Examen Final del Taller - Regulares

Algoritmos y Estructuras de Datos II

Tarea

El alumno deberá implementar el tipo abstracto de dato *StackCalc* en el lenguaje de programación C, utilizando la técnica de ocultamiento de información visto en el taller de la materia.

Es requisito mínimo para aprobar lo siguiente:

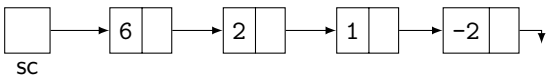
- Implementar en C el TAD *StackCalc* (utilizando punteros a estructuras y manejo dinámico de memoria).
- Implementar en C una función `main`, detallada más abajo.
- El programa resultado no debe tener *memory leaks* ni accesos inválidos a memoria, se chequeará tal condición usando `valgrind`.
- Las funciones deben ser **NO** recursivas.

El TAD StackCalc

El tipo *StackCalc* representa una "pila de cálculo", cuya definición formal se dio en el exámen teórico (ver apéndice al final). La implementación a utilizar será la siguiente:

```
type stack_calc = pointer to node
type node = tuple
              value: int
              next: pointer to node
            end
```

Por ejemplo, la siguiente es la representación gráfica de una pila `sc` de 4 enteros:

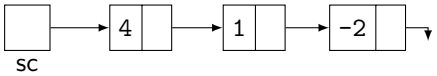


Funciones del TAD

La siguiente es la lista de funciones que debe proveer el TAD:

- `sc_t sc_empty(void)`. Devuelve una pila vacía.
- `sc_t sc_push(sc_t sc, int k)`. Inserta un nuevo entero `k` en el tope de la pila.
- `int sc_top(sc_t sc)`. Devuelve el primer elemento de la pila.
- `sc_t sc_pop(sc_t sc)`. Elimina el tope de la pila.
- `bool sc_is_empty(sc_t sc)`. Devuelve un booleano que indica si la pila es vacía o no.
- `unsigned int sc_size(sc_t sc)`. Devuelve el tamaño de la pila.

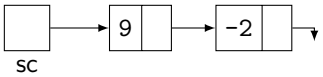
- `sc_t sc_minus(sc_t sc)`. Reemplaza el tope de la pila por la diferencia de los dos primeros elementos. Por ejemplo, si aplicamos `sc = sc_minus(sc)` (donde `sc` es la pila de ejemplo) obtenemos:



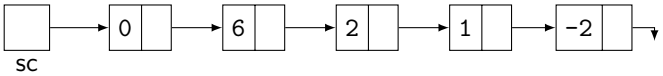
  Esta función asume que la pila tiene al menos 2 elementos.

- `sc_t sc_sum(sc_t sc, unsigned int n)`. Elimina los primeros `n` elementos de la pila, y luego inserta en el tope la suma de esos elementos.

  Por ejemplo, si aplicamos `sc = sc_sum(sc, 3)` obtenemos:



  mientras que si en lugar de eso aplicamos `sc = sc_sum(sc, 0)` obtenemos



  Esta función asume `n <= sc_size(sc)`.

- `void sc_dump(sc_t sc, FILE *fd)`. Imprime la pila en el archivo `fd`.

  Por ejemplo, `sc_dump(sc, stdout)` debe mostrar en pantalla:

  ```
  6, 2, 1, -2
  ```

- `sc_t sc_destroy(sc_t sc)`. Destruye la pila, liberando todos los recursos de memoria.

Función Main

Escribir una función `main` que realice en orden los siguientes pasos:

1. Crear una pila vacía `sc`, e imprimir `sc` en pantalla.
2. Agregar los números del 1 al 10 en forma ascendente en la pila `sc` e imprimir `sc`.
3. Llamar a `sc = sc_minus(sc)` e imprimir `sc`.
4. Llamar a `sc = sc_sum(sc, 3)` e imprimir `sc`.
5. Llamar a `sc = sc_sum(sc, 0)` e imprimir `sc`.
6. Llamar a `sc = sc_sum(sc, 8)` e imprimir `sc`.

Archivos a entregar

En resumen, se deben entregar los siguientes archivos:

- `sc.h`, el archivo de cabeceras.
- `sc.c`, con la implementación de las funciones.
- `main.c`, con la función main pedida.

Recordar

- Se debe resolver el ejercicio en cuatro horas (o menos).
- Se debe compilar pasando todos los flags usados en los proyectos.
- Comentar e indentar el código apropiadamente, siguiendo el estilo de código ya indicado por la cátedra (indentar con 4 espacios, no pasarse de las 80 columnas, inicializar todas las variables, etc).
- Todo el código tiene que usar la librería estándar de C, y no se puede usar extensiones GNU de la misma.
- El programa resultante **no** debe tener *memory leaks* **ni** accesos (read o write) inválidos a la memoria.
- Las funciones deben ser **NO** recursivas.

Apéndice: Definición formal del TAD

TAD calc_pila
constructores
vacía : calc_pila
apilar : entero $\times$ calc_pila $\to$ calc_pila
operaciones
es_vacía : calc_pila $\to$ booleano
primero : calc_pila $\to$ entero {sólo se aplica a pilas no vacías}
desapilar : calc_pila $\to$ calc_pila {sólo se aplica a pilas no vacías}
menos : calc_pila $\to$ calc_pila {sólo se aplica a pilas con al menos dos elementos}
tamaño : calc_pila $\to$ nat
suma : nat $\times$ calc_pila $\to$ calc_pila {sólo se aplica a pares (n, p) con n menor al tamaño de p}
ecuaciones
es_vacía(vacía) = verdadero
es_vacía(apilar(i,p)) = falso

primero(apilar(i,p)) = i
desapilar(apilar(i,p)) = p

menos(apilar(i,apilar(j,p))) = apilar(j-i,p)

tamaño(vacía) = 0
tamaño(apilar(i,p)) = 1 + tamaño(p)

suma(0,p) = apilar(0,p)
suma(1,p) = p
n $\geq$ 2 $\implies$ suma(n,apilar(i,apilar(j,p))) = suma(n-1,apilar(i+j,p))