1. Implementa un algoritmo y el programa correspondiente para convertir las siguientes expresiones **infijas** a expresiones **prefijas** (usa el método de agrupar completamente) y evaluarlas dando valores a los operandos:

   (A+B)*(C+D)*(E+F)
   A+((B+C)*(D+E))
   A*B*C*D+E+F
2. Modifica el algoritmo de **evaluación en notación sufija** para que pueda manejar errores y que pueda manejar operandos de más de una cifra **int** o **float**
3. Evalúa las siguientes expresiones **sufijas**. Muestra la pila a medida que es procesado cada operando y cada operador.

   2.5 31 * 4 +
   1.5 22 + 3 + 4 + 5 +
   123 2.5 3.1 4 5 * + * +
4. Implementa como un nuevo método de la clase, la **concatenación** de dos colas para constituir una nueva
5. **Problema de Simulación: Tareas de impresión** Disponemos en un laboratorio de una impresora antigua capaz de procesar 10 páginas por minuto en calidad de borrador. Suele haber 10 usuarios por hora, imprimiendo hasta 2 veces en ese tiempo, y la extensión de estas tareas oscila entre 1 y 20 páginas. ¿Podrá la impresora actual manejar la carga de tareas si fuera ajustada para imprimir con una mejor calidad, pero con una tasa de página más lenta? (Tema2_12.py y Tema2_12.cpp) Simular distintas situaciones de la impresora
6. Implementar el **TAD Impresión** usando **Colas de Prioridad**
7. Realiza la especificación informal del TAD **Cola Doble**
8. Implementar el TAD **Colas de Prioridad**, usando listas enlazadas ordenadas
9. Escribe la especificación informal del TAD **Conjunto**. Enriquece la clase Conjunto con:
   - Un método ***elimina*** que borre del conjunto un elemento dado
   - Un método ***unión*** que devuelva el conjunto resultante de unir dos conjuntos
   - Un método ***intersección*** que devuelva un conjunto con la intersección de dos conjuntos
   - Un método ***diferencia*** que devuelva un conjunto con la diferencia entre dos conjuntos
   - Un método ***incluye*** que consulta si un conjunto dado está incluido en el conjunto
10. Implementar el método **anexar una nueva lista** para **ListaNoOrdenada**. ¿Cuál es la complejidad de tiempo del método que creaste?
11. Implementar los métodos no desarrollados en el TAD **Lista_No_Ordenada**, así como los métodos **fin**, **primero**, **siguiente** y **anterior**
12. Implementar los métodos no desarrollados en el TAD **ListaOrdenada**, así como los métodos **borrar**, **indice**, **extraer** y **extraer(pos)**
13. ¿Cuál es el resultado de ejecutar en orden inverso los dos pasos del método **agregar** de la **Lista_No_Ordenada**? ¿Qué tipo de referencia resultaría? ¿Qué tipos de problemas pueden resultar?
14. Para implementar el método **tamano** contamos el número de nodos en la lista. Una estrategia alternativa sería almacenar el número de nodos en la lista como una pieza de datos adicional en la cabeza de la lista. Modifica la clase **ListaNoOrdenada** para incluir esta información y reescribe el método **tamano**. ¿Qué complejidad tiene el método **tamano** ahora?

Exámenes años anteriores

La empresa **Entrepinares** nos ha encargado la programación de un robot para el almacén de producto finalizado. En concreto se trata de crear un **TAD lista enlazada circular** de posiciones dadas por (X,Y,Z) [**con origen en (X_0,Y_0,Z_0)**] en las que el robot ha de coger una pieza de la posición **(X_1,Y_1,Z_1)**, elevarla en el eje Z, trasladarla a otra posición y, bajando en el eje Z, depositarla en **(X_2,Y_2,Z_2)**, repitiendo esta operación en posiciones **(X_3,Y_3,Z_3)** → **(X_4,Y_4,Z_4)**, **(X_5,Y_5,Z_5)** → **(X_6,Y_6,Z_6)** y **(X_7,Y_7,Z_7)** → **(X_8,Y_8,Z_8)**, volviendo a continuación a la posición inicial.

Implementar el **TAD Lista enlazada circular** en Python (**especificación informal incluida**) y codificar un ejemplo que use dicha implementación. Una lista enlazada circular es una variación de la lista enlazada en la que sus nodos están conectados de tal manera que forman un círculo, es decir, el siguiente puntero del último nodo no se establece en nulo, sino que contiene la dirección del primer nodo, formando así un círculo.

Coger_pieza(accion,posicion)

- **efecto:** Cierra el mecanismo para coger una pieza.

Dejar_pieza(accion,posicion)

- **efecto:** Abre el mecanismo para dejar una pieza

Go_to(accion,posición)

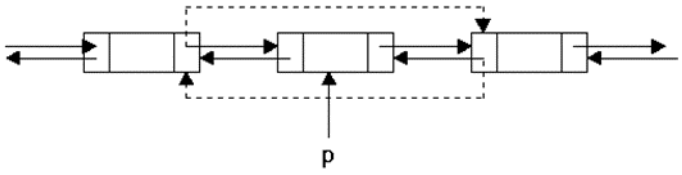
- **efecto:** Posiciona el robot en la posición dada, elevando en el eje Z y posteriormente, bajando en el mismo eje.

Simular el funcionamiento de estas tres operaciones simplemente con un **print(acción,posicion)**, donde acción será una de estas tres:

- **coger pieza de**
- **dejar pieza en**
- **trasladándose a**

Problema

Implementar el **TAD cola de prioridad** usando tuplas **(clave, prioridad),** mediante una lista ordenada doblemente enlazada.



Problema

La implementación del **TAD lista enlazada no ordenada** se ha desarrollado en los apuntes de la asignatura.

Una implementación alternativa se conoce como una **lista doblemente enlazada**. En esta implementación, cada nodo tiene una referencia al siguiente nodo (comúnmente llamado **siguiente**) así como una referencia al nodo precedente (comúnmente llamado **anterior**). La

Cabecera de la lista también contiene dos referencias, una al primer nodo en la lista enlazada y una al último.

Implementar el **TAD Lista Doblemente enlazada** en Python (**especificación informal incluida**) y codificar un ejemplo que use dicha implementación.

Problema

1. Implementar el TAD Pila descrito en la página 5 y 6 del tema de Estructuras Lineales, usando para ello **listas enlazadas no ordenadas**. En concreto hay que implementar las operaciones siguientes

    a. **Pila():** crea una nueva pila que está vacía. No necesita parámetros y devuelve una pila vacía.

    b. **print(P):** muestra todos los elementos de la pila **P**.

    c. **incluir(item):** agrega (**apila**) un nuevo ítem en el tope de la pila. Requiere el ítem y modifica la pila "apilándolo".

    d. **extraer():** elimina (**desapila**) el ítem en el tope de la pila. No requiere parámetros y devuelve el ítem. La pila se modifica desapilando el item.

    e. **inspeccionar():** devuelve el ítem en el tope de la pila pero no lo elimina.

    f. **estaVacia():** comprueba si la pila está vacía. No requiere parámetros y devuelve un valor booleano.

    g. **tamano():** devuelve el número de ítems en la pila. No requiere parámetros y devuelve un entero.

    Desarrollar un programa en Python que demuestre su uso.