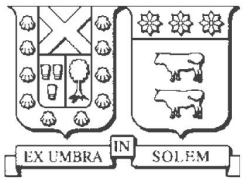
Departamento de Informática
Universidad Técnica Federico Santa María



INF134 Estructura de Datos
Tarea 3

John Rodríguez
jirodrig@alumnos.inf.utfsm.cl

Sebastián Bórquez
sborquez@alumnos.inf.utfsm.cl

Alvaro Luzzi
alvaro.luzzi@usm.cl

1 de Mayo de 2017

1. Instrucciones

En la siguiente tarea crearemos nuestra propia versión de un curioso lenguaje basado en punteros. Para esto utilizarán lo que han aprendido sobre **TDA, listas y punteros**.

1.1. El lenguaje

Consiste de un puntero que apunta a un arreglo "infinito" de números enteros. Este lenguaje solo posee 9 operadores:

Operador	Descripción	Equivalencia C++
>	Incrementa el puntero.	++ptr
<	Decrementa el puntero.	−−ptr
+	Incrementa en uno lo apuntado.	++(*ptr)
-	Decrementa en uno lo apuntado.	−−(*ptr)
.	Muestra en pantalla el valor numérico de lo apuntado.	*ptr
:	Muestra en pantalla el carácter de lo apuntado.	(char)*ptr
[	Inicio de bucle, entra si lo apuntado no es 0.	while(*ptr){
]	Fin de bucle, vuele a [si lo apuntado no es 0.	}
!	Fin de la ejecución.	return 0

Para simplificar el problema **NO** se anidaran bucles y se usará la siguiente tabla de valores para pasar de entero a carácter.

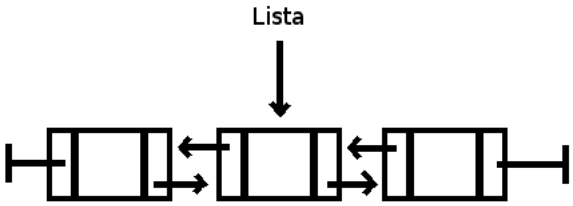
Valores	Caracteres
0	espacio
$[1-26]$	$[a-z]$
$[27-52]$	$[A-Z]$
$[53-62]$	$[0-9]$
$[63-71]$	{. : +− <> []!}

Hints: Use un arreglo de char; Para convertir un número que no está en la tabla debe utilizar el módulo (positivo) de 72 .

1.2. Implementación

Se usarán dos TDA's para esta tarea.

- **Lista doblemente enlazada**



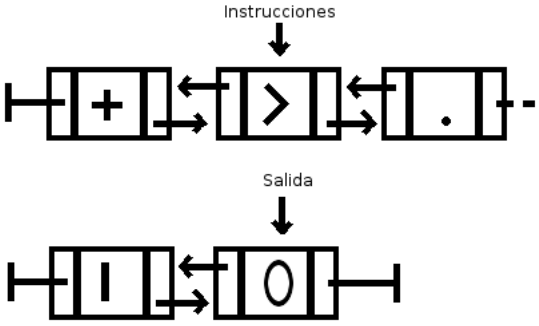
Se usará una versión simplificada, esta consiste de un nodo la cual posee 3 campos: un valor entero, un puntero a un nodo a su izquierda y un puntero a un nodo a su derecha. Solo podrá acceder al nodo apuntado, pero puede desplazarse por los nodos.

Definimos las siguientes funciones:

- Crear: Crea una lista con un nodo.
- Agregar: Agrega un nodo nuevo a la derecha o a la izquierda del nodo al que apunta la lista.
- Mover: Mover el puntero de la lista a nodo de la izquierda o derecha.
- Asignar: Asigna un número al valor del nodo.
- Obtener: Retorna el valor del nodo.
- Eliminar: Elimina el nodo al que apunta la lista.
- Vacia: Determina si la lista está vacía.

Puede crear más funciones auxiliares pero estas solo podrán ser utilizadas en las definiciones de las funciones anteriores.

- **Intérprete**



El intérprete posee listas doblemente enlazadas como campos: **instrucciones** la cual almacena el programa que deseamos ejecutar, **salida** representa nuestro arreglo "infinito" donde se ejecutan las instrucciones. Definimos las siguientes funciones para el intérprete:

- Iniciar: Crea un intérprete.
- Cargar: Carga un programa (un arreglo de char) a la lista instrucciones.
- Mostrar: Muestra el programa contenido en instrucciones, si está vacío debe mostrar "sin instrucciones".
- Ejecutar: Ejecuta las instrucciones.
- Finalizar: Destruye al intérprete.

Puede crear más funciones auxiliares pero estas solo podrán ser utilizadas en las definiciones de las funciones anteriores.

Se debe tomar en cuenta que ambas listas solo almacenan enteros, *instrucciones* solo puede tener un programa cargado, *salida* solo posee datos mientras se ejecuta un programa y cada nodo nuevo de *salida* comienza con valor 0.

1.3. La tarea

Su tarea debe ser capaz de utilizar un intérprete para cargar y ejecutar distintos programas, estos programas se ingresarán y almacenarán (en main) al principio de la ejecución, luego podrá cargar uno de estos al intérprete y para ser ejecutado. Esto último puede repetirse hasta que elijamos terminar el programa. La secuencia del programa es la siguiente:

- Pedir cantidad de programas que desea guardar.
- Ingresar cada uno de los programas.
- Se espera a que se ingrese cualquiera de las siguientes instrucciones:
  - **c n**: Cargar programa, donde n es el índice de los programas ingresados.
  - **e**: Ejecutar programa cargado.
  - **m**: Mostrar el programa cargado.
  - **s**: Terminar la ejecución de tarea3.

2. Ejemplos de entrada y salida

A continuación se presenta un ejemplo de entrada y salida de los datos. En rojo están las entradas y en azul las salidas del programa. Su tarea no debe tener estos colores, pues son solamente referenciales para el uso en el PDF de esta tarea.

Ejemplo 1

```
$ ./tarea3
2
+ > . − −. + + : − − − :!
. > + + + + + + + + .[< + > −] < . > .!
c 0
e
0-2 c
c 1
e
0880
e
0880
s
$
```

Ejemplo 2

```
$ ./tarea3
3
+ + + + +[> + + ++ < −] > − :< + + + + + :< ++ : − :>>: + :< + + ++ :<:>> − − − − −− :!
. > + + + + + + + + .[< + > −] < . > .!
+ + +[> + + + < −] > + : + + + + + :> + + + + + + ++ :< − :!
m
sin instrucciones
e
```

```
c 1
m
. > + + + + + + + + .[< + > −] < . > .!
c 0
e
sebastian
c 2
e
john
s
$
```

Cada programa es ingresado en una línea, sin espacios entre los caracteres, y debe terminar con "!".

3. Consideraciones adicionales

- Los datos de entrada serán correctos, no debe realizar verificación.
- Los programas del lenguaje serán ingresados correctamente.
- La cantidad de programas no está definido.
- El largo de un programa nunca será mayor a 500 caracteres.
- Al estar utilizando TDA solo puede acceder a la estructura a través de sus funciones.

4. Bonificaciones adicionales

- Debe implementarse cada TDA por separado. (10 ptos)
- Habrá bonificación a los 3 primeros grupos en entregar. (15,10,5 ptos)

5. Consideraciones de código

- Sólo se puede usar arreglos básicos de C++.
- Debe seguir el formato de inputs y outputs pedido.
- Debe implementar los TDA utilizando punteros.
- Todo lo estipulado en el reglamento.

6. Sobre entrega

- La fecha límite de entrega de la tarea es el día sábado 13 de mayo antes de las 23:55 hrs.
- Para despejar dudas sobre la tarea o el reglamento de tareas puede consultar en la plataforma Moodle en la sección correspondiente o grupo de facebook.