FaMAF. Algoritmos y Estructuras de Datos 2 - 2019. Proyecto 5 – TAD Stack

Duración: 2 clases

En este proyecto se deberá implementar el TAD Stack usando listas enlazadas de nodos. Como aplicación del TAD, se implementará un comando *reverse* para invertir una cadena de caracteres. También se pide utilizar el stack para resolver el problema de las Torres de Hanoi.

Ejercicio 1: Implementación del TAD Stack

Como primer paso definir la estructura **struct _stack** en el archivo *stack.c* guiándose por las slides del teórico (ver la slide con título "Implementación del TAD pila con listas enlazadas") (la estructura **_stack** es la correspondiente a *node* en el teórico)

Los elementos de la pila serán números enteros (ver la definición del tipo *stack_element_t)*
¿Por qué la definición de la estructura debe ir en stack.c y no en stack.h?
Luego continuar con la implementación de las funciones declaradas en *stack.h.*

Crear una pila vacía:
stack_t stack_empty()

Insertar un elemento al tope de la pila:
stack_t stack_push(stack_t s, stack_elem_t elem)

Remover el tope de la pila:
stack_t stack_pop(stack_t s)

Obtener el tamaño de la pila:
unsigned int stack_size(stack_t s)

Obtener el tope de la pila, sin remover. Sólo aplica a una pila no vacía; usar la función *assert* de la librería *assert.h* para verificar esa precondición:
stack_elem_t stack_top(stack_t s)

Verificar si la pila es vacía.
bool stack_is_empty(stack_t s)

Crear un arreglo con todos los elementos de la pila. El tope de la pila debe quedar en el último elemento del arreglo. Es decir, leyendo el arreglo de derecha a izquierda se obtiene la pila original. Si la pila es vacía, devuelve NULL. Para crear el arreglo nuevo usar *calloc* de la librería *stdlib.h*
stack_elem_t *stack_to_array(stack_t s)

Liberar todos los nodos de la pila usando *free* de la librería *stdlib.h*
stack_t stack_destroy(stack_t s)

Repasar punteros antes de empezar!
Se sugiere crear un archivo de prueba *test.c* para verificar las funciones en casos extremos:
- ¿Funciona bien stack_pop para pilas de tamaño 1?
- Si la pila queda vacía, ¿puedo volver a insertar elementos?
- ¿La función stack_to_array devuelve NULL para una pila vacía? ¿Devuelve los elementos en el orden correcto?

Ejercicio 1* opcional: Cambiar la implementación para lograr que *stack_size* sea O(1) respecto al número de nodos. Mantener un contador dentro de una estructura principal, separada de la estructura de nodos.

Ejercicio 2: String reverse
Como un ejemplo simple para verificar la implementación de la pila, se pide el siguiente ejercicio. En la función *main* del archivo *reverse.c* escribir un algoritmo que utilice el TAD Stack para invertir una cadena de caracteres. Se espera que el programa funcione por línea de comandos de la siguiente manera:

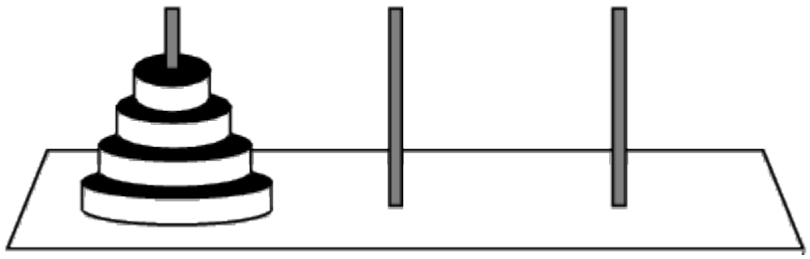
$. /reverse hola
aloh

El único parámetro del comando es la palabra que se quiere invertir. El algoritmo se puede
describir como sigue:

1. Crear una pila vacía
2. Iterar sobre el string (*char *)* de izquierda a derecha, insertando los caracteres uno por uno en la pila.
3. Extraer cada uno de los caracteres de la pila, usando *stack_top* para obtenerlos y *stack_pop* para removerlos.
4. Construir un string nuevo con los caracteres obtenidos.

Se pueden utilizar las funciones de *string.h* si fuese necesario. Notar que si bien nuestra pila acepta números enteros como elementos, es posible insertar caracteres utilizando *type casting*. *Nota:* Otra opción más recomendable es cambiar la definición de *stack_elem_t,* pero en ese caso deberán recompilar el stack.

Ejercicio 3: Torres de Hanoi
En el clásico juego de las torres de Hanoi, se tienen tres torres y N discos de distintos tamaños, que inicialmente se encuentran como lo muestra la siguiente imagen:



Se desean mover los N discos a la tercera torre, usando la torre del medio como auxiliar. Sólo es posible mover un disco a la vez, y nunca de debe apoyar un disco grande sobre uno más pequeño. Ver wikipedia para una descripción más completa del juego y su historia.
Se pide crear un programa que muestre en pantalla todos los movimientos necesarios para lograr el objetivo. El argumento del programa es el número de discos a utilizar.

$ ./solve-hanoi 3

En la implementación que se encuentra en *hanoi.c,* cada torre se representa con una pila. Cada disco se representa con un número entero que indica su tamaño. Inicialmente usamos *hanoi_init* para construir las tres pilas, la primera tiene como elementos la secuencia [4,3,2,1] (asumiendo N=4 como en la imagen anterior) y el resto son pilas vacías.
La función *hanoi_solve* debe mostrar en pantalla el estado de las torres luego de cada movimiento. El código de esta función está incompleto, pero el resto de las funciones se entrega implementado. Se pide programar la solución recursiva, descripta por el siguiente pseudocódigo:

```
mover( N, Source, Target, Auxiliar):
        if N > 0:
                mover(N – 1, Source, Auxiliar, Target)  // Mover N – 1 discos de Source a Auxiliar
                Target.push(Source.top()) // Mover el disco que queda a Target
                Source.pop()
                mover(N – 1, Auxiliar, Target, Source) // Mover N - 1 discos de Auxiliar a Target
        endif
```

La implementación en Python que tiene wikipedia se puede usar como guía.
Se provee una función para “dibujar” las torres en consola, pero pueden modificarla a gusto!
Ejercicio 3* opcional: Implementar además alguna de las soluciones no recursivas.

Ejercicio 4. Verificar leaks de memoria.
Para verificar que ninguno de los programas tiene *memory leaks* usaremos la herramienta valgrind.

$ valgrind --leak-check=full ./reverse hola
$ valgrind --leak-check=full ./solve-hanoi 10
El reporte de la herramienta debe mostrar que no hay bytes perdidos de memoria. Ejemplo:
==7929== HEAP SUMMARY:
==7929== in use at exit: 0 bytes in 0 blocks
==7929== total heap usage: **244** allocs, **244** frees, 1,928 bytes allocated
==7929== **All heap blocks were freed -- no leaks are possible**

*También deben corregir los errores (**Invalid reads/writes**) si los hubiera.*
Agregar una regla en Makefile para ejecutar valgrind.