Tercer Parcial

22 de mayo de 2021

Indicaciones generales

1. Este es un examen **individual** con una duración de **100 minutos: de 10:10am a 11:50am**.
2. En `e-aulas` puede acceder a las diapositivas y a la sección correspondiente a este parcial.
3. Solamente será posible tener acceso a `e-aulas.urosario.edu.co` y a los sitios web correspondientes a la documentación de C++ dispuestos por el profesor.
4. Maletas, morrales, bolsos, etc. deben estar ubicados al frente del salón.
5. Celulares y otros dispositivos electrónicos deben estar apagados y ser guardados dentro de las maletas antes de ser ubicadas en su respectiva posición.
6. El estudiante no debe intentar ocultar ningún código que no sea propio en la solución a la actividad.
7. El estudiante solo podrá disponer de hojas en blanco como borrador de apuntes (opcional).
8. El estudiante puede tener una hoja manuscrita de resumen (opcional). Esta hoja debe estar marcada con nombre completo.
9. `e-aulas` se cerrará a la hora en punto acordada. La solución de la actividad debe ser subida antes de esta hora. El material entregado a través de `e-aulas` será calificado tal como está. Si ningún tipo de material es entregado por este medio, la nota de la evaluación será 0.0. **Se aconseja subir a e-aulas versiones parciales de la solución a la actividad.**
10. Todas las evaluaciones serán realizadas en el sistema operativo GNU/Linux.
11. Todas las entregas están sujetas a herramientas automatizadas de detección de plagio en códigos.
12. **Cualquier incumplimiento de lo anterior conlleva la anulación del examen.**
13. Las respuestas deben estar totalmente justificadas.
14. **Entrega**: archivos con extensión `.cpp` o `.hpp` según el caso, conteniendo el código. Comprima su código y demás archivos en *un único* archivo `parcial.zip` y súbalo a `e-aulas`. **Importante:** no use acentos ni deje espacios en los nombres de los archivos que cree.

En el desarrollo del examen, no olvide usar la plantilla para cada ejercicio.
Las implementaciones deben ejecutarse sin errores usando las funciones `main()` de cada plantilla.

1. [50 ptos.] Una lista simplemente enlazada consiste en nodos que contienen un apuntador `next` que apunta al siguiente nodo en la lista o a `nullptr` si se trata del primer nodo añadido a la lista (front).

   Una lista doblemente enlazada es una estructura lineal en la que cada nodo tiene dos apuntadores: `next` y `prev`, que apuntan respectivamente al nodo siguiente o al nodo que lo precede. En caso del primer nodo (back), `next` apunta a `nullptr` y en caso de ser el último nodo (front), es `prev` el que apunta a `nullptr`.

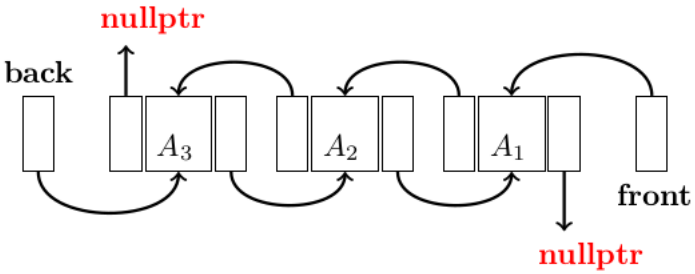

Figura 1: Estructura de una lista doblemente enlazada.

A continuación se presenta la interfase de una lista doblemente enlazada:

```
template <typename T>
struct Cell {
  T val;
  Cell *next;
  Cell *prev;
};

template <typename T>
class List {
private:
  Cell<T> *back;
  Cell<T> *front;
  int size;

public:
  List();
  ~List();

  bool empty();
  /**implement***/
  void push_back(T c);
  void pop_back();
  /*************/
  T get_back();
  /**implement***/
  void push_front(T c);
```

```
27    void pop_front();
28    /**************/
29    T get_front();
30
31    void display();
32  };
```

Usando la plantilla proporcionada implemente:

a) Los métodos `void push_back(T c)` y `void push_front(T c)` que añaden un elemento nuevo al final/inicio de la lista, respectivamente.

b) Los métodos `void pop_back()` y `void pop_front()` que remueven el elemento del final/inicio de la lista, respectivamente.

2. [50 ptos.] Agregue a la implementación e interfaz de un árbol binario de búsqueda las siguientes rutinas que no modifican el estado del árbol. Considere la estructura de datos árbol binario de búsqueda (*binary search tree*). Implemente como métodos privado y público, respectivamente, los siguientes predicados:

```
1  bool check_if_bst(bstNode *root) const; // private
2  bool check_if_bst() const;              // public
```

que verifican si el árbol binario que tiene como raíz a `root` satisface la propiedad de árbol binario de búsqueda. Su implementación debe tomar tiempo lineal, en el número de nodos del árbol binario. El método público solamente debe invocar al método privado con los parámetros apropiados.