Tarea 2
Manejo dinámico de estructuras lineales
Curso 2021

Índice

1. Introducción y objetivos

Esta tarea tiene como principal objetivo trabajar sobre el manejo dinámico de memoria con estructuras lineales, en particular con nodos doblemente enlazados.

Para ello, a partir de esta tarea **se utilizará el analizador de uso de memoria** `valgrind`.

Esta herramienta debe estar instalada en las máquinas en las que se prueban los programas y está disponible en las máqinas `Linux` de la facultad.

Por más información sobre la herramienta recurrir al material sobre *valgrind*: Detección de errores en uso de memoria, que se encuentra disponbile en el sitio EVA del curso.

Recordar que:

- **El correcto uso de la memoria será parte de la evaluación**.
- **En esta tarea se puede trabajar únicamente en forma individual**.

El resto del presente documento se organiza de la siguiente forma. En la Sección 2 se presenta una descripción de los materiales disponibles para realizar la presente tarea, y en la Sección 3 se detalla el trabajo a realizar. Luego, en la Sección 4 se explica mediante ejemplos el comportamiento esperado de algunas de las funciones a implementar. La Sección 5 describe el formato y mecanismo de entrega, así como los plazos para realizar la misma. En la sección 6 se muestra un ejemplo gráfico de una cadena.

2. Materiales

Los materiales para realizar esta tarea se extraen de *MaterialesTarea2.tar.gz*. Para conocer la estructura de los directorios ver la Sección **Materiales** de Estructura y funcionamiento del Laboratorio.

En esta tarea los archivos en el directorio `include` son **utils.h**, **info.h**, **cadena.h** y **usoTads.h**.

En el directorio `src` se incluyen ya implementados **utils.cpp** y **info.cpp**. Además, se incluye la implementación de las estructuras y de algunas funciones de **cadena.cpp**.

3. ¿Qué se pide?

Ver la Sección **Desarrollo** de Estructura y funcionamiento del Laboratorio.

En esta tarea solo se deben implementar los archivos **cadena.cpp** y **usoTads.cpp**.

Para implementar `cadena.cpp` se puede usar como base el código entregado.

Se sugiere implementar y probar los tipos y las funciones siguiendo el orden de los ejemplos que se encuentran en el directorio **test**.

En la primera línea de cada caso de prueba se indica que entidades se deben haber implementado. Después de implementar se compila y se prueba el caso.

Los siguientes comandos compilan, ejecutan el test (sin valgrind) y verifican que la salida generada por su programa es igual a la salida esperada:

```
$ make
$ ./principal < test/N.in > test/N.sal
$ diff test/N.out test/N.sal
```

Si la salida generada por su programa es igual a la salida esperada no se imprime nada.

Pero, para tener en cuenta el correcto uso de la memoria y ser notificado de posibles errores de memoria, se debe ejecutar además el siguiente comando:

```
$ valgrind --leak-check=full ./principal <test/NN.in> test/NN.sal
```

Al terminar, para confirmar, se compila y prueban todos los casos mediante la regla `testing` de `Makefile`:

```
$ make testing
```

Al ejecutar el comando `make testing` los casos se prueban usando `valgrind` y la utilidad `timeout`:

```
$ timeout 4 valgrind -q --leak-check=full ./principal < test/01.in > test/01.sal
```

`timeout` es una utilidad que establece un plazo para la ejecución de un proceso (4 en este ejemplo). Si la ejecución demorara más de ese plazo `timeout` terminaría ese proceso.

Recordar que la regla `testing` se dejará disponible sobre la fecha de entrega de la tarea.

Ver el material disponible sobre *Makefile*: Automatización de procedimientos.

Ver también el Método de trabajo sugerido.

4. Descripción de los módulos y algunas funciones

En esta sección se describen los módulos que componen la tarea. Además, de los módulos *cadena* y *uso-Tads* se describen algunas de las funciones que se deben implementar. Tanto de estas funciones como de las otras que también hay que implementar se debe estudiar la especificación en los archivos `.h` correspondientes.
Se recuerda que además del módulo *info* se entrega una implementación completa. Y del módulo *cadena* ejemplos de alguna de las funciones y de la representación de los tipos.

4.1. Módulo *utils*

En este módulo se declara el tipo de los enteros no negativos, arreglos de tipos simples, y operaciones de lectura desde la entrada estándar.

4.2. Módulo *info*

En este módulo se declara el tipo `TInfo`. Este tipo representa información compuesta por un natural (tipo `nat`) y un punto flotante (tipo `double`).
Se debe notar que en esta tarea para acceder a los componentes de un elemento de tipo `TInfo` se deben usar las operaciones declaradas en `info.h`.

4.3. Módulo *cadena*

El módulo declara dos conceptos, denominados `TCadena` y `TLocalizador`. Con `TCadena` se implementan cadenas doblemente enlazadas de elementos de `TInfo` con una cabecera con punteros al inicio y al final. El tipo `TLocalizador` es un puntero a un nodo de una cadena, o no es válido, en cuyo caso su valor es `NULL`. Con el uso de los localizadores se pueden realizar algunas operaciones de manera eficiente (sin tener que hacer un recorrido de la cadena).
A continuación se muestran gráficamente ejemplos de ejecución de algunas de las operaciones que se solicitan implementar. Cuando la ubicación de los localizadores es relevante se indica mediante colores.

4.3.1. *insertarAntes(i, loc, cad)*

i	cad (loc)	cad luego de la ejecución
(3,2.5)	[(1,1.5),(5,3.2),(2,56.1),(8,13.9)]	[(3,2.5),(1,1.5),(5,3.2),(2,56.1),(8,13.9)]
(3,2.5)	[(1,1.5),(5,3.2),(2,56.1),(8,13.9)]	[(1,1.5),(5,3.2),(3,2.5),(2,56.1),(8,13.9)]

4.3.2. *removerDeCadena(loc, cad)*

cad (loc)	cad luego de la ejecución
[(1,1.5),(5,3.2),(2,56.1),(8,13.9)]	[(5,3.2),(2,56.1),(8,13.9)]
[(1,1.5),(5,3.2),(2,56.1),(8,13.9)]	[(1,1.5),(5,3.2),(2,56.1)]
[(1,1.5),(5,3.2),(2,56.1),(8,13.9)]	[(1,1.5),(5,3.2),(8,13.9)]

4.3.3. *intercambiar(loc1, loc2, cad)*

cad (loc1 y loc2)	cad luego de la ejecución
[(1,1.5),(3,8.1),(5,3.2),(9,7.4),(6,-4.7)]	[(1,1.5),(9,7.4),(5,3.2),(3,8.1),(6,-4.7)]

4.3.4. *siguienteClave(clave, loc, cad)*

clave	cad (loc)	resultado (representado en cad)
8	[] (loc es ignorado)	localizador no válido
3	[(1,1.5),(4,0.9),(2,56.1)]	localizador no válido
9	[(1,1.5),(4,0.9),(2,56.1),(9,7.4)]	[(1,1.5),(4,0.9),(2,56.1),(9,7.4)]
4	[(1,1.5),(4,0.9),(2,56.1),(9,7.4)]	[(1,1.5),(4,0.9),(2,56.1),(9,7.4)]

4.3.5. *borrarSegmento(desde, hasta, cad)*

cad (desde y hasta)	cad luego de la ejecución
[(1,1.5),(3,8.1),(5,3.2),(9,7.4),(6,-4.7)]	[(6,-4.7)]
[(1,1.5),(3,8.1),(5,3.2),(9,7.4),(6-4.7)]	[(1,1.5),(3,8.1)]
[(1,1.5),(3,8.1),(5,3.2),(9,7.4),(6,-4.7)]	[(1,1.5),(6,-4.7)]

4.4. Módulo *usoTads*

Este módulo corresponde a funciones que usan las operaciones declaradas en los módulos *info* y *cadena*.

5. Entrega

Se mantienen las consideraciones reglamentarias y de procedimiento de la tarea anterior.
Se debe entregar el siguiente archivo, que contiene los módulos implementados `cadena.cpp` y `usoTads.cpp`:

- **Entrega2.tar.gz**

Este archivo se obtiene al ejecutar la regla `entrega` del archivo *Makefile*:

```
$ make entrega
tar zcvf Entrega2.tar.gz -C src cadena.cpp usoTads.cpp
cadena.cpp
usoCadena.cpp
```

Con esto se empaquetan los módulos implementados y se los comprime.

5.1. Plazos de entrega

El plazo para la entrega es el **martes 13 de abril a las 18 horas**.

5.2. Identificación de los archivos de las entregas

Cada uno de los archivos a entregar debe contener, en la primera línea del archivo, un comentario con el número de cédula del estudiante, **sin el guión y sin dígito de verificación**. Ejemplo:

```
/* 1234567  */
```

5.3. Individualidad

Ver la Sección **Individualidad** de Reglamento del Laboratorio.

6. Ejemplo de cadena

En la Figura 1 se muesta un ejemplo de `TCadena`.

La variable *cad* de tipo `TCadena` es un puntero a una estructura de dos miembros, *inicio* y *final* que son punteros a `nodo`. Un elemento de tipo `nodo` es una estructura con tres miembros: *anterior* y *siguiente* de tipo puntero a `nodo`, y *dato* de tipo `TInfo`. El tipo `TInfo` es un puntero a una estructura de dos miembros: *n*, de tipo `nat` (con valor 13 en el ejemplo) y *r*, de tipo `double` (con valor 6.5 en el ejemplo).

Las variables *loc1*, *loc2* y *loc3* son de tipo `TLocalizador`, que es un puntero a `nodo`.

En este ejemplo, accediendo a la representación de `cadena`, se puede afirmar que se cumplen los predicados

- `loc1 == cad->inicio->siguiente`,
- `loc2 == cad->final->siguiente`,
- `loc2 == NULL`.

Usando las operaciones de `cadena` se cumplen, entre otros:

- `! esVaciaCadena(cad)`,
- `loc1 == siguiente(inicioCadena(cad), cad)`,
- `loc2 == siguiente(finalCadena(cad), cad)`,
- `! esLocalizador(loc2)`,
- `localizadorEnCadena(loc1, cad)`,
- `! localizadorEnCadena(loc2, cad)`,
- `precedeEnCadena(loc1, loc3, cad)`.

Nótese que no se puede expresar `loc3->dato->n == 13` porque desde `cadena` no se puede acceder a la representación de `TIinfo`. En cambio es válido `natInfo(loc3->dato) == 13` o, usando las operaciones de `cadena`,
`natInfo(infoCadena(loc3, cad)) == 13`.

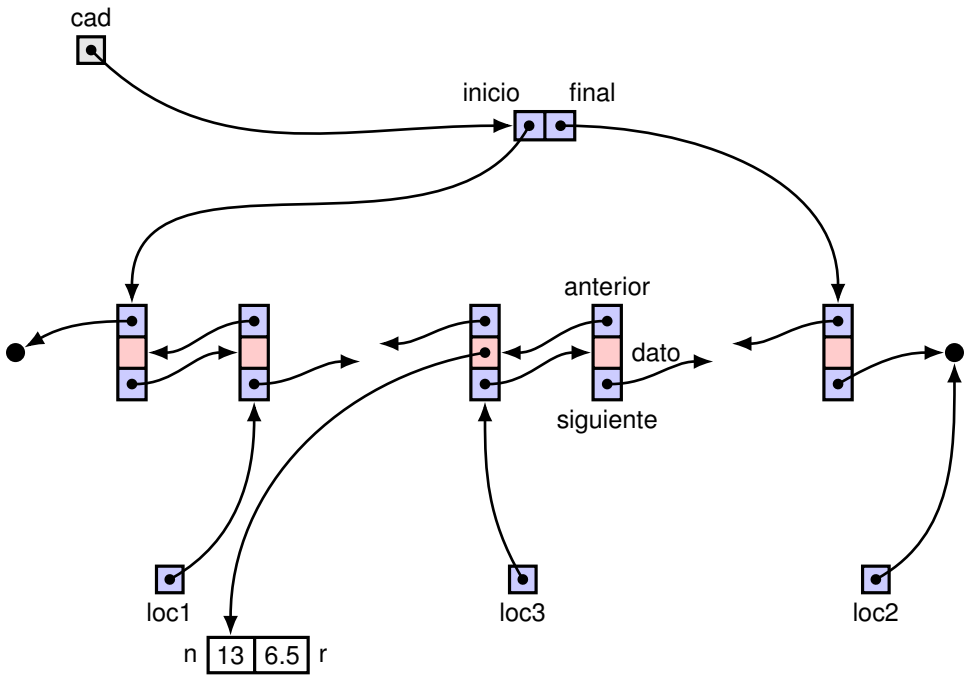

Figura 1: Ejemplo de cadena