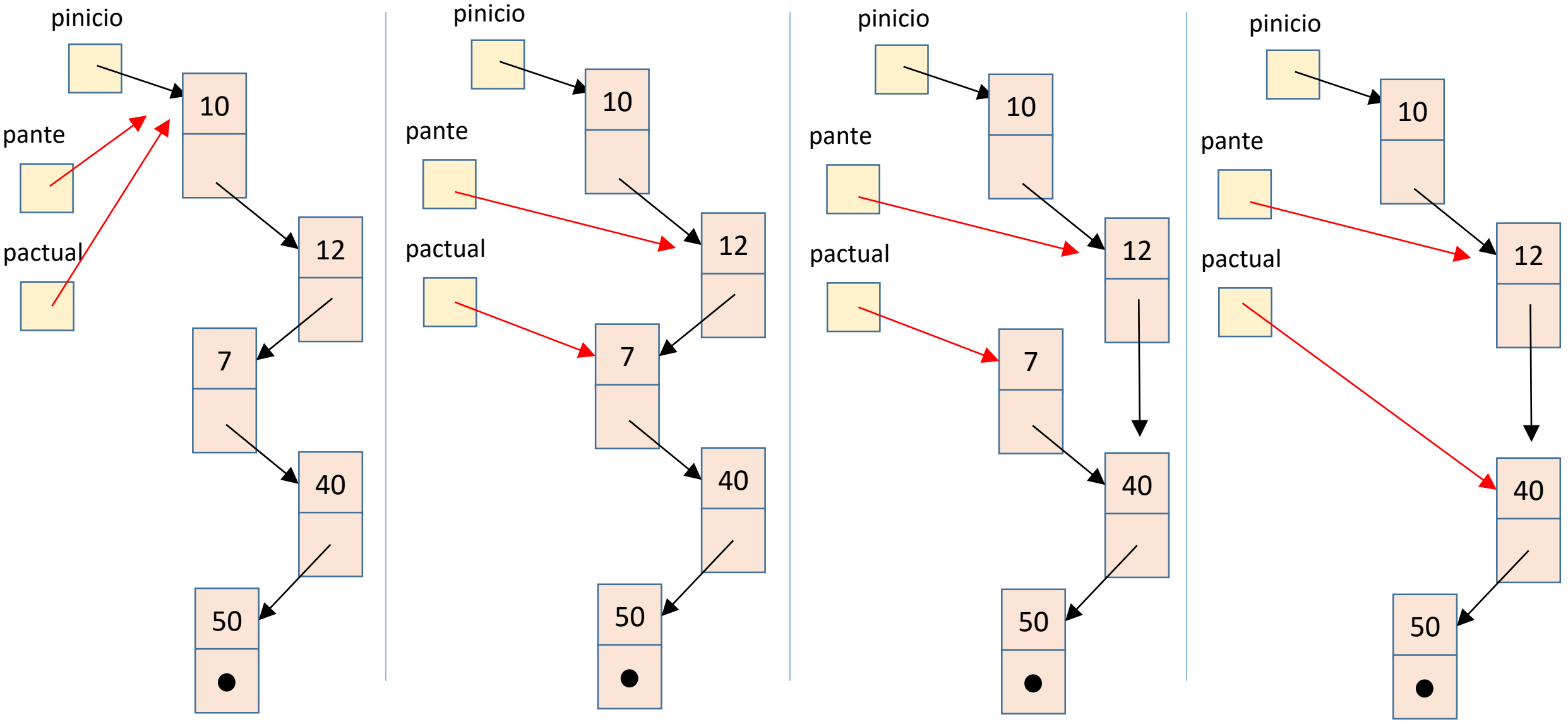




Eliminar un nodo de la lista

Construir una lista de numeros enteros tomados de un archivo de texto y luego eliminar de la lista todos los numeros primos.
El programa debe mostrar la lista inicial, y luego la lista despues de elimianar los valores primos. De no haber valores primos debe mostrar un mensaje indicando que no se encontraron numeros primos.

Eliminar un nodo intermedio

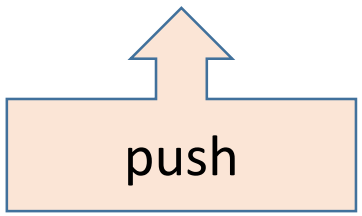


Estructura dinámica: Pila

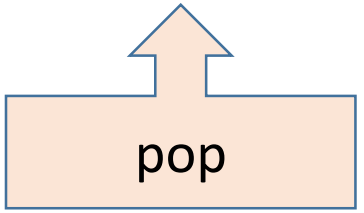
Una **pila** es una lista enlazada donde la agregación o eliminación de un nodo siempre es al principio de la lista. Ejemplo: una pila de platos.

Pila de platos

Añadir un plato a la pila



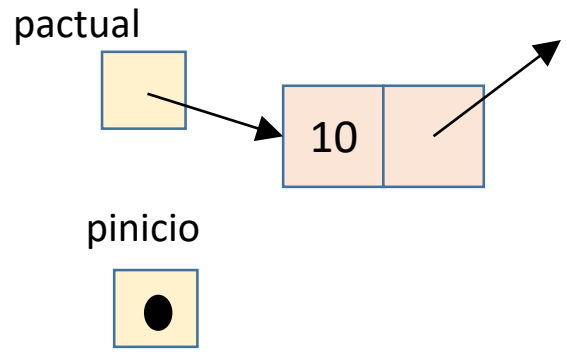
Retirar un plato de la pila

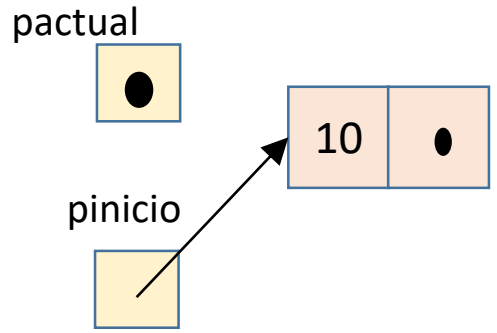


Agregar un nodo en la Pila

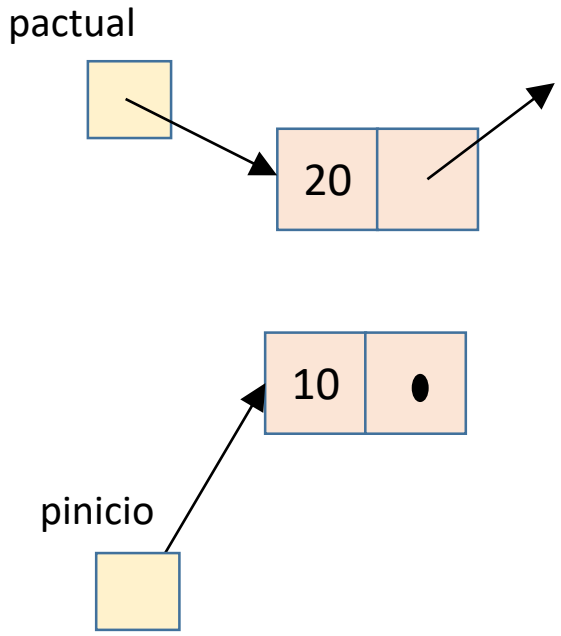
Cuando la pila esta vacia

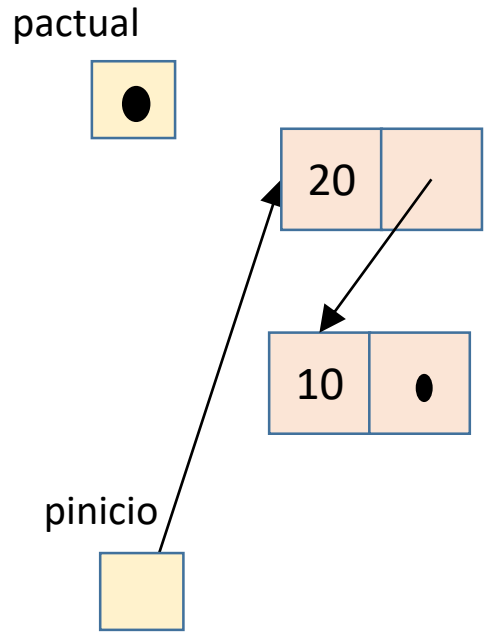
```
// PUSH Agregar un nodo a la pila
pactual=new reg_nodo;
pactual->numero=num;
if (pinicio==NULL) {
  pinicio=pactual;
  pinicio->psigue=NULL;
}
else {
  pactual->psigue=pinicio;
  pinicio=pactual;
}
pactual=NULL;
```





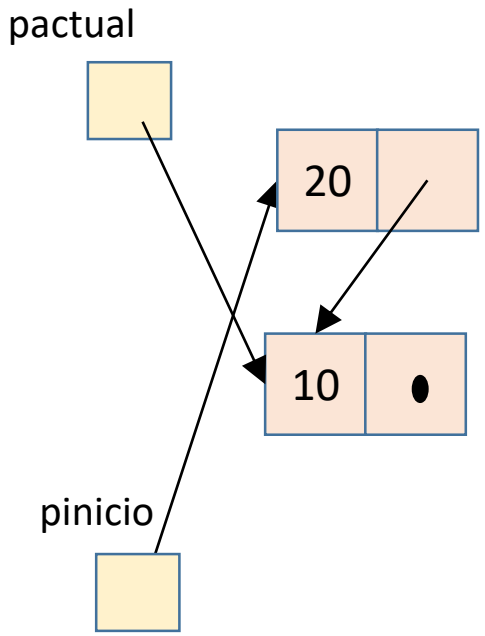
Cuando la pila tiene nodos

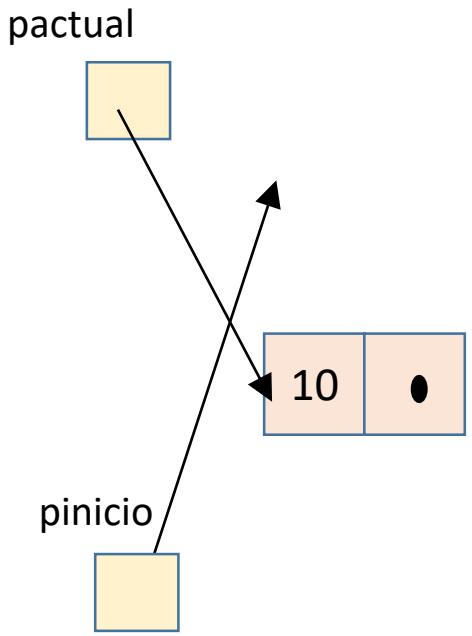


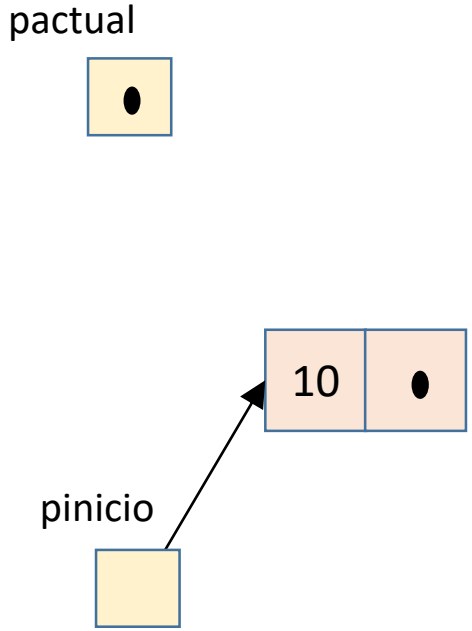


Eliminar un nodo de la pila

```
// POP eliminar un nodo de la pila
if (pinicio!=NULL){
  pactual=pinicio->psigue;
  delete pinicio;
  pinicio=pactual;
  pactual=NULL;
}
```







Problema para resolver con pila

Dada una cadena de texto que representa a una expresión con variables, constantes, operadores artiméticos y paréntesis de agrupación, hacer un algoritmo que determine si los paréntesis son correctos.

`3*(4-x)/2-(y-3)+2`

Paréntesis correctos

`3*(4-x/2-(y-3)-(2-x)`

`3*)4-x(/2-(y-3)+2`

Paréntesis incorrectos

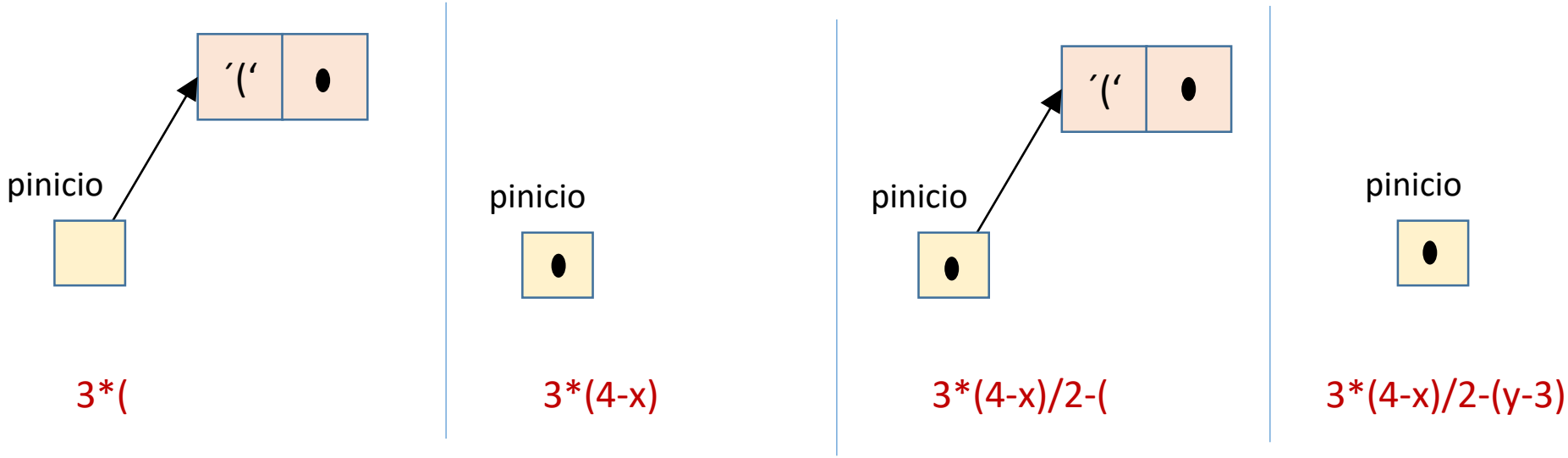
Problema para resolver con pila

Estrategia

Recorrer la cadena y agregar a la **pila** cada paréntesis de apertura. Cuando suceda un paréntesis de cierre retirar un paréntesis de la **pila**. Si al terminar de revisar la cadena, la **pila** queda vacia, los parentesis son correctos, de lo contrario son incorrectos. Si al suceder un paréntesis de cierre, la pila esta vacia, los paréntesis son incorrectos.

Primer caso

`3*(4-x)/2-(y-3)+2`

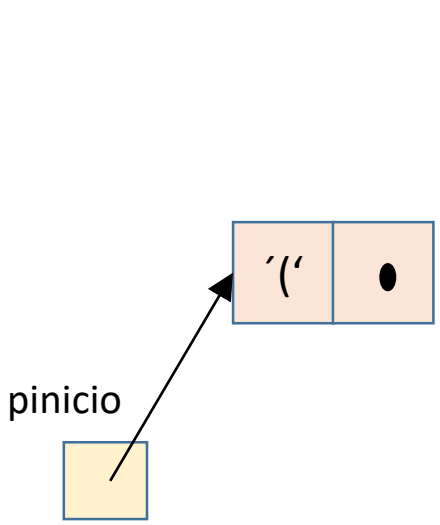


La **pila** queda vacia porque todos los paréntesis que se abren, se cierran correctamente, por tanto los paréntesis son correctos

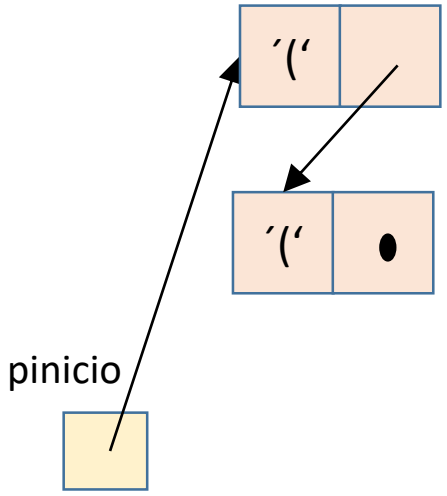
Problema para resolver con pila

Segundo caso

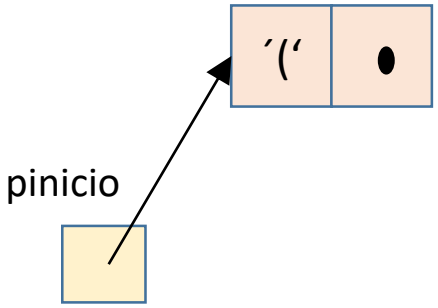
```
3*(4-x/2-(y-3)-(2-x)
```



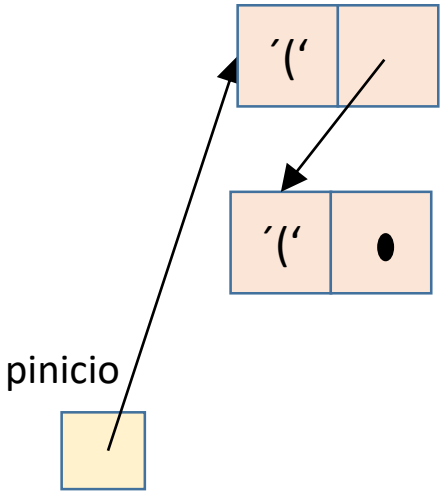
3*(



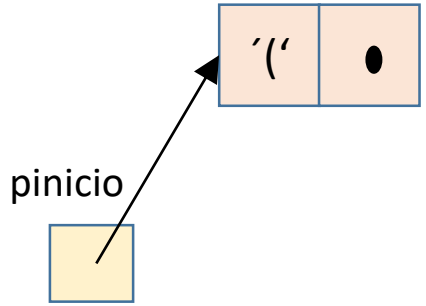
3*(4-x/2-(



3*(4-x/2-(y-3)



3*(4-x/2-(y-3)-(



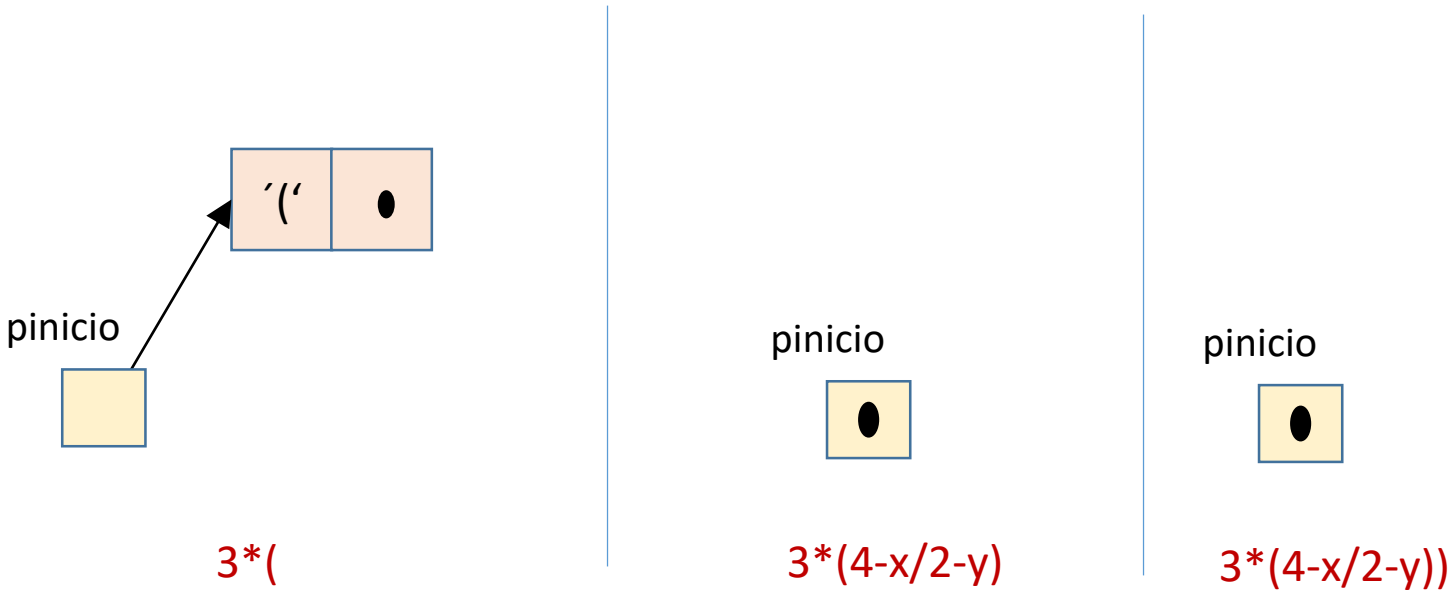
3*(4-x/2-(y-3)-(2-x)

La **pila** queda con un paréntesis de apertura sin cerrar,
por tanto los paréntesis son incorrectos

Problema para resolver con pila

Tercer caso

```
3*(4-x/2-y))+y-3-(2-x))
```

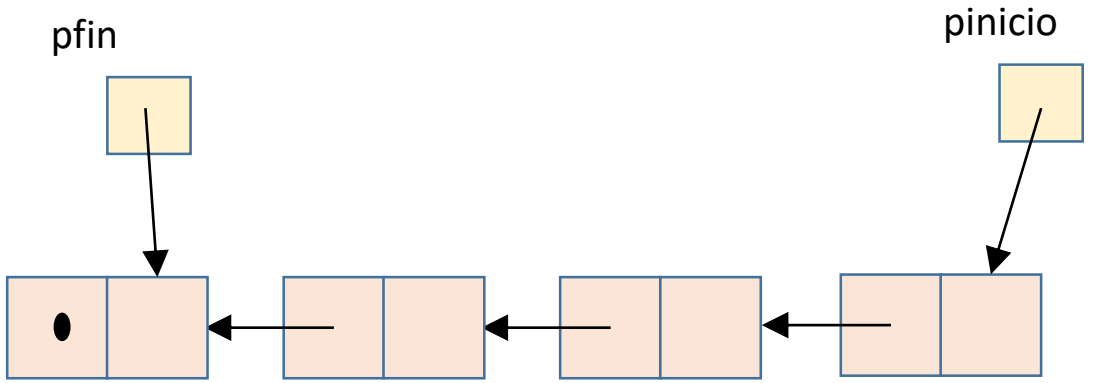


Cuando sucede el segundo paréntesis de cierre la pila esta vacia, lo que significa que no existe ningún paréntesis de apertura pendiente de cerrar. Por tanto, los paréntesis son incorrectos

Estructura dinámica: Cola

Una **cola** es una lista enlazada donde la agregación de un nodo es siempre al final de la lista y la eliminación es siempre del nodo inicial de la lista. Ejemplo: una cola de en el cajero automático.

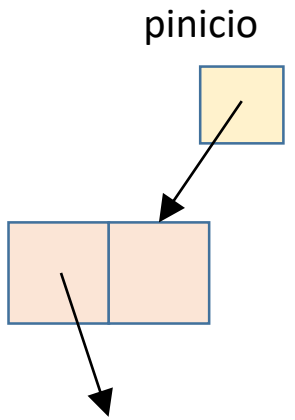
Adaptado de es.123rf.com

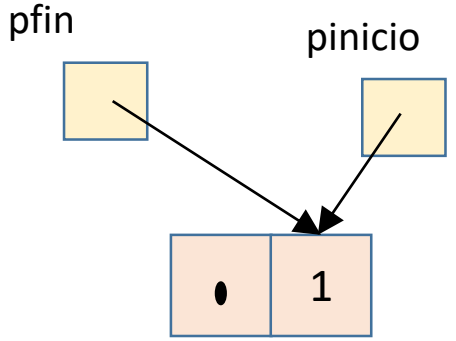


Agregar un nodo a la cola

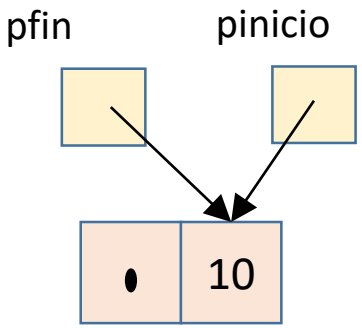
Cuando la cola esta vacia

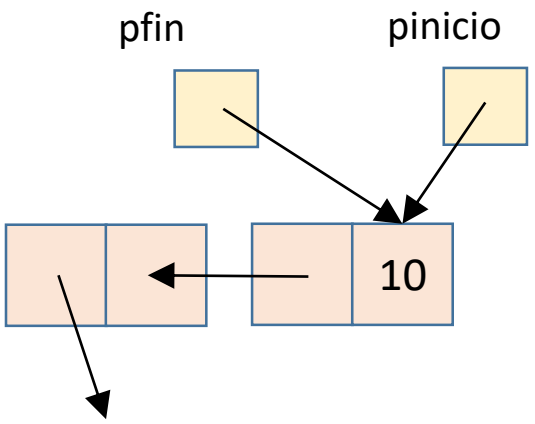
```
//Agregar un nodo a la cola
if (pinicio==NULL){
  pinicio=new reg_nodo;
  pinicio->numero=num;
  pinicio->psigue=NULL;
  pfin=pinicio;
}
else {
  pfin->psigue=new reg_nodo;
  pfin->psigue->numero=num;
  pfin->psigue->psigue=NULL;
  pfin=pfin->psigue;
}
```

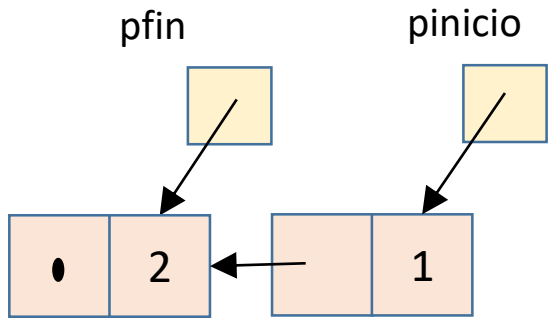




Cuando la cola tiene nodos

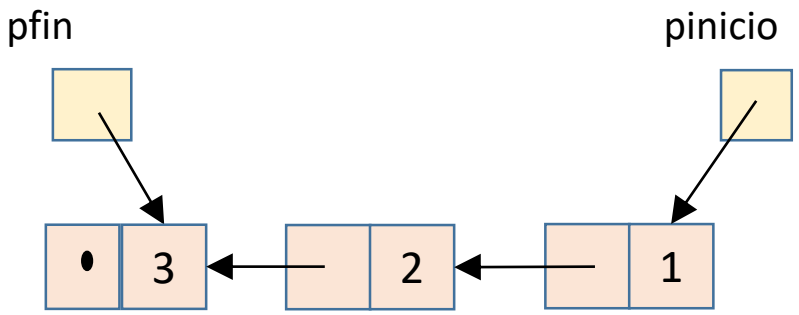


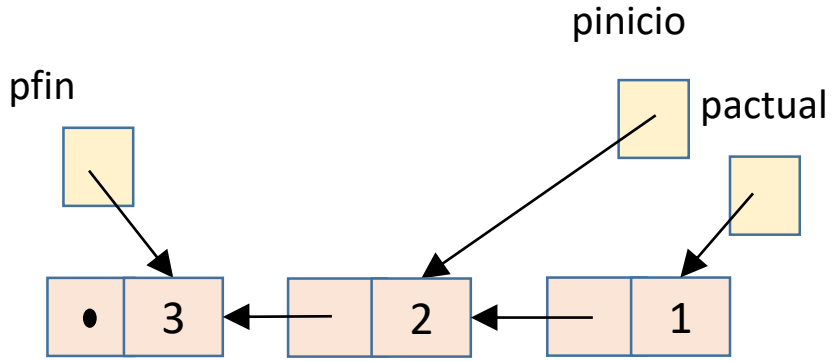


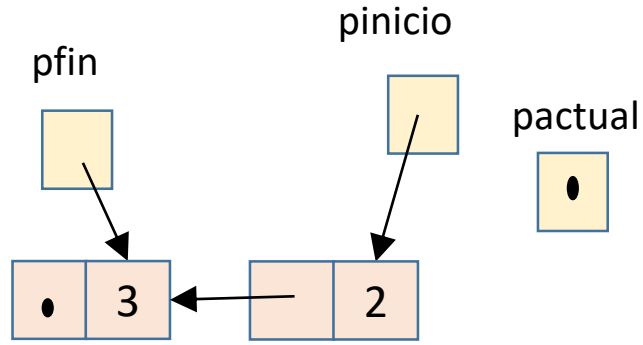


Retirar un nodo de la cola

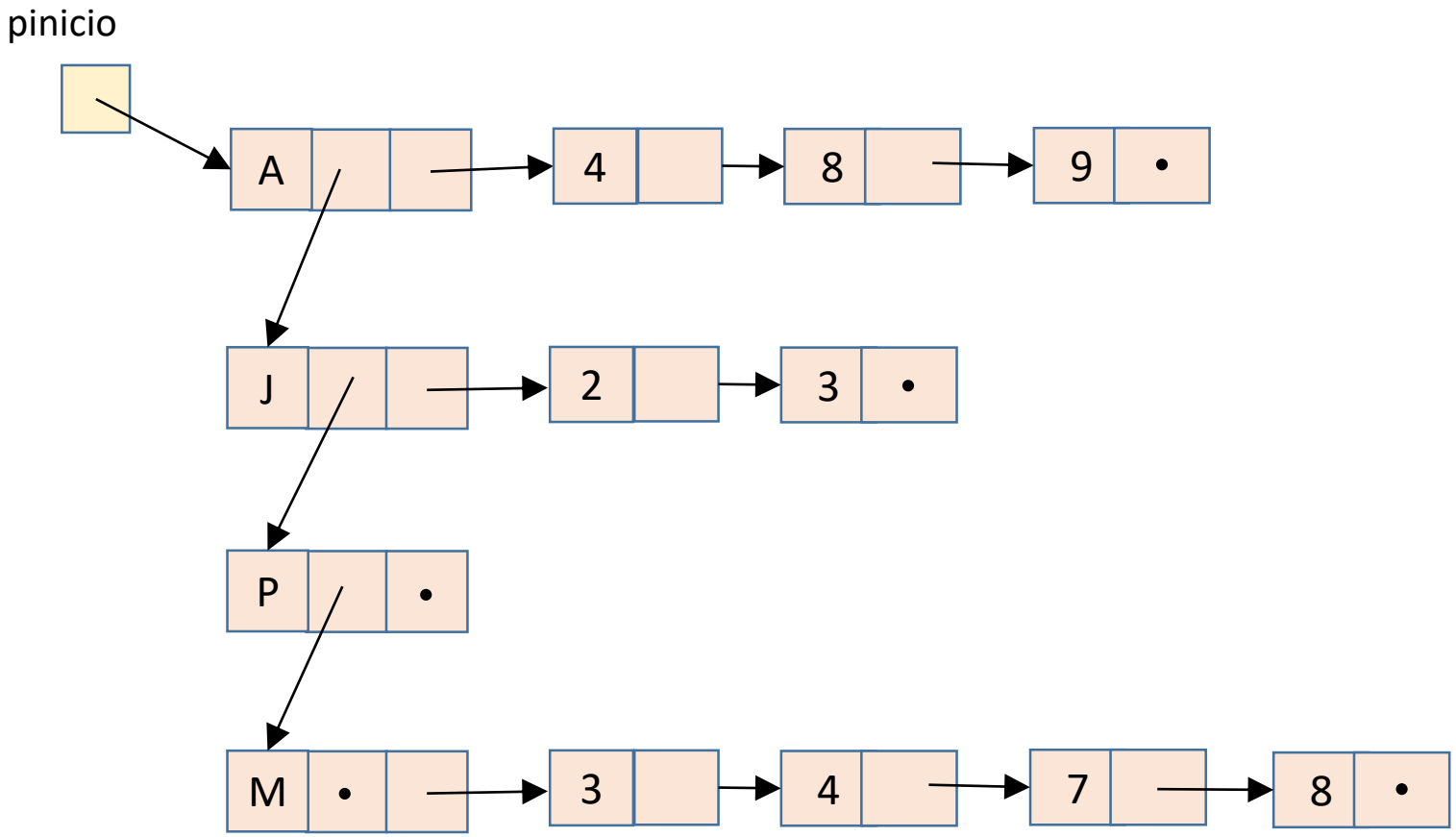
```
//Retirar un nodo de la cola
if (pinicio!=NULL){
  pactual=pinicio;
  pinicio=pinicio->psigue;
  delete pactual;
  pactual=NULL;
}
```



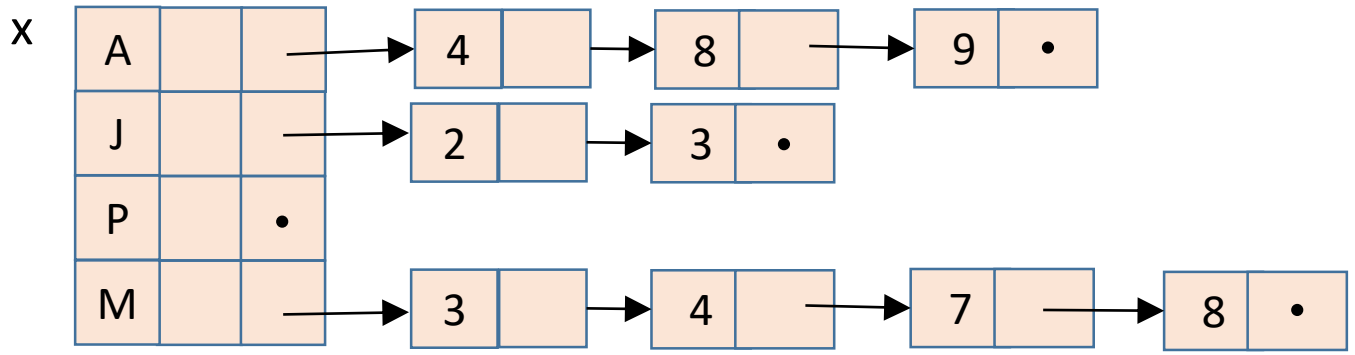




Combinación de estructuras: Lista de listas



Combinación de estructuras: Arreglo de Listas



Combinación de estructuras: Árbol binario