Segundo Parcial

12 de octubre de 2019

Indicaciones generales

1. Este es un examen **individual** con una duración de **120 minutos: de 11:00 a 13:00**.
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. La evaluación debe presentarse exclusivamente en uno de los computadores ubicados en el salón de clase y a la hora acordada. Presentar la evaluación desde otro dispositivo o en otro horario diferente al estipulado es causa de anulación.
13. **Cualquier incumplimiento de lo anterior conlleva la anulación del examen.**
14. Las respuestas deben estar totalmente justificadas.
15. **Entrega**: archivos con extensión `.txt`, `.cpp` o `.hpp` según el caso, conteniendo la demostración o el código. Nombre los archivos como `pY.Z`, con `Y = 1,2,3` y `Z = txt, cpp, hpp`. 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. [40 ptos.] La idea de este ejercicio es ordenar instancias de la clase `Racional`. Esta tarea debe realizarse en dos pasos.

   a) [20 ptos.] Implemente el `operator<` que compara dos objetos de la clase `Racional`. El prototipo de este método debe ser

```
1 bool operator<(const Racional &p, const Racional &q);
```

Antes de continuar, verifique el correcto funcionamiento de su implementación. Por ejemplo, las expresiones booleanas $1/4 < 1/2$, $2 < 4$ y $2/7 < 3/5$ son verdaderas.

b) [20 ptos.] Implemente una función que ordena un vector de racionales. Esta función debe ser de la forma

```
1 void rational_sort(vector<Racional> & vr);
```

Note que la modificación del vector debe hacerse por referencia; es decir, la función no debe retornar ningún tipo de dato. Implemente una variación, para ordenar racionales, de cualquiera de los algoritmos de ordenamiento básicos: `bubble sort`, `insertion sort` o `selection sort`.

No olvide verificar el correcto funcionamiento del código implementado. Pruebe con dos tipos de vectores: ordenados y desordenados.

NOTA: Asegúrese de implementar el literal *b*) incluso si no ha implementado el literal *a*).

2. [20 ptos.] Escriba una función recursiva `int digit_sum(int n)` que toma un entero, mayor o igual a cero, y retorna la suma de sus dígitos. Por ejemplo, al invocar a `digit_sum(1729)` debe retornar 19 que corresponde a la suma $1 + 7 + 2 + 9$.

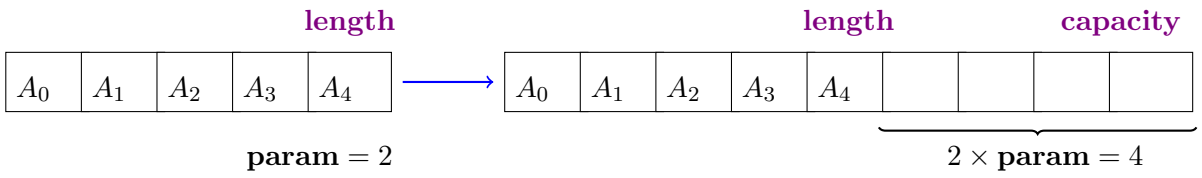
   La implementación recursiva de `digit_sum` depende del hecho de que es muy fácil dividir un entero en dos componentes utilizando la división entera entre 10. Por lo tanto, cada uno de los enteros resultantes es estrictamente más chico que el original y, por lo tanto, representa una reducción del tamaño del problema original.

3. [40 ptos.] Extienda la clase `SimpleVec` de forma que incluya dos métodos nuevos, descritos a continuación.

   a) [20 ptos.] Implemente un método privado que expande la capacidad de una instancia de `SimpleVec`

   ```
   1 void expandCapacityWith(int param);
   ```

   donde la expansión se realiza bajo el siguiente criterio: Si la capacidad del vector debe ser aumentada, entonces, su nueva capacidad debe ser igual a la capacidad anterior más dos veces el valor de `param`. El siguiente diagrama ejemplifica el funcionamiento del método:

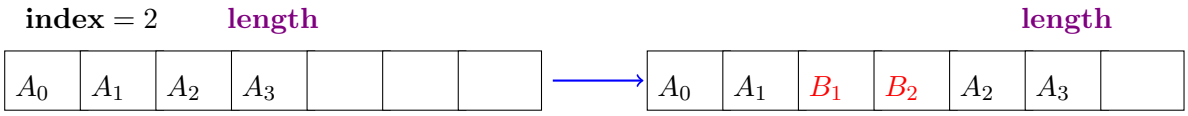


   b) [20 ptos.] Implemente el siguiente método público de inserción de elementos, usando un arreglo de `double`s,

   ```
   1 void insert_chunk(int index, double *vals, int vals_size);
   ```

   que inserta el arreglo `vals`, de tamaño `vals_size`, en un objeto de la clase `SimpleVec` a partir del índice `index`. Invoque `expandCapacityWith(param)` desde `insert_chunk`, donde `param = vals_size`. Es decir, en el método `insert_chunk` se debe verificar si al insertar los nuevos valores se sobrepasa la capacidad actual del arreglo. En tal caso, esta se

incrementa en una cantidad igual a `2*vals_size`. La expansión de la capacidad debe ocurrir siempre que `capacity` sea menor que `length + vals_size`. El siguiente diagrama ejemplifica el funcionamiento típico del método:



Verifique el correcto funcionamiento de los métodos implementados.

NOTA: Asegúrese de implementar el literal b) incluso si no ha implementado el literal a).