Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática
Semestre I - 2021
Curso CI-0113 - Programación II
Profesor: Edgar Casasola Murillo
Fecha: 9 de junio 2021

EXAMEN I

Para la resolución de este examen usted puede utilizar cualquiér código construido en clase disponibles en git.ucr.ac.cr. Para su solución usted NO DEBE utilizar funciones de la biblioteca estándar de plantillas STL (ni la clase string, ni métodos como strcmp o strcpy de C).

NO SE TOMARÁ EN CUENTA SU TRABAJO SI "COPIA" CÓDIGO FUENTE DE TERCEROS, EL CÓDIGO DEBERÁ SER ESCRITO POR USTEDES CON LO APRENDIDO EN EL CURSO.

Parte A

La empresa HOMEALONE S.A. necesita desarrollar un software que procese ***transacciones bancarias*** y genere ***saldos*** a partir de ellas. Una ***transacción bancaria*** es una transferencia de dinero de una persona 1 hacia una persona 2, se representa como un objeto de la clase **Transaccion** con atributos *Persona1 Persona2 y Monto,* donde *Persona 1 y Persona 2* son hileras de caracteres con nombres de personas y *el Monto* es un número real que indica el dinero que la persona *1* le transfirió a *la persona 2.* Un ***saldo*** está representado de la forma ***Persona Monto****,* donde *Persona* es el nombre de una persona y *Monto* es un saldo de dinero que la Persona tiene, *el Saldo* puede ser negativo (debe dinero).

Por ejemplo, para el siguiente conjunto de transacciones:

Esteban Nicole 1500.00
Erik Nicole 3000.00
Nicole Asch 2000.00

Se tienen los siguientes saldos:

Asch	2000.00
Erik	-3000.00
Esteban	-1500.00
Nicole	2500.00

La empresa HOMEALONE S.A. le ha encargado a usted que programe lo siguiente:

Ejercicio #1 (30 %)

Programe las clases ***Transacción y Saldo*** con **constructores por omisión, por copia**, **constructor con parámetros** y **destructor**. Ambas clases deberán tener ***métodos get()*** para obtener ***una referencia a cada uno de sus atributos por separado*** (NO UNA COPIA). Además: **sobrecargue los operadores de lectura desde un flujo de datos** y **salida hacia un flujo** (operator>> y operator <<). Puede crear todos los métodos privados que considere necesarios. **Sobrecargue además** los operadores **<, ==, !=** de ambas clases. Para los operadores **==** y **!=,** dos objetos se consideran iguales si y sólo si los nombres son exactamente iguales a los del segundo objeto (no así el monto). (Nota: Mayúsculas y minúsculas son diferentes). Para el **operador < se debe tomar en cuenta tanto los nombres como el saldo.** Para el **Saldo** el orden lo define primero el nombre alfabéticamente y si son iguales se compara el saldo de menor a mayor para definir el orden. En el caso de la **Transacción para definir el orden <** la prioridad la tiene el primer nombre alfabéticamente, si son iguales, se compara usando los segundos nombres alfabéticamente, y si son iguales se compara usando el saldo.

Ejercicio #2 (40%)

Escriba un programa completo y las clases necesarias para resolver este problema:

Su programa debe leer desde el archivo "*Transacciones.txt"* un conjunto de transacciones, una por línea, debe aplicar un modelo ***map reduce*** para determinar el *saldo* de cada persona. Luego, debe almacenar estas transacciones en una lista de Transacciones con iterador, **ordenados de menor a mayor**, (usando el operador < que se sobrecargó en la clase Saldo).

El modelo ***map reduce*** es muy utilizado en computación, cuando se deben procesar grandes volúmenes de datos (también conocidos como *big data*). En él, se tienen dos funciones: ***map****,* que se encarga de aplicar alguna función a los datos y generar un par ordenado de la forma *(a,b);* y la función ***reduce****,* que se encarga de unir o "reducir" todos los valores de *b* para un mismo *a.* En este problema, la función ***map*** se encarga de generar pares del tipo ***(Persona, Monto)*** *a partir de una* ***Transaccion*** y la función ***reduce*** se encarga de **sumar** todos los ***Montos*** para determinar el balance de cada ***Persona****.*

Note que, para cada transacción, se deben generar **dos** pares ordenados: uno para la persona que transfiere el dinero y otro para la persona que recibe el dinero.

Por ejemplo, para el siguiente conjunto de transacciones:
Transacciones.txt

Esteban Nicole 1500
Erik Nicole 3000

Nicole Asch 2000

La función *map* generaría los siguientes pares ordenados:

(Esteban, -1500)
(Nicole, 1500)
(Erik, -3000)
(Nicole, 3000)
(Nicole, -2000)
(Asch, 2000)

Los cuales, al ser procesados por la función *reduce* y ordenados, se convierten en:

Salida en consola

(Asch, 2000)
(Erik, -3000)
(Esteban, -1500)
(Nicole, 2500)

PUEDE PROGRAMAR TODAS LAS CLASES (O LISTAS EXTRA NECESARIAS PARA SU SOLUCIÓN)

PARTE B

Objetivo de este último ejercicio: Evaluar manejo de punteros, asignación y liberación de memoria.

1) (20%) Programe una clase llamada **Separador** que contenga un método llamado **separar** que reciba un puntero constante hacia una hilera de caracteres y retorna un puntero a un vector de punteros a cada una de las palabras contenidas en la hilera. El puntero de la última celda tendrá el valor 0 como se muestra en el diagrama de ejemplo.
Puede programar los métodos privados que necesite.

El método:

char * * Separador::separar(const char *);

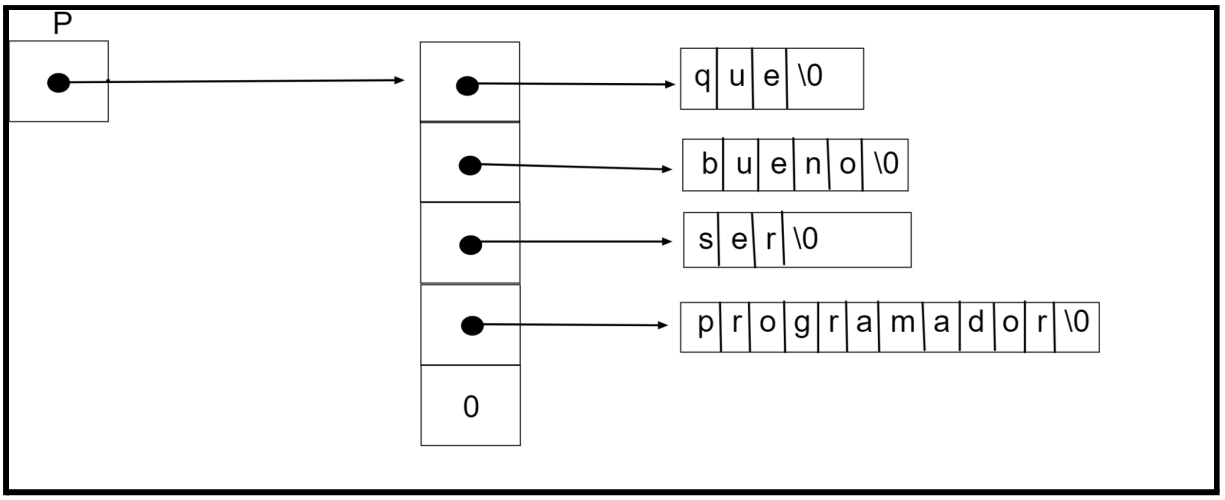
Asuma que la hilera se compone solamente de caracteres alfanuméricos (a-z, A-Z, 0-9), espacios (‘ ‘) y tabuladores (‘\t’).

El método debe separar las palabras delimitadas por espacios o tabuladores y **asignarles solamente la memoria que necesitan** (debe incluir espacio para un carácter adicional de cierre de hilera al final de cada palabra).

Por ejemplo, si se ejecuta:

Char * * p = instancia.separar(“que bueno ser programador”);

El método debe devolver un **vector de punteros**, con la siguiente estructura y contenido:



Programe un main que muestre como se separa la frase anterior y se imprime palabra por palabra.

2) (10%). Escriba un segundo método dentro de la clase **Separador** llamado

void liberar(char * * p);

que solo recibe el puntero p que apunta a la estructura que tiene que liberar. E método **libera toda la memoria solicitada** por el método **separar(...)** t**anto de las palabras como del vector de punteros** con el fin de **no dejar fugas de memoria.**

FORMA DE ENTREGA:

Debe subir su solución al enlace respectivo en mediacionvirtual.ucr.ac.cr antes de las 5:00 p.m. del día Miércoles 9 de Junio del 2021.

Su solución puede consistir de:

1. Si hoy no cuenta con computador adecuado para la solución del examen o si se va la electricidad o internet (Casos especiales conocidos):
   Carpeta con Fotos o Documento en formato .pdf de su solución a puño y letra en caso de que no cuente con computador para trabajar el día de hoy. Explique su solución y escriba todo el código fuente .h y .cpp de cada respuesta.
2. Caso contrario, suba una carpeta con subcarpetas para cada pregunta. Debe incluir todo el código .h y .cpp y main de prueba con el que usted verificó el funcionamiento de las respuestas a cada pregunta.

Puede replicar código en las carpetas con tal de que sea compilable sin problema. Incluya un archivo de texto llamado README.md donde explique como compilar, ejecutar cada solución y posibles problemas encontrados en caso de que no le compile.

PUNTOS EXTRA

C**omo parte de este examen** y en ambos casos se dará un 10% extra por entregar una solución extra debidamente documentada y funcional del examen. Cuya entrega será el miércoles 16 de junio en el enlace indicado en mediación virtual.