

Tarea 4

Implementación de Tipos Abstractos de Datos

Curso 2019

Índice

1. Introducción	2
2. Materiales	2
3. ¿Qué se pide?	3
3.1. Verificación de la implementación	3
4. Descripción de los módulos y algunas funciones	3
4.1. Módulos de tareas anteriores	3
4.2. Módulos nuevos	3
4.2.1. Módulo <i>pila</i>	3
4.2.2. Módulo <i>avl</i>	3
4.2.3. Módulo <i>cola_binarios</i>	3
4.2.4. Módulo <i>iterador</i>	3
4.2.5. Módulo <i>conjunto</i>	3
4.2.6. Ejemplos	3
5. Entrega	4
5.1. Plazos de entrega	5
5.2. Identificación de los archivos de las entregas	5
6. Funciones auxiliares	6
7. Material Complementario	6

1. Introducción

El objetivo de esta tarea es la implementación de **Tipos Abstractos de Datos (TADs)** y de nuevas estructuras cumpliendo requerimientos de tiempo de ejecución. Mantenemos los módulos de las tareas anteriores. Los detalles sobre los módulos de tareas anteriores se presentan en la Sección 4.1.

También trabajaremos sobre módulos nuevos, que implementan pilas, árboles AVL, colas de árboles AVL, iteradores y conjuntos. Los detalles de los módulos nuevos se presentan en la Sección 4.2.

Como corresponde al concepto de TAD no se puede usar la representación de un tipo fuera de su propio módulo. Por ejemplo, en `uso_tads.cpp` no se puede usar `rep_cadena` sino las operaciones declaradas en `cadena.h`.

El correcto uso de la memoria y el cumplimiento de las restricciones de tiempo será parte de la evaluación.

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. Por último, la Sección 5 describe el formato y mecanismo de entrega, así como los plazos para realizar la misma.

2. Materiales

Los materiales para realizar esta tarea se encuentran en el archivo *MaterialesTarea4.tar.gz* que se obtiene en la carpeta *Materiales* de la sección *Laboratorio* del sitio EVA del curso ¹.

A continuación se detallan los archivos que se entregan. La estructura de directorios es igual a la de la tarea anterior.

- **info.h**: módulo de definición del tipo `info_t`
- **cadena.h**: módulo de definición de `cadena`.
- **binario.h**: módulo de definición de árbol binario de búsqueda
- **uso_cadena.h**: módulo de definición de funciones sobre cadenas
- **pila.h**: módulo de definición de pilas de elementos de tipo `int`
- **avl.h**: módulo de definición de árboles AVL de elementos de tipo `info_t`
- **cola_avls.h**: módulo de definición de colas de árboles AVL
- **iterador.h**: módulo de definición de iteradores sobre elementos de tipo `info_t`
- **conjunto.h**: módulo de definición de conjuntos de elementos de tipo `info_t`
- Casos de prueba en el directorio **test**
- **Makefile**: archivo para usar con el comando `make`, que provee las reglas `principal`, `testing`, `clean` y entrega como en las tareas anteriores.
- **principal.cpp**: módulo principal.
- En el directorio `src` se entregan además:
 - el archivo *ejemplos_cpp.png* que contiene la imagen de la implementación de algunos tipos y funciones que se piden.
 - el archivo *plantilla_iterador.cpp* con una implementación de *iterador.cpp*.

Los contenidos de estos archivos **pueden** copiarse en los archivos a implementar.

Ninguno de estos archivos deben ser modificados.

¹<https://eva.fing.edu.uy/course/view.php?id=132§ion=6>

3. ¿Qué se pide?

Para cada archivo de encabezamiento **.h**, descrito en la Sección 2, se debe implementar un archivo **.cpp** que implemente **todas** las definiciones de tipo y funciones declaradas en el archivo de encabezamiento correspondiente. Los archivos **cpp** serán los entregables y deben quedar en el directorio **cpp**.

En algunas operaciones se piden requerimientos de tiempo de ejecución. Ese tiempo es el del peor caso, a menos que se especifique otra cosa de manera explícita.

3.1. Verificación de la implementación

Para testear la implementación debe generar el ejecutable y cumplir con todas las pruebas utilizando los archivos **.in** y **.out**. Tenga en cuenta que el archivo **Makefile** proveerá las reglas **principal**, **testing** y **clean** como en las tareas anteriores.

4. Descripción de los módulos y algunas funciones

4.1. Módulos de tareas anteriores

Se mantienen los mismos módulos de la Tarea 3: *info*, *cadena*, *uso_cadena* y *binario*.

La definición del tipo *nat* se movió del módulo *cadena* al módulo *info*.

4.2. Módulos nuevos

4.2.1. Módulo *pila*

En este módulo se debe implementar una pila (de tipo *pila_t*) de elementos de tipo *int*.

4.2.2. Módulo *avl*

En este módulo se debe implementar un árbol binario de búsqueda (de tipo *avl_t*) de elementos de tipo *info_t* con la propiedad de estructura *avl*. El orden en el árbol está determinado por los datos numéricos de sus elementos.

4.2.3. Módulo *cola_binarios*

En este módulo se debe implementar una cola (de tipo *cola_avls_t*) de elementos de tipo árboles de tipo *avl_t*. Es posible que un árbol sea subárbol de otro, o sea, los árboles pueden compartir memoria.

4.2.4. Módulo *iterador*

En este módulo se debe implementar un iterador (de tipo *iterador_t*) sobre elementos de tipo *info_t*. Un iterador es una colección de elementos que se puede recorrer.

4.2.5. Módulo *conjunto*

En este módulo se debe implementar un conjunto (de tipo *conjunto_t*) de elementos de tipo *info_t*. Una restricción adicional es que no puede haber dos elementos con el mismo dato numérico.



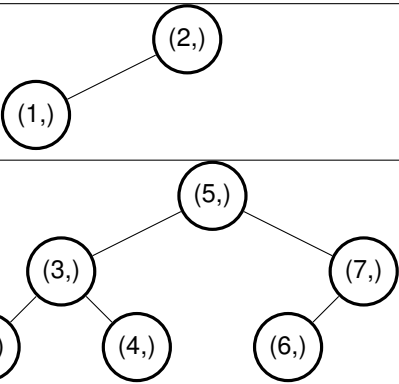
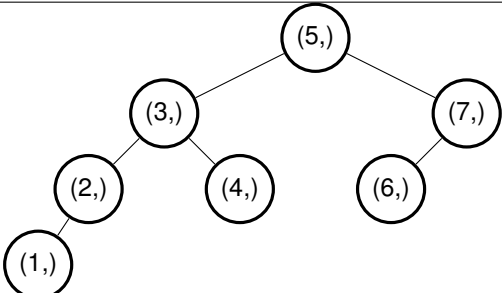
4.2.6. Ejemplos

En esta sección se muestran ejemplos de lo que se espera de la ejecución de algunas funciones.

imprimir_avl Imprime los datos numéricos de los elementos del árbol por niveles, un nivel por línea desde el más profundo hasta la raíz.

avl	Salida esperada
	2
	b d
	4 7 1 5 2

min_avl Devuelve el avl que tiene la mínima cantidad de nodos para la altura pasada por parámetro. Los datos numéricos deben ser consecutivos y empezar en 1. El dato de texto de cada elemento es la cadena vacía. En ningún nodo puede ocurrir que el subárbol derecho tenga más nodos que el subárbol izquierdo.

Altura	Árbol resultado
0	
1	
2	
4	

5. Entrega

Se mantienen las consideraciones reglamentarias y de procedimiento de las tareas anteriores.

En particular, la tarea otorga dos punto a los grupos que la aprueben en la primera instancia de evaluación y uno o cero a los que apruebn en la segunda instancia. La adjudicación de los puntos de las tareas de laboratorio queda condicionada a la respuesta correcta de una pregunta sobre el laboratorio en el segundo parcial. La adjudicacion se aplica de manera individual a cada estudiante del grupo.

La tarees es **eliminatória**.

Se debe entregar el siguiente archivo, que contiene los módulos implementados *avl.cpp*, *binario.cpp*, *cadena.cpp*, *cola_avls.cpp*, *conjunto.cpp*, *info.cpp*, *iterador.cpp*, *pila.cpp* *uso_cadena.cpp*:

■ Entrega4.tar.gz

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

```
$ make entrega
tar zcvf Entrega4.tar.gz -C src info.cpp cadena.cpp uso_cadena.cpp binario.cpp pila.cpp avl.cpp cola.cpp
info.cpp
cadena.cpp
uso_cadena.cpp
binario.cpp
pila.cpp
avl.cpp
cola_avls.cpp
iterador.cpp
conjunto.cpp
```

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

Nota: En la estructura del archivo de entrega los módulos implementados deben quedar en la raíz, NO en el directorio *src* (y por ese motivo se usa la opción *-C src* en el comando *tar*).

NO SE PUEDEN ENTREGAR MÓDULOS ADICIONALES A LOS SOLICITADOS

5.1. Plazos de entrega

El plazo para la entrega es el **jueves 6 de junio a las 21 horas**.

NO SE ACEPTARÁN ENTREGAS DE TRABAJOS FUERA DE FECHA Y HORA. LA NO ENTREGA O LA ENTREGA FUERA DE LOS PLAZOS INDICADOS IMPLICA LA PÉRDIDA DEL CURSO.

5.2. Identificación de los archivos de las entregas

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

```
/* 1234567 - 2345678 */
```

6. Funciones auxiliares

Para la implementación de algunas de las funciones de esta tarea podrían necesitarse funciones auxiliares. En C no se puede definir una función anidada dentro de otra como se puede hacer en Pascal. Por lo tanto la función auxiliar debe definirse fuera de las funciones en las que se va a usar. Además, la declaración debe estar antes de que la función sea usada.

Esto puede llegar a generar un conflicto en el momento del enlazado si en otros archivos también hubiera funciones auxiliares con la misma firma ya que en C las funciones de manera predeterminada tienen ámbito global. Para evitar este problema se debe anteponer la palabra reservada `static` a la declaración de la función. Con esto queda como local al archivo. Su ámbito va desde la declaración hasta el fin del archivo.

```
// Declaración de 'auxiliar' usada en 'f' y 'g'.
// Debe preceder a la implementación de las funciones que la usan.
static int auxiliar(int p);
// La implementación se puede hacer junto con la declaración
// o después.

// f y g usan 'auxiliar'
void f() {
    ...
    int a = auxiliar(5);
    ...
}

void g() {
    ...
    int a = auxiliar(8);
    ...
}

// implementación de auxiliar
static int auxiliar(int p) {
    ...

    ...
    return p + ...;
}
```

7. Material Complementario

En la sección de laboratorio se encuentra el documento Material Complementario, el cual contiene la información referente a Makefile, Control de manejo de memoria, Assert y Método sugerido.