J.P. Vidocevich / I. Sánchez
Facultad de Ingeniería y Ciencias Hídricas
Universidad Nacional del Litoral

GTP 2 — Guía de Trabajos Prácticos 2 - Programación 2/3

1. Pilas y Colas

1. **Clase stack:** Implementar la clase `stack` que implementa una pila a partir de nodos enlazados. Implementar los métodos públicos:

   a) `push(e)`: anexa el elemento `e` en el tope de la pila

   b) `pop()`: quita el elemento en el tope de la pila y lo devuelve, genera una excepción si la pila está vacía.

   c) `top()`: devuelve el elemento en el tope de la pila sin eliminarlo, genera una excepción si la pila está vacía.

   d) `is_empty()`: retorna True si la pila no contiene ningún elemento.

   e) `len(S)`: retorna el numero de elementos en la pila S.

   Implementación de la excepción: Definimos la excepción trivial `Empty` que extiende la clase básica de python `Exception` y es la que debe usarse en los casos de llamar a `pop()` o `top()` sobre una pila vacía:

   ```
   class Empty(Exception):
       """Error attempting to access an element from an empty container."""
       pass
   ```

   Las excepciones en python se lanzan con el comando `raise(Excepcion(mensaje)`, donde `Excepcion` es la clase de excepción que se desea lanzar y mensaje es una cadena que informa al usuario respecto del error o evento que tuvo lugar. Para el ejercicio, la llamada sería:
   `raise Empty('La pila está vacía')`

   Implementación de método `len`: La función `len(S)` de Python llamará al método `__len()__` que deberá ser definido en la clase stack. (no hay métodos privados en las clases de python, se suele indicar esta condición definiendo el nombre del método antecedido por `__`. Si además está sucedido por `__`, se suele tratar de un método incorporado en el intérprete de Python).

2. **Varios:** Utilizando las operaciones de la clase stack definida en el ejercicio 1, implementar procedimientos que realicen cada una de las actividades siguientes:

   a) Asignar i al segundo elemento desde la parte superior de la pila, descartando los dos elementos preexistentes en la pila.

   b) Asignar i al segundo elemento desde la parte superior de la pila, sin modificar el resto de sus elementos

   c) Dado un entero n, asignar i al elemento n-ésimo desde la parte superior de la pila, dejando la pila sin sus n elementos superiores.

   d) Asignar i al elemento del fondo de la pila, dejando el resto de la pila sin modificar.

   e) Insertar i como un elemento adicional en el fondo de la pila, dejando el resto de la pila sin modificar.

3. **Clase queue:** Implementar la clase `queue` que implementa una cola a partir de nodos enlazados. Implementar los métodos públicos:

a) `push(e)`: anexa el elemento `e` en el final de la cola

b) `pop()`: quita el elemento en el primer lugar de la cola y lo devuelve, genera una excepción si la cola está vacía.

c) `head()`: devuelve el elemento en el primer lugar sin eliminarlo, genera una excepción si la cola está vacía.

d) `is_empty()`: retorna True si la cola no contiene ningún elemento.

e) `len(Q)`: retorna el numero de elementos en la cola Q.

4. **Varios cola:** Utilizando las operaciones de la clase `queue` definida en el ejercicio 3, implementar procedimientos que realicen cada una de las actividades siguientes:

   a) Asignar i al segundo elemento de la cola, descartando los dos elementos preexistentes.

   b) Asignar i al segundo elemento de la cola, sin modificar el resto de sus elementos.

   c) Dado un entero n, asignar i al elemento n-ésimo de la cola, eliminando los elementos antecesores.

   d) Asignar i al último elemento de la cola, dejando el resto sin modificar.

   e) Insertar i como un elemento adicional en el primer lugar de la cola, dejando el resto de la cola sin modificar.

5. **Inverso**. Escribir un procedimiento `inverso(s)` para determinar si una cadena de caracteres de entrada es de la forma z=xy donde y es la cadena inversa (o espejo) de la cadena x, ignorando los espacios en blanco. Emplear una cola y una pila auxiliares.

6. **Chequeo**. Escribir un segmento de programa que lea expresiones aritméticas del estilo `((a - b)*(5 - c))/4` y verifique si los paréntesis están embebidos correctamente. Es decir, debemos verificar si:

   a) Existe igual número de paréntesis a izquierda y derecha;

   b) Cada paréntesis de la derecha está precedido de su correspondiente paréntesis a la izquierda.

7. **SortStack**. Escribir una función `sort(L)`, que ordena los elementos de L de mayor a menor. Para ello emplear el siguiente algoritmo simple, utilizando una pila auxiliar P: ir tomando el menor elemento de L, eliminarlo de L e insertarlo en P hasta que L este vacía. Luego insertar los elementos de P en L.

8. **SortQueue**. Escribir una función `sort(L)`, que ordena los elementos de L de menor a mayor. Para ello utilizar el siguiente algoritmo simple, utilizando una cola auxiliar C: ir tomando el menor elemento de L, eliminarlo de L e insertarlo en C hasta que L este vacía. Luego insertar los elementos de C en L.

9. **PancakeSort**. Dada una pila de números S, implementar la función `pancakesort(S)` la cual ordena la pila solo haciendo operaciones en las cuales se invierte un rango contiguo de elementos en el tope de la pila.

10. **BurntPancakeSort**. Dada una pila de números S, implementar la función `burntpancakesort(S)` la cual ordena la pila solo haciendo operaciones en las cuales se invierte un rango contiguo de elementos en el tope de la pila teniendo en cuenta de que cada elemento debe ser invertido un número par de veces (la cara quemada de cada panqueque debe quedar hacia abajo).

11. **Rotación**. Escribir una función `rotacion(C)`, la cual saca una cierta cantidad de enteros del frente de la cola C y los vuelve a insertar en fin de la misma, de tal manera que quede en el frente de cola un número par. Por ejemplo, si C = [1, 3, 5, 2, 4] entonces, después de rotacion(C), debe quedar C = [2, 4, 1, 3, 5].