/\*

Ejercicio

La plataforma de ventas online FreeMarket nos encarga el desarrollo de una aplicación que se ejecute de manera offline

para procesar los datos de los productos allí publicados y vendidos.

La plataforma posee, por separado, los datos de los productos que se venden al contado de aquellos que se venden en

cuotas. De los productos que se venden al contado, se conoce su nombre, código, código de categoría y precio. De los

productos que se venden en cuotas se conoce, nombre, código, código de categoría y precio de venta en 3, 6 y 12 cuotas.

Tener en cuenta que existen productos que se venden tanto al contado como en cuotas, por lo que estarán presentes en

ambos conjuntos.

contado:

typedef struct{

float tresCuotas;

float seisCuotas;

float doceCuotas;

} Cuotas;

typedef struct{

string nombre;

int codigo;

int codCategoria;

float precioContado;

Cuotas precioVenta;

} Producto;

typedef struct{

int codigo;

string nombre;

} Categoria; //30 categorias, ordenado por código. No completo (NO PUP)

Tanto el código del producto como el de la categoría son datos de tipo entero.

La plataforma dispone también de un conjunto ordenado por código con las 30 categorías de productos. Este conjunto

posee código y nombre de categoría.

En base a la situación establecida, se pide:

a. Desarrollar las estructuras que permitan resolver el problema.

b. Desarrollar un módulo que permita unificar de forma ordenada, los conjuntos de productos (contado y cuotas),

teniendo en cuenta que el conjunto resultante debe poseer una única entrada para cada producto con el precio de

contado, el precio de la mayor cantidad de cuotas y la cantidad de cuotas que se haya definido.

typedef struct{

string nombre;

int codigo;

int codCategoria;

float precioContado;

float precioMayorCuota;

int cantCuotas;

} ProductoFinal;

c. Desarrollar un módulo que permita obtener los 3 productos de mayor precio contado, tomando como entrada el

conjunto resultante del punto “b”. Para realizarlo no puede ordenar el conjunto de entrada.

d. Imprimir por pantalla el listado de productos obtenido en el punto “c” agregando los datos de la categoría de cada

producto. Para obtener los datos de cada categoría, no puede realizar búsquedas secuenciales.

e. Desarrolle el diagrama de flujo del programa principal.

\*/

#include <iostream>

using namespace std;

#define CANT\_PRODUCTOS 50

#define CANT\_MAYORES 3

#define CANT\_CATEGORIAS 30

//a)

typedef struct

{

float tresCuotas;

float seisCuotas;

float doceCuotas;

} Cuotas;

typedef struct

{

string nombre;

int codigo;

int codCategoria;

float precioContado;

Cuotas precioVenta;

} Producto;

typedef struct

{

int codigo;

string nombre;

} Categoria; //30 categorias, ordenado por código. No completo (NO PUP)

//b

typedef struct

{

string nombre;

int codigo;

int codCategoria;

float precioContado;

float precioMayorCuota;

int cantCuotas;

} ProductoFinal;

void ordenarPorCodigo(Producto arr[], int n)

{

int i = 0, j;

Producto aux;

bool ordenado = false;

while (i < n && !ordenado)

{

ordenado = true;

for (j = 0; j < n - i - 1; j++)

{

if (arr[j].codigo > arr[j + 1].codigo)

{

aux = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = aux;

ordenado = false;

}

}

i++;

}

}

void obtenerMayorCuota(Producto producto, float &precio, int &cantCuotas)

{

if (producto.precioVenta.doceCuotas != NULL)

{

precio = producto.precioVenta.doceCuotas;

cantCuotas = 12;

}

else

{

if (producto.precioVenta.seisCuotas != NULL)

{

precio = producto.precioVenta.seisCuotas;

cantCuotas = 6;

}

else

{

if (producto.precioVenta.tresCuotas != NULL)

{

precio = producto.precioVenta.tresCuotas;

cantCuotas = 3;

}

else

{

precio = NULL;

cantCuotas = NULL;

}

}

}

return;

}

void apareo(Producto vecA[], int n, Producto vecB[], int m, ProductoFinal vecC[], int &k)

{

//contadores para la posicion de los vectores A y B

int i = 0, j = 0;

//contador para la posicion del vector resultante

k = 0;

float precio;

int cantCuotas;

//mientras pueda comparar valores

while (i < n && j < m)

{

if (vecA[i].codigo == vecB[j].codigo)

{

vecC[k].nombre = vecA[i].nombre;

vecC[k].codigo = vecA[i].codigo;

vecC[k].codCategoria = vecA[i].codCategoria;

vecC[k].precioContado = vecA[i].precioContado;

obtenerMayorCuota(vecB[j], precio, cantCuotas);

vecC[k].precioMayorCuota = precio;

vecC[k].cantCuotas = cantCuotas;

i++;

j++;

}

if (vecA[i].codigo < vecB[j].codigo)

{

vecC[k].nombre = vecA[i].nombre;

vecC[k].codigo = vecA[i].codigo;

vecC[k].codCategoria = vecA[i].codCategoria;

vecC[k].precioContado = vecA[i].precioContado;

vecC[k].precioMayorCuota = NULL;

vecC[k].cantCuotas = NULL;

i++; //me muevo en el vector A

}

else if (vecA[i].codigo > vecB[j].codigo)

{

vecC[k].nombre = vecB[i].nombre;

vecC[k].codigo = vecB[i].codigo;

vecC[k].codCategoria = vecB[i].codCategoria;

vecC[k].precioContado = vecB[i].precioContado;

obtenerMayorCuota(vecB[j], precio, cantCuotas);

vecC[k].precioMayorCuota = precio;

vecC[k].cantCuotas = cantCuotas;

j++;

}

k++;

}

//Para kos valores restantes de los vectores

while (i < n)

{

vecC[k].nombre = vecA[i].nombre;

vecC[k].codigo = vecA[i].codigo;

vecC[k].codCategoria = vecA[i].codCategoria;

vecC[k].precioContado = vecA[i].precioContado;

vecC[k].precioMayorCuota = NULL;

vecC[k].cantCuotas = NULL;

i++; //me muevo en el vector A

k++;

}

while (j < m)

{

vecC[k].nombre = vecB[i].nombre;

vecC[k].codigo = vecB[i].codigo;

vecC[k].codCategoria = vecB[i].codCategoria;

vecC[k].precioContado = vecB[i].precioContado;

obtenerMayorCuota(vecB[j], precio, cantCuotas);

vecC[k].precioMayorCuota = precio;

vecC[k].cantCuotas = cantCuotas;

j++;

k++;

}

}

void obtenerMayores(ProductoFinal arrA[], int cantA, ProductoFinal mayores[])

{

for (int i = 0; i < cantA; i++)

{

if (arrA[i].precioContado > mayores[0].precioContado)

{

mayores[2] = mayores[1];

mayores[1] = mayores[0];

mayores[0] = arrA[i];

}

else

{

if (arrA[i].precioContado > mayores[1].precioContado)

{

mayores[2] = mayores[1];

mayores[1] = arrA[i];

}

else

{

if (arrA[i].precioContado > mayores[2].precioContado)

{

mayores[2] = arrA[i];

}

}

}

return;

};

/\*

d. Imprimir por pantalla el listado de productos obtenido en el punto “c” agregando los datos de la categoría de cada

producto. Para obtener los datos de cada categoría, no puede realizar búsquedas secuenciales.

\*/

void imprimirListado(ProductoFinal arr[], int cant, Categoria categorias)

{

cout << "Listado:" << endl;

for (int i = 0; i < cant; i++)

{

cout << arr[i].nombre << "\t\t" << arr[i].codigo << "\t\t" << arr[i].codCategoria << "\t\t" << /\*nombreCategoria\*/ << "\t\t" << arr[i].precioContado << "\t\t" << arr[i].precioMayorCuota << "\t\t" << arr[i].cantCuotas << endl

<< endl;

}

}

int main()

{

Producto contado[CANT\_PRODUCTOS];

Producto cuotas[CANT\_PRODUCTOS];

Categoria categorias[CANT\_CATEGORIAS];

//b

ordenarPorCodigo(contado, CANT\_PRODUCTOS);

ordenarPorCodigo(cuotas, CANT\_PRODUCTOS);

int cantProductosFinales = CANT\_PRODUCTOS + CANT\_PRODUCTOS;

ProductoFinal productosFinales[cantProductosFinales];

apareo(contado, CANT\_PRODUCTOS, cuotas, CANT\_PRODUCTOS, productosFinales, cantProductosFinales);

//c

ProductoFinal mayoresPrecios[CANT\_MAYORES];

obtenerMayores(productosFinales, cantProductosFinales, mayoresPrecios);

//d

imprimirListado(mayoresPrecios, CANT\_MAYORES, )

return 0;

 }

