PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

PROGRAMACIÓN 2

2da práctica (tipo b)
(Segundo Semestre 2024)

Indicaciones Generales:
Duración: **110 minutos.**

- No se permite el uso de apuntes de clase, fotocopias ni material impreso.
- No se pueden emplear variables globales, ni objetos (con excepción de los elementos de `iostream`, `iomanip` y `fstream`). No se puede utilizar la clase `string`. Tampoco se podrán emplear las funciones de C que gesten memoria como `malloc`, `realloc`, `memset`, `strdup`, `strtok` o similares, igualmente no se puede emplear cualquier función contenida en las bibliotecas `stdio.h`, `cstdio` o similares y que puedan estar también definidas en otras bibliotecas. No podrá emplear plantillas en este laboratorio.
- Deberá modular correctamente el proyecto en archivos independientes. Las soluciones deberán desarrollarse bajo un estricto diseño descendente. Cada función no debe sobrepasar las 20 líneas de código aproximadamente. El archivo `main.cpp` solo podrá contener la función `main` de cada proyecto y el código contenido en él solo podrá estar conformado por tareas implementadas como funciones. En el archivo `main.cpp` deberá incluir un comentario en el que coloque claramente su nombre y código, de no hacerlo se le descontará 0.5 puntos en la nota final.
- El código comentado no se calificará. De igual manera no se calificará el código de una función si esta función no es llamada en ninguna parte del proyecto o su llamado está comentado.
- Los programas que presenten errores de sintaxis o de concepto se calificarán en base al 40 % de puntaje de la pregunta. Los que no muestren resultados o que estos no sean coherentes en base al 60 %.
- Se tomará en cuenta en la calificación el uso de comentarios relevantes.
- Se les recuerda que, de acuerdo al reglamento disciplinario de nuestra institución, constituye una falta grave copiar del trabajo realizado por otra persona o cometer plagio.
- No se harán excepciones ante cualquier trasgresión de las indicaciones dadas en la prueba.

Puntaje total: 20 puntos

- La unidad de trabajo será `t:\` (Si lo coloca en otra unidad, no se calificará su laboratorio y se le asignará como nota cero). En la unidad de trabajo `t:\` colocará el(los) proyecto(s) solicitado(s).
- Cree allí una carpeta con el nombre "Lab02_2024_2_CO_PA_PN" donde: **CO** indica: Código del alumno, **PA** indica: Primer Apellido del alumno y **PN** indica: primer nombre. De no colocar este requerimiento se le descontará 3 puntos de la nota final.

Cuestionario

- La finalidad principal de este laboratorio es la de reforzar los conceptos contenidos en el capítulo 2 del curso: "Arreglos y punteros". En este laboratorio se trabajará con **memoria dinámica** y el método de **asignación exacta de memoria**.
- Al finalizar la práctica, comprima la carpeta dada en las indicaciones iniciales empleando el programa `Zip` que viene por defecto en el Windows, no se aceptarán los trabajos compactados con otros programas como `RAR`, `WinRAR`, `7zip` o similares.

Para el diseño de su solución, considere que:

- No podrá emplear arreglos estáticos de más de una dimensión.
- No puede manipular un puntero con más de un índice.
- No puede emplear arreglos auxiliares, estáticos o dinámicos, para guardar los datos de los archivos.
- Los archivos solo se pueden leer una vez.

Descripción del caso

Se cuenta con un proyecto en C++ que permite gestionar los siguientes archivos de texto: `platos.csv` (código, nombre, precio y tipo), `repartidores.csv` (código, nombre, tipo de vehículo) y `pedidos.txt` (DNI, código del plato, cantidad de platos, código del repartidor). El proyecto ya se encuentra implementado: incluye las estructura, función `main`, operadores sobrecargados, solo falta implementar lo solicitado en las preguntas 1, 2 y 3.

`platos.csv`

```
AP-500,CHORIZOS COCKTAIL,12.90,APERITIVO
AP-410,ANTICUCHO,12.90,APERITIVO
...
```

`pedidos.txt`

```
12484697 AD-546 2 LAF361
12484697 PO-751 1 LAF361
...
```

`repartidores.csv`

```
JNV387,Justino Norabuena Virginia Karina,Motocicleta
PRT150,Pairazaman Raffo Tatiana Delicia,Bicicleta
...
```

Para la lectura de los archivos antes mencionados, se han diseñado las siguientes estructuras:

1: `Plato.hpp`

```
struct Plato{
    char *codigo;
    char *nombre;
    double precio;
    char *tipo;
};
```

2: `Repartidor.hpp`

```
struct Repartidor{
    char *codigo;
    char *nombre;
    char *vehiculo;
};
```

3: `Pedido.hpp`

```
struct Pedido{
    int dni_cliente;
    char codigo_plato[7];
    int cantidad_platos;
    char codigo_repartidor[7];
};
```

La estructura `Plato` cuenta con los siguientes operadores sobrecargados:

- `operator>>` Permite leer los datos de una línea de un archivo de texto que posee un formato como el del archivo `platos.csv`. Tipo de retorno: `bool`. Retorna `false` si se llega al fin del archivo, en cualquier otro caso retorna retorna `true`. Ejemplo de uso: `archivo_de_platos >> plato`.
- `operator<<` Permite mostrar los datos de un plato en un flujo de salida. Tipo de retorno: `ostream &`. Ejemplo de uso: `cout << plato`.

La estructura `Repartidor` cuenta con los siguientes operadores sobrecargados:

- `operator>>` Permite leer los datos de una línea de un archivo de texto que posee un formato como el del archivo `repartidodes.csv`. Tipo de retorno: `bool`. Retorna `false` si se llega al fin del archivo, en cualquier otro caso retorna retorna `true`. Ejemplo de uso: `archivo_de_repartidodes >> repartidor`.
- `operator<<` Permite mostrar los datos de un repartidor en un flujo de salida. Tipo de retorno: `ostream &`. Ejemplo de uso: `cout << repartidor`.

La estructura `Pedido` cuenta con los siguientes operadores sobrecargados:

- `operator>>` Permite leer los datos de una línea de un archivo de texto que posee un formato como el del archivo `Pedidos.txt`. Tipo de retorno: `bool`. Retorna `false` si se llega al fin del archivo, en cualquier otro caso retorna retorna `true`. Ejemplo de uso: `archivo_de_pedidos >> pedido`.
- `operator<<` Permite mostrar los datos de un repartidor en un flujo de salida. Tipo de retorno: `ostream &`. Ejemplo de uso: `cout << pedido`.

Con el objetivo de almacenar el contenido de los archivos, se han creado 3 Tipos Abstractos de Datos (TAD): `ConjuntoDePlatos`, `ConjuntoDeRepartidores` y `ConjuntoDePedidos`.

4: `ConjuntoDePlatos.hpp`

```
struct ConjuntoDePlatos{
    Plato *conjunto_de_datos;
    int cantidad;
};
```

5: `ConjuntoDeRepartidores.hpp`

```
struct ConjuntoDeRepartidores{
    Repartidor *conjunto_de_datos;
    int cantidad;
};
```

6: `ConjuntoDePedidos.hpp`

```
struct ConjuntoDePedidos{
    int **enteros;
    char ***cadenas;
    double **reales;
};
```

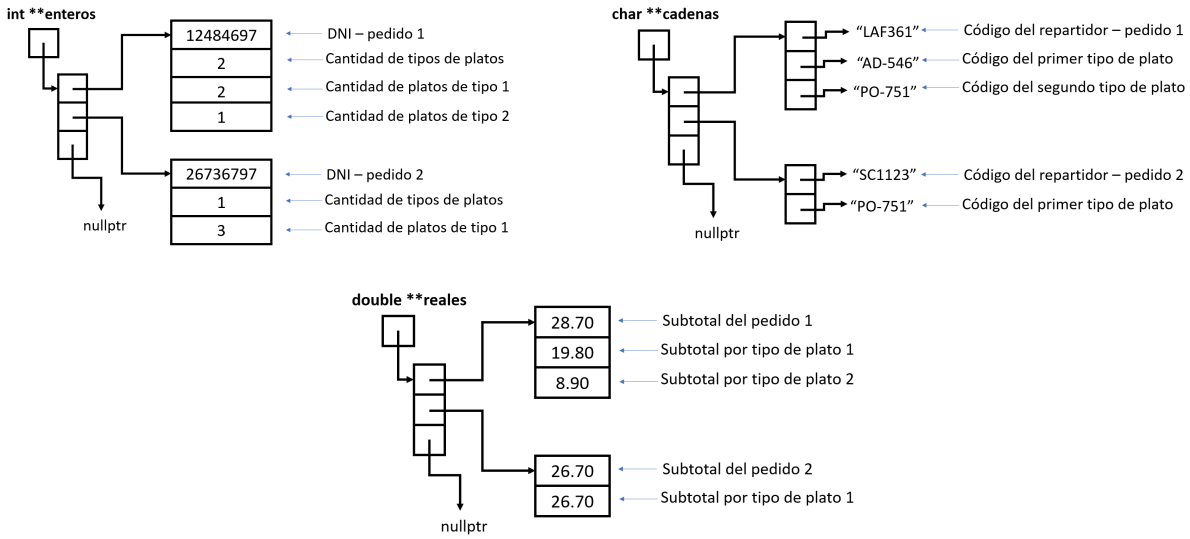
La estructura `ConjuntoDePlatos` cuenta con los siguientes operadores sobrecargados:

- `operator>>` Permite leer los datos de un archivo de texto que posee un formato como el del archivo `platos.csv` usando el método de asignación exacta de memoria. Para la lectura usa la estructura `Plato`, así como su operador >> sobrecargado. Tipo de retorno: `void`. Ejemplo de uso: `archivo_de_platos >> conjuntoDePlatos`.
- `operator+ =` Busca el código del plato en el conjunto de datos y retorna su precio. Para la búsqueda del plato utiliza el método de búsqueda secuencial. Tipo de retorno: `double`. En caso no se encuentre el plato, el operador debe retornar 0. Ejemplo de uso: `double precio_plato = conjuntoDePlatos + = codigo_del_plato`.
- `operator==` Busca el código del plato en el conjunto de datos y retorna su nombre. Para la búsqueda del plato utiliza el método de búsqueda secuencial. Tipo de retorno: `char *`. En caso no se encuentre el plato, el operador debe retornar `nullptr`. Ejemplo de uso: `char *nombre_del_plato = conjuntoDePlatos == codigo_del_plato`.

La estructura `ConjuntoDeRepartidores` cuenta con los siguientes operadores sobrecargados:

- `operator>>` Permite leer los datos de un archivo de texto que posee un formato como el del archivo `repartidores.csv` usando el método de asignación exacta de memoria. Para la lectura usa la estructura `Repartidor`, así como su operador >> sobrecargado. Tipo de retorno: `void`. Ejemplo de uso: `archivo_de_repartidores >> conjuntoDeRepartidores`.

En TAD `ConjuntoDePedidos` almacena un conjunto de pedidos, internamente está implementado a través de 3 bloques de datos: `enteros`, `cadenas` y `reales`. Tal como se muestra en la figura.



La estructura `ConjuntoDePedidos` cuenta con los siguientes operadores sobrecargados:

- `operator>>` primera sobrecarga. Permite leer los datos de un archivo de texto que posee un formato como el del archivo `pedidos.txt`. Para la lectura usa la estructura `Pedido`, así como su operador >> sobrecargado. Tipo de retorno: `void`. Ejemplo de uso: `archivo_de_pedidos >> conjuntoDePedidos`.

- `operator<<` Permite incluir un pedido leído a usando la estructura `Pedido` en la estructura `ConjuntoDePedidos`. Tipo de retorno: `void`. Ejemplo de uso: `buffer_de_conjuntoDePedidos << pedido`.
- `operator>>` segunda sobrecarga. Permite pasar de una estructura los datos de una estructura `ConjuntoDePedidos` a otra estructura `ConjuntoDePedidos` garantizando que la memoria reservada para la segunda estructura posee la cantidad de memoria exacta. Tipo de retorno: `void`. Ejemplo de uso: `buffer_de_conjuntoDePedidos >> conjuntoDePedidos`.
- `operator++` Permite calcular el precio de cada uno de los pedidos de una estructura de tipo `ConjuntoDePedidos`. Para obtener el precio de un plato deberá cargar el conjunto de platos usando la estructura `ConjuntoDePlatos` y buscar cada plato usando el operador `+ =` que se encuentra implementado en la estructura `ConjuntoDePlatos`. Tipo de retorno: `void`. Ejemplo de uso: `++conjuntoDePedidos`.
- `operator!` Permite cambiar el código del plato por el nombre del plato dentro de `ConjuntoDePedidos`. Para obtener el nombre de un plato deberá cargar el conjunto de platos usando la estructura `ConjuntoDePlatos` y buscar el nombre de cada plato usando el operador `==` que se encuentra implementado en la estructura `ConjuntoDePlatos`. Tipo de retorno: `void`. Ejemplo de uso: `!conjuntoDePedidos`
- `operator*` Recorre cada uno de los pedidos y genera por cada pedido un archivo de texto conteniendo los datos del pedido. Tipo de retorno: `void`. Ejemplo de uso: `*conjuntoDePedidos`

Pregunta 1

Se cuenta con el proyecto `2024_2_Lab02_Dev` que contiene parte de la implementación de las estructuras `Plato`, `Repartidor`, `Pedido`, `ConjuntoDePlatos` y `ConjuntoDeRepartidores`. La función `main` de este proyecto ejecuta pruebas sobre los TAD `ConjuntoDePlatos` y `ConjuntoDeRepartidores`. Se le pide que:

- (3 puntos) Realice la implementación de operador >> del TAD `ConjuntoDePlatos`. Para la lectura deberá utilizar necesariamente la estructura `Plato`, así como la sobrecarga del operador >> de dicha estructura.
- (3 puntos) Realice la implementación de operador >> del TAD `ConjuntoDeRepartidores`. Para la lectura deberá utilizar necesariamente la estructura `Repartidor`, así como la sobrecarga del operador >> de dicha estructura.

Pregunta 2

Se cuenta con el proyecto `2024_2_Lab02_Pedidos` que contiene parte de la implementación de la estructura `ConjuntoDePedidos`. Este proyecto debe usar la librería denominada `lib_platos_repartidores.a` que implementa todas las estructuras antes mencionadas, inclusive las sobrecargas de la pregunta 1, excepto el TAD `ConjuntoDePedidos`. La librería se encuentra en la carpeta del proyecto, usted debe añadirla al proyecto. Se le pide que:

- (2 puntos) Realice la implementación de operador << del TAD `ConjuntoDePedidos`. Para facilitar la implementación de esta sobrecarga ya se encuentra implementada la función `buscar_cliente` que retorna el índice en donde se encuentra un cliente.
- (2 puntos) Realice la implementación de operador >>, la segunda sobrecarga del TAD `ConjuntoDePedidos`. Cuando implemente esta sobrecarga, el bloque para los reales debe apuntar a `nullptr`.
- (2 puntos) Realice la implementación de operador >>, la primera sobrecarga del TAD `ConjuntoDePedidos`. Para la lectura deberá utilizar necesariamente la estructura `Pedido`, así como la sobrecarga del operador >> de dicha estructura.

Luego de desarrollar esta pregunta los test `test01`, `test02`, `test03` y `test04` de la función `main`, deberían funcionar.

Pregunta 3

Completar la implementación de los demás operadores sobrecargados del TAD `ConjuntoDePlatos` en el proyecto `2024_2_Lab02_Pedidos`. Las demás sobregargas solo funcionarán si es que se realizar la carga de los pedidos por lo que deberá implementar antes lo solicitado en la pregunta 2, si no hace esto, esta pregunta no podrá ser calificada. Se le pide que:

- (3 puntos) Realice la implementación de operador ++, tanto su versión en prefijo como en posfijo. La prueba de esta sobrecarga se puede realizar con el método `test05` de la función `main`.
- (3 puntos)Realice la implementación de operador !. La prueba de esta sobrecarga se puede realizar con el método `test06` de la función `main`.
- (2 puntos)Realice la implementación de operador $*$. Para facilitar esta implementación se pone a su disposición la función `obtener_nombre_archivo` que permite obtener del archivo que se creará para el cliente dado un número de DNI. De forma similar se dispone de la función `imprimir_linea` que permite escribir una línea en el archivo. La prueba de esta sobrecarga se puede realizar con el método `test07` de la función `main`. El archivo debe tener el siguiente formato:

12484697.txt

```
========================================
    D.N.I. CLIENTE: 12484697
 CODIGO REPARTIDOR: LAF361
========================================
ITEM PLATO                 CANT SUBTOTAL
----------------------------------------
  01 CHOCLO.................2 19.80
  02 MAZAMORRA MORADA.......1  8.90
  03 ANTICUCHO DE CORAZON.....3 77.70
  04 SPRITE 1 LITRO...........3 46.50
----------------------------------------
                     SUBTOTAL: 152.90
                       I.G.V.: 027.52
                        TOTAL: 180.42
========================================
```

Profesores del curso: Miguel Guanira, Rony Cueva, Erasmo Gómez, Andrés Melgar, Erick Huiza

Pando, 13 de septiembre de 2024