Pipeline sacerdote Playmobil

DIARIO DE TRABAJO

- Inicio del proceso de Pipeline -

Lunes 7 de junio:

Seguido de la clase en donde se explicó el funcionamiento de las consignas a realizar, decidimos juntarnos virtualmente para comentar la división de las tareas del código y así poder llevar a cabo la realización del mismo de una manera rápida y organizada.

Miércoles 9 de junio:

Nos unimos virtualmente y comentamos el planteo de las consignas del trabajo práctico entre las tres, para así entender de qué forma se iba a mover el código en relación a los distintos procesos.

- 1º Proceso Pipeline (Fabricante) -

Jueves 10 de junio:

Luego de tener una idea de que función iba a cumplir el código del proceso 1 decidimos comenzar la codificación del mismo.
Como la consigna lo decía, este proceso iba a recibir el archivo llamado “Paquetes.bin”, sabiendo esto decidimos despersistir estos datos que traía para poder trabajar con ellos.

También decidimos hacer una función de persistencia para que esta almacene los datos que íbamos a cargar en el archivo “Componentes”, el cual le iba a llegar al proceso 2.

Funciones creadas:

void despersistencia (PaqueteMateriaPrima arr[], int* val)

void muestraarreglo (PaqueteMateriaPrima arr[], int val)

void persistencia (Componente* dato)

Problemas Presentados:

Nos surgen problemas con la utilización de **¡feof** para abrir el archivo.

Solución:

Investigamos en videos de distintas personas opinando sobre la utilización y funcionamiento del mismo por lo que optamos en este caso no ponerlo.

Lunes 14 de junio

Decidimos comenzar a realizar la función para la carga del arreglo de tipo ‘Componente’, para ubicarnos en el contexto, la estructura del mismo estaba compuesta por: **Nombre/ Falla/ Costo/ Calidad.**

Para los **Nombres** usamos ‘strcpy’, las **Fallas** según la consigna debían ser aleatorias entre 0 y 2, por lo que usamos ‘srand (time(NULL))’, por otro lado, la **Calidad** la iba a recibir en forma de copia del archivo “Paquetes.bin” y por último, el **Costo** que debido a que este tenía varios pasos a realizar, decidimos hacer una función aparte que luego iba a ser llamada dentro de la carga.

La función de **Costo** dependía del nombre del paquete que iba a recibir, ya que son 3 nombres, se crean tres sentencias de ‘if’. En las mismas lo que se hace es el cálculo del costo, dependiendo del nombre del producto estos varían. En términos generales cada bucle hace la cantidad de materia total del producto multiplicada por el costo total por gramo **[**costo dividido cantidad de materia, del archivo de “Paquetes.bin”**]** sumado eso al ‘costo del proceso’

Funciones creadas:

void carga (PaqueteMateriaPrima arr[], int val)

void costocomponente (Componente* a, float costo_g)

Problemas Presentados:

- Con la sentencia -> costoxGramo = arr[i].costo / arr[i].cantMateria.

- Con la función ‘strcmp’ de la librería <string.h>.

- Sentencias de ‘ if ’ anidados.

Solución:

Logramos darnos cuenta al ejecutar el código que la sentencia de ‘costoxGramo’ no podría nunca estar ubicada por debajo de la resta de materia del paquete, ya que esta debía ser calculada antes de utilizar los mismos.

Con respecto al uso del ‘strcmp’, nos fijamos el GitHub el uso del mismo y el error lo encontramos en las comillas

Sentencias de if anidados son el primer método que se nos ocurrió para llevar a cabo la función de la carga, pero no nos funcionó ya que, si la situación de no tener materia suficiente para crear el muñeco se presentaba 2 veces, en la función no se volvía a crear la muñeca (que era el primer componente que tenía que crear) sino que creaba los siguientes, por lo que decidimos utilizar el ‘flag’ para solucionar ese problema y así, logró funcionar correctamente.

Miércoles 16 de junio

Decidimos mostrar el arreglo que creamos para ver si tenía alguna falla.

Por otro lado decidimos hacer una función más para volver al código un poco más eficiente y "reutilizable", en cuanto a la carga en la parte que repetíamos el mismo bucle.

En la función de carga lo que hacemos es que, si le alcanza la materia para realizar la parte del muñeco entra al bucle y se dirige a esta función nueva, la cual se encarga de ponerle los datos que debíamos crear en el nuevo arreglo de tipo 'Componentes'. (**Nombre/ Falla/ Costo/ Calidad**)

Funciones creadas:

void muestracomponentes ()

void mostrarcomponente (Componente a)

int cargaporcomponente (PaqueteMateriaPrima arr[], int pos, char nombre[], Componente* p, int materia, float* costoxgramo)

Problemas Presentados:

- Problemas con el planteo de la función de carga sin la utilización del flag.

Solución:

Investigamos qué forma nos podía solucionar el problema y terminamos decidiendo en una utilización nuevamente del flag.

- 2º Proceso Pipeline (Ensamblador)-

Lunes 21 de junio

En primer lugar, declaramos los registros Componente, en el que vamos a recibir los datos que nos llegan del archivo de componentes, y el de PlayMobil, en el que vamos a ingresar los datos de los productos finales.

Luego recibimos el archivo de componentes y guardamos los componentes en tres distintos arreglos según su tipo, es decir que el primer arreglo sólo va a tener a los **muñecos**, el segundo va a tener a las **biblias** y el tercero a las **copas de vino.**

Funciones creadas:

void cargar_arreglos(Componente doll[], Componente wine[], Componente bible[], int*d, int*w, int*b);

Problemas Presentados:

Pusimos el fread() como condición del while y como primera línea después del bucle, entonces el primero leía un Componente del archivo y el segundo leía otro componente sobre el mismo.

Solución:

Consultamos con la ayudante de la comisión, quien nos ayudó a encontrar el error y simplemente borramos el segundo fread().

Martes 22 y Miércoles 23 de junio

Después, en la función "dos_fallas", vamos a recorrer los arreglos y mandamos los costos de los componentes que tengan dos fallas al archivo de pérdidas. Los componentes que tengan 1 o 0 fallas, decidimos que van a ir a otro arreglo, el cual va a quedar como definitivo, es decir que ese va a ser con el que vamos a trabajar de ahora en más.

Ahora que ya tenemos sólo los componentes con los que vamos a trabajar, vamos a proceder a cargar el PlayMobil.

Vamos a recorrer el archivo de muñecos y llamamos a la función "calidad". En esta función, se va a buscar en los arreglos de biblias y de copas, componentes que coincidan con la calidad del muñeco, si resulta ser que hay un muñeco, una biblia y una copa de la misma calidad, la función va a devolver esa calidad, sino, devuelve 'n'.

En la función de carga del PlayMobil, si la función de calidad no devuelve 'n', va a cargar esa calidad en el campo calidad del PlayMobil.

Funciones creadas:

void dos_fallas (Componente total[], Componente definitivo[], int* validos);

void mandar_a_perdidas(Componente arr[], int pos);

Problemas Presentados:

Tuvimos dificultades para eliminar los componentes con dos fallas de los arrays de componentes.

Solución:

Lo resolvimos creando otro array al que pasamos los componentes que no tenían dos fallas y usando ese array como el definitivo.

Jueves 24 de junio

Como nosotras pasamos por parámetro las posiciones de la biblia y la copa que usamos, vamos a ir a la función de “costo”, donde se va a sumar, los costos de los tres componentes y después, en la función "costo_pm" va a sumarle a ese costo, 14, 12 o 10, según la calidad del muñeco. Eso va a ir al campo de costo del PlayMobil. Cuando ya esté el PlayMobil cargado, va a borrar del arreglo de biblias y de copas, los componentes que ya fueron usados.

Por último, decidimos hacer una persistencia del arreglo de PlayMobil en el archivo de Productos Finales, y una muestra del mismo para corroborar que el código funcione correctamente

Funciones creadas:

float costo_final (Componente doll[], int posd, Componente bible[], int posb, Componente wine[], int posw);

float costo_pm (Componente doll[], int posd, Componente bible[], int posb, Componente wine[], int posw);

char calidad (Componente doll[], int pos1, Componente bible[], int* pos2, Componente wine[], int* pos3, int b, int w);

void cargar_PlayMobil (PlayMobil arr[], int* val, Componente doll[], Componente bible[], Componente wine[], int d, int b, int w);

void persistir_PlayMobil (PlayMobil arr[], int validos);

Problemas Presentados:

Al principio habíamos entendido mal y le sumamos el costo de ensamble a cada componente por separado.

Solución:

Cambiamos las funciones que habíamos hecho y le sumamos el costo de ensamble a la suma de los costos de los componentes.

- 3º Proceso Pipeline (Venta de Lote) -

Viernes 25 de junio

Lo primero que hicimos fue declarar las estructuras que deberíamos cargar para luego persistir, “Contaduría” y “DetalleLote”, además de “PlayMobil” la cual corresponde a los datos que nos llegan en el archivo desde el segundo proceso.

Planteamos los prototipados de las funciones que creímos necesarias:

- Dos funciones para las despersistencias de ambos archivos junto con los arreglos de tipos correspondientes y sus validos.

- Función para la carga general de los Playmobil y parte de los campos de Contaduría.

- Función para calcular los precios.

- Función para las ganancias.

- Dos funciones para la persistencia de los nuevos archivos creados.

Declaramos las constantes con los nombres de los archivos para facilitar el manejo de los dos recibidos, y los dos a creados.

Funciones creadas:

void despersistenciaPM (PlayMobil [], int*l);

void despersistenciaP (float [], int*);

void cargaDetalleLote (PlayMobil [], int , DetalleLote [], Contaduria*);

void precioF (DetalleLote [], int , Contaduria*);

void gananciaB (DetalleLote [], int , Contaduria*);

void persistenciaDL (DetalleLote [], int);

void persistenciaC (Contaduria);

Problemas Presentados:

Tuvimos dificultades con la despersistencia de los archivos, al igual que en el “proceso 1”, ya que algunos datos se pasaban al arreglo dos veces. Debido a esto, intentamos usar como condición dentro del while a (!feof) pero no funcionó.

Solución:

Lo resolvimos utilizando en lugar de (!feof), una condición con (fread > 0). Además , agregamos una función más para mostrar ambos arreglos, el de PlayMobil con los datos despersistidos del archivo “ProductosFinales.bin”, y el float con los datos despersistidos del archivo Perdidas.bin.

Domingo 27 de junio

Desarrollamos la función de carga del arreglo de tipo DetalleLote, la cual se encarga de cargar parte de los campos de todas las estructuras de lotes dependiendo la calidad, y parte de los campos de Contaduría, previamente declarada en el main.

La función recibe un muñeco PlayMobil y, dependiendo de su calidad, avanza en el campo ‘cantidad’ de la estructura de DetalleLote correspondiente a la misma calidad (a,b o c).

Luego suma al campo de ‘costoLote’ de la estructura de Contaduría el precio de dicho PlayMobil, y avanza en el contador de ‘cantTotal’, que corresponde a un campo de esa misma estructura.

Decidimos continuar con el desarrollo de la función ‘precioF’,la cual calcula el precio final de cada PlayMobil dependiendo de su calidad. Como primero realiza el cálculo de un promedio por paquete, y luego de suma el porcentaje correspondiente (125%, 123% o 120%).

Por último, hicimos el cálculo manual de algunos costos para asegurarnos que los resultados estén bien.

Funciones creadas:

void cargaDetalleLote (PlayMobil arrP[], int valP, DetalleLote arr[], Contaduria* c)

void precioF (DetalleLote arr[], int val, Contaduria* c)

Problemas Presentados:

1- Se nos dificulto lograr que cada PlayMobil se cargará en una celda del arreglo correspondiente a su calidad.

2- Por otro lado, lo que nos sucedió fue que, los campos de la estructura de Contaduría no lograban modificarse con los valores correctos.

3- Por último en cuanto al promedio por paquete, nos pasó que era siempre igual, por lo que los precios finales eran siempre los mismos.

Solución:

1- Decidimos resolverlo asignando a cada celda del arreglo una calidad, y con bucles if buscar a cuál correspondía.

2- Nos pusimos de acuerdo en resolverlo inicializando los campos a modificar en 0 dentro de la función.

3- Lo resolvimos llamando a la función que calcula el precio en el main en vez de llamarla dentro de la carga de DetalleLote (agregando las modificaciones necesarias a los parámetros).

Lunes 28 de junio

Desarrollamos la función de pérdidas utilizando un algoritmo de suma de todas las posiciones del arreglo previamente cargado con los datos del archivo Perdidas.bin.

Calculamos manualmente los valores que nos mostraba por pantalla para asegurarnos de que el resultado sea el correcto.

Desarrollamos la función de ganancia la cual calcula la **ganancia bruta**, **los costos**, los compara e informa por pantalla si existe o no una ganancia.

Calculamos nuevamente los calores que nos mostraba por pantalla de forma manual para asegurarnos de que el resultado sea el correcto.

Funciones creadas:

void perdidasFun (Contaduria* c, float arr[], int val)

void gananciaB (DetalleLote arr[], int val, Contaduria* c)

Problemas Presentados:

1- La función cargaba valores basura al campo de costoPerdidas de la variable de contaduría.

2- La función informaba que no se generaban ganancias.

Solución:

1- Debatimos entre las tres y concluimos en inicializar este campo en 0 antes de asignarle el nuevo valor.

2- Lo resolvimos inicializando el arreglo de pérdidas en 0 para comprobar que sin pérdidas habría ganancias, o si era un problema relacionado al cálculo de las ganancias en sí.

Martes 29 de junio

Desarrollamos las funciones de persistencia, tanto del arreglo cargado con los lotes como de la estructura de contaduría.

Por último, para volver al código un poco más eficiente y "reutilizable", declaramos como constantes los valores del porcentaje que se agrega a cada producto PlayMobil dependiendo de su calidad, para poder ser modificados fácilmente en un futuro.

Para finalizar el desarrollo del proceso 3, decidimos hacer una llamada de todas las funciones, sin comentar, en el main para una prueba final. En esta se hace una muestra del archivo para asegurarnos de que fuera correctamente persistido.

Funciones creadas:

void persistenciaDL (DetalleLote arr[], int val)

void persistenciaC (Contaduria c)

Problemas Presentados:

Dándole un último chequeo al código, encontramos discrepancias con algunos parámetros.

Solución:

Decidimos resolverlo haciendo coincidir el prototipado de las funciones con las nuevas modificaciones, lo cual nos pareció la forma más eficaz.

CONCLUSIÓN:

Consideramos que nuestro primer trabajo en grupo fue una gran experiencia para cada una, nos generó expectativas sobre el trabajo en equipo en un futuro y nos ayudó a ver las formas de efectuar un código, de mil maneras distintas, lo que hizo que se ampliaran nuestros conocimientos. En cuanto a la resolución de los problemas presentados podemos decir, que nos manejamos de una forma tranquila, para así poder ir de a partes y ver donde estaba el error. Logramos solucionarlos a cada uno de ellos con éxito.
