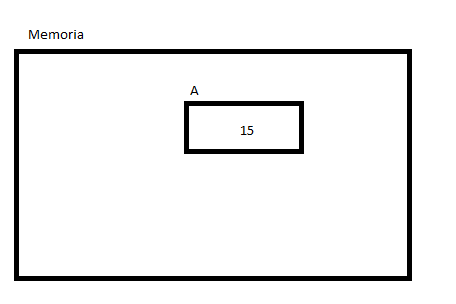
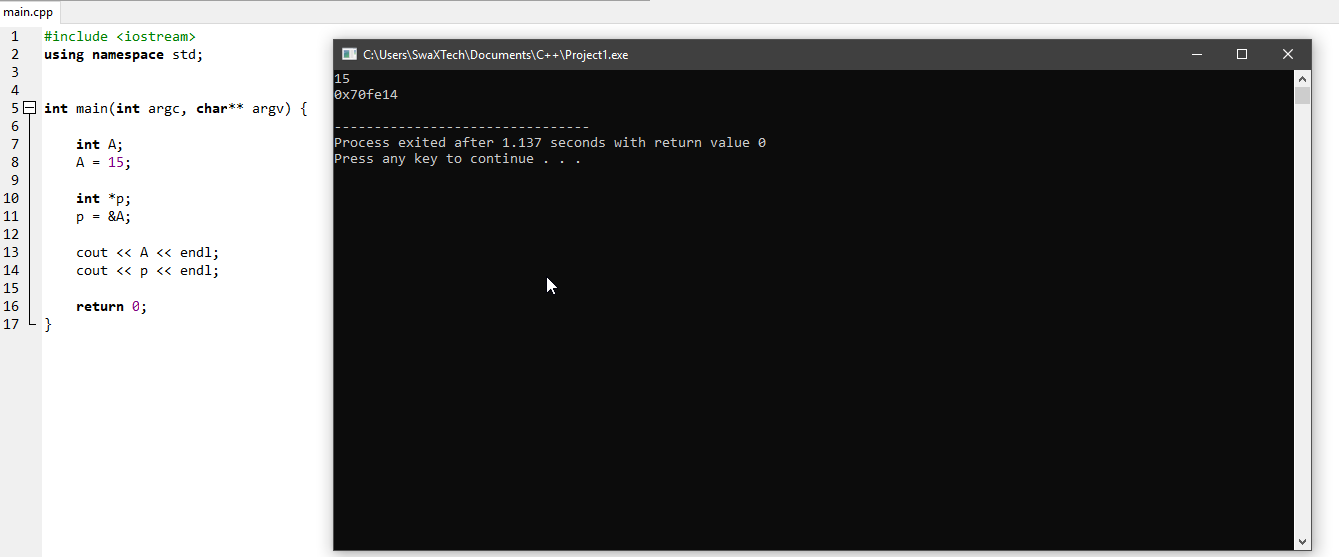
Punteros y Estructuras Enlazadas

Punteros



Es un tipo de dato, que almacena direcciones de memoria.

En el caso del dibujo, podemos declarar un puntero hacia la dirección de memoria A.



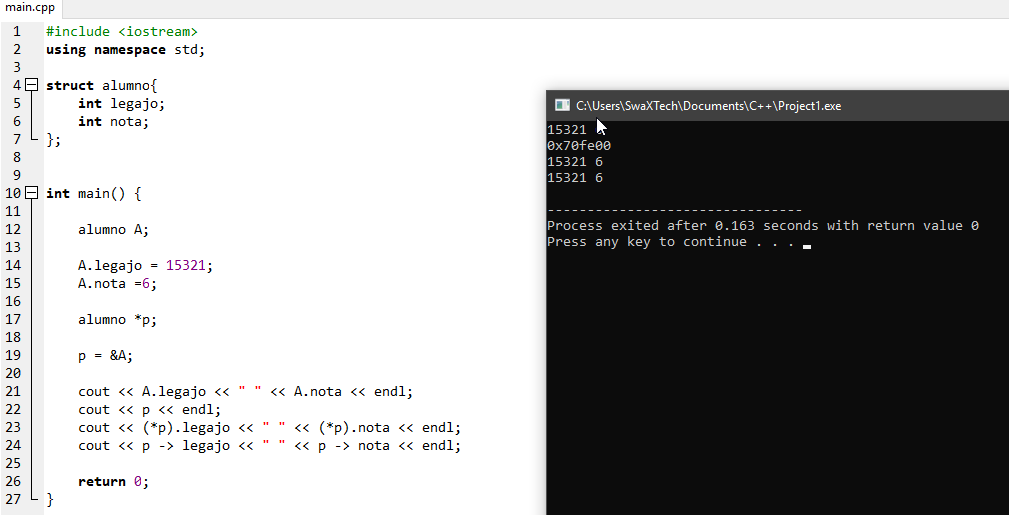
En la foto se pueden apreciar 2 operadores:

* **& (Ampersand)** 🡪 Es un operador de dirección, es decir, al anteponerlo al nombre de una variable, envío la dirección de memoria de esa variable.

En el caso de la foto, ‘A‘ vale 15, pero ‘&A’ (Que está almacenado en el puntero ‘p’) vale 0x70fe14, que es la dirección de memoria donde está almacenado el 15.

* **\*(Asterisco)** 🡪 Es un operador de indirección y también se usa para la declaración de punteros, al ponerlo luego del tipo de dato de una variable al ser declarada, esa variable pasa a ser un puntero a ese tipo de dato.

En el caso de la foto, ‘A’ es un entero (su declaración es int A;) pero ‘p’ es un puntero a un entero (su declaración es int\*p;)

Hay otros operadores que se utiliza al trabajar con registros (struct), que son el punto y la flechita:

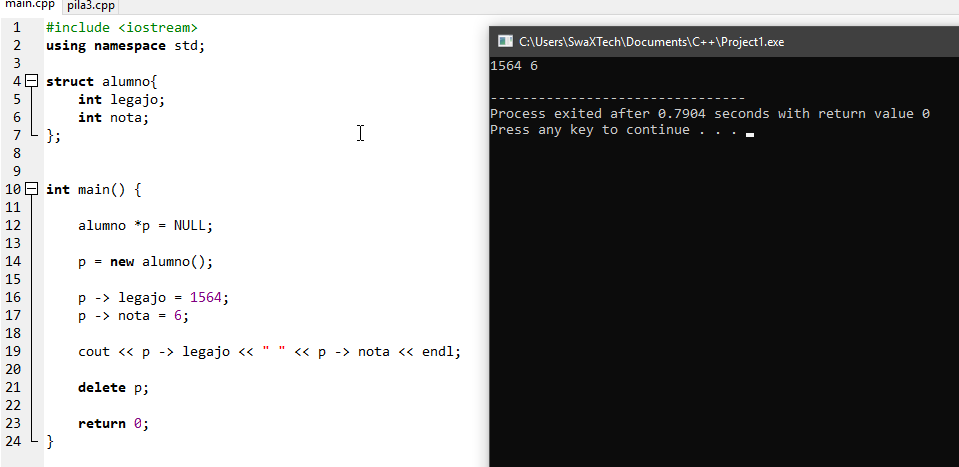
Hay dos formas de acceder al registro al que hace referencia un puntero

* Usando el operador de indirección (\*)
* Usando la flechita

Por comodidad se usa la flechita

Asignación dinámica

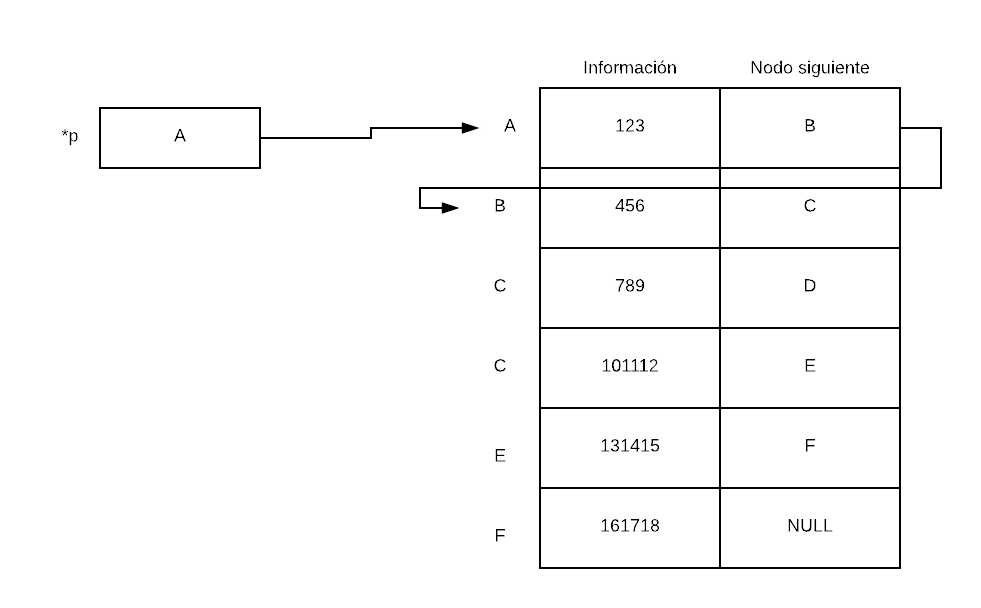
Para la asignación dinámica de valores, se utiliza el constructor ***new***, y el deconstructor ***delete,*** la característica que tiene la asignación dinámica es que usamos variables anónimas, nunca le damos un nombre como veníamos haciendo.

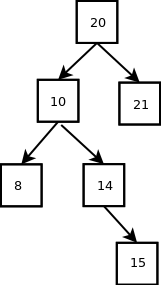


Nota: En la foto se usa la constante universal ‘NULL’, significa que el puntero no apunta a nada.

Con new, le asignamos un nuevo valor al puntero, y con delete lo destruimos.

Estructuras enlazadas

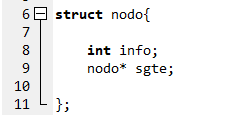
Son estructuras compuestas por nodos que poseen dos campos, uno con la información y otro con una referencia al nodo siguiente



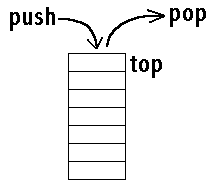
Las estructuras enlazadas se pueden organizar de forma:

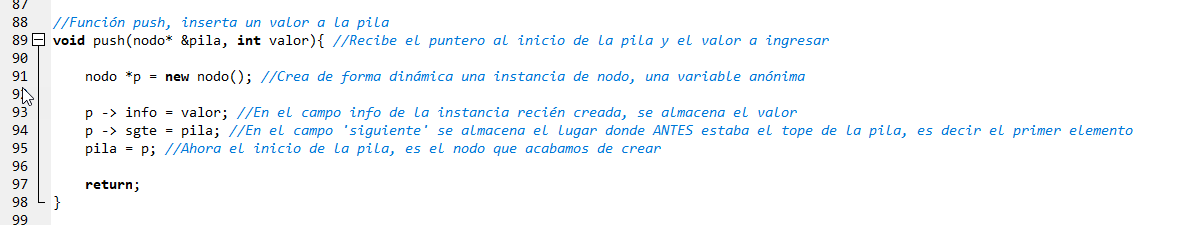
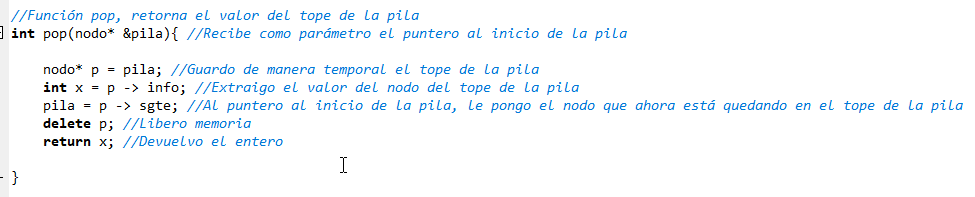
* Lineal
* Arboreas (Árbol)
* Grafos

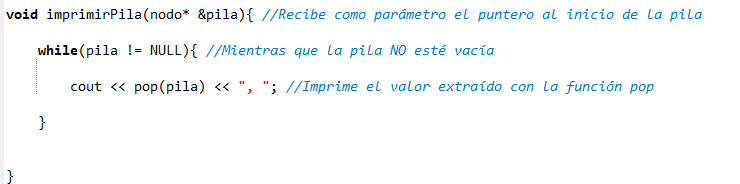
En la foto de la derecha se muestra un ejemplo de árbol, acomoda a su izquierda los valores menores y hacia la derecha los valores mayores

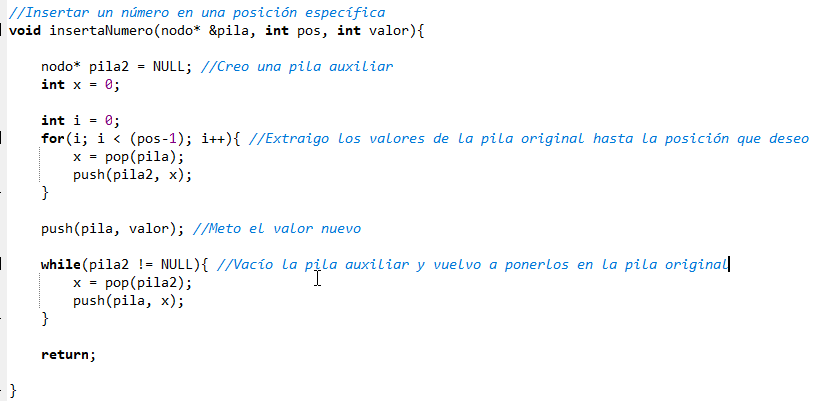
A diferencia de los vectores, con las estructuras enlazadas podemos formar ‘arreglos’ de datos cuyo tamaño varía en tiempo de ejecución.

Para las estructuras enlazadas siguientes consideramos la siguiente estructura:

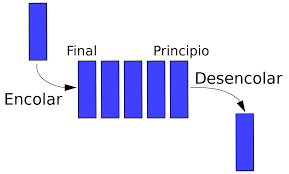
Pilas

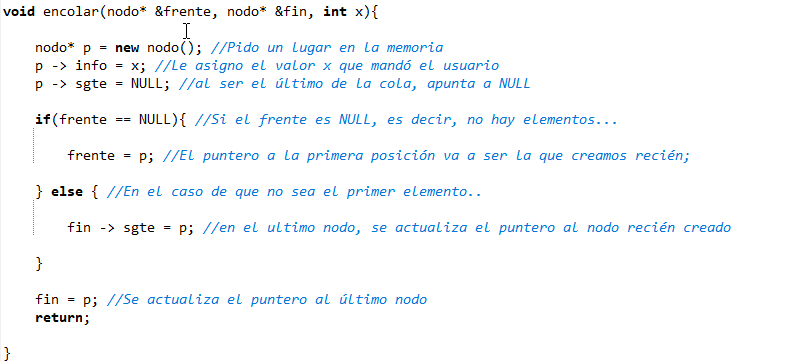
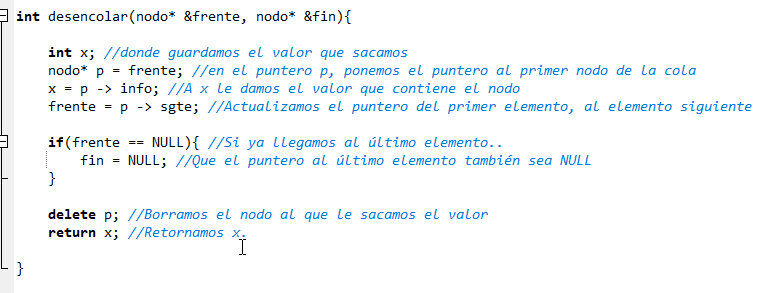
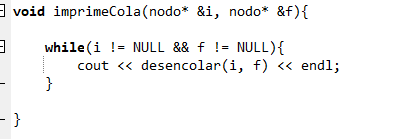
* Last in, first out -> El ultimo que ingresa es el primero que sale
* Usamos un puntero para indicar el inicio de la pila, es decir, el lugar donde se encuentra el último elemento ingresado
* Los datos salen en orden inverso al ingresado
* No puedo sacar un elemento de en medio de la pila, sin sacar todos los que están por encima antes.
* Las pilas NO se recorren
* Usamos las funciones push y pop



Nota: Sin el while, da error en tiempo de ejecución

Nota: Como es una pila, si quiero meter algo en el medio, tengo que sacar antes todo lo que hay encima, para eso lo pongo en otra pila auxiliar

Cola

* First int first out 🡪 El primero que ingresa es el primero que sale.
* Usamos un puntero para indicar el inicio de la cola y uno para indicar el final
* Los datos salen en el mismo orden que entraron
* Las colas NO se recorren
* Usamos las funciones encolar() y desencolar()

Listas

* Los nodos están ordenados
* Usamos un puntero al inicio de la lista
* Podemos buscar valores
* Podemos insertar valores en su lugar correspondiente según orden
* Podemos hacer listas de listas
* Podemos cargar valores si no están previamente
* Podemos sacar de una posición arbitraria
* Usamos principalmente buscar\_o\_insertar() y eliminaNodo();

