Centro Universitario de Ciencias Exactas e Ingenierías (CUCEI)

Actividad de aprendizaje 2:
Anidación estructural: Registros con Arreglos, Arreglos de Registros y Arreglos de Objetos

Víctor Agustín Díaz Méndez
Ingeniería en Informática

Estructura de Datos I (Seccion D12)
Profesor: Dr. Gutierrez Hernandez Alfredo

Para la realizacion de esta actividad implemente 3 clases y un registro.

Clases:

- Menu
- Date
- productsCollection

Registro:

- Product

La clase “Date”

Esta clase consta de 4 metodos publicos:

```
void setDate(int&, int&, int&);
int getDay();
int getMonth();
int getYear();
```

El método “getDay” guarda los datos de las fechas y realiza ahí mismo su validación, tomando en cuenta también si es año bisiesto y en caso de que la fecha no sea valida lanza una excepción especificado que la fecha no es valida.

```
void Date::setDate(int &d, int &m, int &y)
{


    if(d>0 && m>0){
        if((m==4 || m==6 || m==9 || m==11) && d<=30){
            saveDate(d, m, y);
        }
        else{
            if((m==1 || m==3 || m==5 || m==7 || m==8 || m==10 || m==12) &&
d<=31){
                saveDate(d, m, y);
            }
            else{
                if((y%4) != 0){
                    //Normal year
                    if(d<=28 && m==2){
                        saveDate(d, m, y);
                    }
                    else{
                        throw DateException("ERROR, INVALID DATE");
                    }
                }
                else{
```

```
                        //Leap year
                        if(d<=29 && m==2){
                            saveDate(d, m, y);
                        }
                        else{
                            throw DateException("ERROR, INVALID DATE");
                        }
                    }
                }
            }
        } else {
            throw DateException("ERROR, INVALID DATE");
        }
}
```

Y para guardar la fecha solo la copia a los atributos de la clase para dia, mes y año.

```
int Date::saveDate(int &d, int &m, int &y){
    day=d;
    month=m;
    year=y;
}
```

Los métodos “getDay”, “getMonth” y “getYear” funcionan de una forma muy parecida, donde cada uno hace una validacion para verificar que la fecha no este vacia, en caso de que lo estuviera arroja un excepcion advirtiendo insuficiencia de datos.

```
int Date::getYear()
{
    if(isDate()){
        return year;
    }
    else {
        throw DateException("ERROR, BUFFER UNDERRUN");
    }
}
```

El registro Producto

El registro producto es muy sencillo, solo fue necesario definir los atributos de los que constaría e incluir la clase Date para poder definir un atributo de su tipo.

```
struct Product
{
        string name;
```

```
        string barCode;
        float weight;
        Date enterDate;
        float wholesalePrice;
        float retailPrice;
        int stock;
};
```

La clase ProductsCollection

Esta clase puede ser una de las mas complejas de todas las que programe, para realizarla me base en la demostración de las listas que hizo el profesor en clase. Las principales diferencias se encontrarían en que utiliza un tipo de dato diferente a la lista que usamos en clase, que aunque se puede adaptar a cualquier tipo de dato, se necesitaba hacer la búsqueda de los productos por “codigo de barras” del producto, el cual es un atributo del tipo de dato que maneja la lista.

```
int ProductsCollection::findData(const string& prod)
{
    for(int i = 0; i <= last;    i++){
        if(data[i].barCode == prod){
            return i;
        }
    }
    return -1;
}
```

Para realizar la búsqueda por código de barras solo fue necesario modificar el método mostrado en clase, para que en lugar de comparar directamente con uno de los elementos de la clase lo hiciese con uno de sus atributos.

La clase Menu

Esta clase fue la que mas trabajo o tiempo me llevo programar, debido a que a diferencia de ProductsCollections, no tenia un ejemplo donde basarme así que tuve que pensar en cada uno de los métodos y su funcionamiento.

Tenemos cuatro metodos:

```
void showMenu();
void addProduct();
void getProduct();
void addProductStock();
```

El metodo “showMenu” tiene la tipica funcion de mostrar un menu.

El método “addProduct” tiene la función de mostrar un formulario para almacenar los productos, donde realiza la validación de si los datos admitidos son validos, si no lo son los vuelve a solicitar. Una vez capturados los almacena en colección de productos.
El método “getProduct” obtiene un producto especificado y disminuye en uno su existencia en el inventario (ya que se trata de un producto retirado).
El método “addProductStock” incrementa las existencias del producto especificado en una determinada cantidad, para realizar este método primero busca el producto en la colección, lo obtiene, incrementa las existencias de la copia obtenida, elimina el producto del inventario y copia la copia que tenemos.

Date.h

```
#ifndef DATE_H
#define DATE_H

#include <string>
#include <exception>

using namespace std;

class DateException : public std::exception {
	private:
		std::string msg;
	public:
		explicit DateException(const char* message) : msg(message) {}
		explicit DateException(const std::string& message) :msg(message) {}
		virtual ~DateException() throw () {}
		virtual const char* what() const throw () {
			return msg.c_str();
			}
	};

class Date {
	public:
		Date();
		virtual ~Date();
		void setDate(int&, int&, int&);
		int getDay();
		int getMonth();
		int getYear();
	protected:
		int day;
		int month;
		int year;
		void saveDate(int &, int&, int&);
		bool isDate();
	private:
	};

#endif // DATE_H
```

Date.cpp

```
#include <iostream>
#include "date.h"
```

```
Date::Date() {
	day=0;
	month=0;
	year=0;
	}

Date::~Date() {
	//dtor
	}

void Date::setDate(int &d, int &m, int &y) {


	if(d>0 && m>0) {
		if((m==4 || m==6 || m==9 || m==11) && d<=30) {
			saveDate(d, m, y);
			}
		else {
			if((m==1 || m==3 || m==5 || m==7 || m==8 || m==10 || m==12) &&
d<=31) {
				saveDate(d, m, y);
				}
			else {
				if((y%4) != 0) {
					//Normal year
					if(d<=28 && m==2) {
						saveDate(d, m, y);
						}
					else {
						throw DateException("ERROR, INVALID DATE");
						}
					}
				else {
					//Leap year
					if(d<=29 && m==2) {
						saveDate(d, m, y);
						}
					else {
						throw DateException("ERROR, INVALID DATE");
						}
					}
				}
			}
```

```
        }
    else {
        throw DateException("ERROR, INVALID DATE");
        }
    }

void Date::saveDate(int &d, int &m, int &y) {
    day=d;
    month=m;
    year=y;
    }

bool Date::isDate() {
    if(day != 0 && month != 0 && year != 0 ) {
        return true;
        }
    else {
        return false;
        }
    }

int Date::getDay() {
    if(isDate()) {
        return day;
        }
    else {
        throw DateException("ERROR, BUFFER UNDERRUN");
        }
    }

int Date::getMonth() {
    if(isDate()) {
        return month;
        }
    else {
        throw DateException("ERROR, BUFFER UNDERRUN");
        }
    }

int Date::getYear() {
    if(isDate()) {
        return year;
        }
```

```
    else {
        throw DateException("ERROR, BUFFER UNDERRUN");
        }
    }
```

Menu.h

```
#ifndef MENU_H
#define MENU_H

#include "productsCollection.h"

class Menu {
    public:
        Menu();
        virtual ~Menu();
        void showMenu();
        ProductsCollection pCollection;
    protected:
        void addProduct();
        void getProduct();
        void addProductStock();
    private:
    };

#endif // MENU_H
```

Menu.cpp

```
#include "menu.h"
#include <iostream>
#include <string>

#include <product.h>
using namespace std;

Menu::Menu() {
    pCollection.initialize();
    }

Menu::~Menu() {
    //dtor
    }
```

```
void Menu::showMenu() {
	short option;
	bool finished=false;
	do {
		cout << "\tINVENTARIO" << endl
			<< "1.-Añadir producto" << endl
			<< "2.-Retirar producto" << endl
			<< "3.-Añadir existencias a producto" << endl
			<< "0.-Salir" << endl
			<< "Elige una opción: ";
		cin >> option;
		cin.ignore();
		cout << endl << endl;

		switch(option) {
			case 1:
				addProduct();
				break;
			case 2:
				getProduct();
				break;
			case 3:
				addProductStock();
				break;
			case 0:
				finished=true;
				break;
			default:
				break;
			}
		}
	while(!finished);
	}

void Menu::addProduct() {
	Product product;
	int day, month, year;
	bool answerError;

	cout << "\tPRODUCTO" << endl;

	do {
		cout << "Codigo de barras: ";
```

```
        getline(cin, product.barCode);

        if(product.barCode.size()==11) {
            answerError = false;
            }
        else {
            cout << "ERROR, el codigo de barras no es de 11 digitos" << endl;
            answerError = true;
            }
        }
while(answerError);

do {
        cout << "Nombre: ";
        getline(cin, product.name);

        if(product.name != "") {
            answerError = false;
            }
        else {
            cout << "ERROR, por favor teclee el nombre" << endl;
            answerError = true;
            }

        }
while(answerError);

do {
        cout << "Peso (Kgs.): ";
        cin >> product.weight;

        if(product.weight >    0) {
            answerError = false;
            }
        else {
            cout << "ERROR, el peso no puede ser igual o menor a cero" << endl;
            answerError = true;
            }
        }
while(answerError);

do {
        cout << "Fecha " << endl;
```

```
        cout << "Dia: ";
        cin >> day;
        cout << "Month: ";
        cin >> month;
        cout << "Year: ";
        cin >> year;

        try {
            product.enterDate.setDate(day,month,year);
            answerError=false;
            }
        catch(DateException ex) {
            cout << "FECHA NO VALIDA" << endl;
            answerError=true;
            }
        }
while(answerError);

do {
        cout << "Precio mayoreo: ";
        cin >> product.wholesalePrice;

        if(product.wholesalePrice > 0) {
            answerError = false;
            }
        else {
            cout << "ERROR, el precio no puede ser menor o igual a cero" << endl;
            answerError = true;
            }
        }
while(answerError);

do {
        cout << "Precio menudeo: ";
        cin >> product.retailPrice;

        if(product.retailPrice > 0) {
            answerError = false;
            }
        else {
            cout << "ERROR, el precio no puede ser menor o igual a cero" << endl;
            answerError = true;
            }
```

```
        }
    while(answerError);

    do {
        cout << "Existencias: ";
        cin >> product.stock;
        if(product.stock>=0) {
            answerError=false;
            }
        else {
            answerError=true;
            }
        }
    while(answerError);

    pCollection.insertData(pCollection.getLastPos(), product);
    cout << "Producto guardado... " << endl << endl;
    }

void Menu::getProduct() {
    Product product;
    cout << "\tObtener producto" << endl
         << "codigo de barras del producto: ";
    getline(cin, product.barCode);

    try {
        product=pCollection.retrieve(pCollection.findData(product.barCode));

        cout << "Codigo de barras " << product.barCode << endl
             << "Nombre " << product.name << endl
             << "Peso " << product.weight << endl
             << "Fecha ingreso " << product.enterDate.getDay()
             << "-" << product.enterDate.getMonth()
             << "-" << product.enterDate.getYear() << endl
             << "Precio mayoreo " << product.wholesalePrice << endl
             << "Precio menudeo " << product.retailPrice << endl;

        if(product.stock>0) {
            product.stock--;
            pCollection.deleteData(pCollection.findData(product.barCode));
            pCollection.insertData(pCollection.getLastPos(), product);
            cout << "Existencias" << product.stock << endl;
            }
```

```
            else {
                cout << "Existencias" << product.stock << endl;
                cout << "SIN EXISTENCIAS" << endl << endl;
                }
            }
        catch(ProductsCollectionException ex) {
            cout << "ERROR, no se pudo insertar el producto" << endl;
            }
        cout << endl << endl;
        }


void Menu::addProductStock() {
        Product product;

        cout << "Ingresa el codigo de barras del producto";
        getline(cin, product.barCode);
        try {
            product=pCollection.retrieve(pCollection.findData(product.barCode));
            cout << "Ingrese las existencias ";
            cin >> product.stock;
            pCollection.deleteData(pCollection.findData(product.barCode));
            pCollection.insertData(pCollection.getLastPos(), product);

            }
        catch(ProductsCollectionException ex) {
            cout << "ERROR, no se pudo guardar las existencias" << endl;
            }
        cout << endl << endl;
        }
```

Product.h (registro)

```
#ifndef PRODUCT_H
#define PRODUCT_H
#include <date.h>

using namespace std;

struct Product {
        string name;
        string barCode;
        float weight;
        Date enterDate;
```

```
    float wholesalePrice;
    float retailPrice;
    int stock;
    };

#endif // PRODUCT_H
```

ProductsCollection.h

```
#ifndef PRODUCTSCOLLECTION_H
#define PRODUCTSCOLLECTION_H
#define ARRAY_SIZE 1024
#include "product.h"


class ProductsCollectionException : public std::exception {
    private:
        std::string msg;
    public:
        explicit ProductsCollectionException(const char* message) : msg(message) {}
        explicit ProductsCollectionException(const std::string& message) :msg(message)
{}
        virtual ~ProductsCollectionException() throw () {}
        virtual const char* what() const throw () {
            return msg.c_str();
            }
    };

class ProductsCollection {
    private:
        Product data[ARRAY_SIZE];
        int last;
        bool isValidPos(int);
    public:
        void initialize();

        bool isEmpty();
        bool isFull();

        void insertData(int, const Product&);
        void deleteData(int);

        int getFirstPos();
```

```
        int getLastPos();
        int getPrevPos(int);
        int getNextPos(int);
        int findData(const string&);

        Product retrieve(int);

        void sortData();

        void printData();

        void deleteAll();
    };

#endif // PRODUCTSCOLLECTION_H
```

ProductsCollection.cpp

```
#include "productsCollection.h"

bool ProductsCollection::isValidPos(int p)
{
    return p >= 0 and p <= last;
}

void ProductsCollection::initialize()
{
    last = -1;
}

bool ProductsCollection::isEmpty()
{
    return last == -1;
}

bool ProductsCollection::isFull()
{
    return last == ARRAY_SIZE - 1;
}

void ProductsCollection::insertData(int p, const Product& prod)
{
    if(isFull()){
```

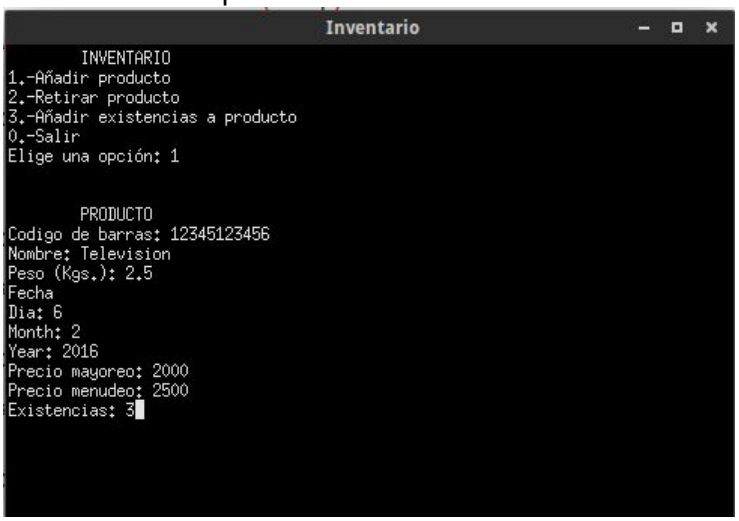
```
            throw ProductsCollectionException("The list is full, trying to insert");
        }
        if(p !=-1 and !isValidPos(p)){
            throw ProductsCollectionException("Invalid position, trying to insert");
        }
        for(int i = last; i > p; i--){
            data[i+1] = data[i];
        }

        data[p+1] = prod;

        last++;

}

void ProductsCollection::deleteData(int p)
{
        if(isEmpty()){
            throw ProductsCollectionException("DATA UNDERRUN. The list is empty, trying
to delete");
        }
        if(!isValidPos(p)){
            throw ProductsCollectionException("INVALID POSITION, trying ot delete");
        }
        for (int i=p; i < last; i++){
            data[i] = data[i + 1];
        }
        last--;
}

int ProductsCollection::getFirstPos()
{
        if(isEmpty()){
            return -1;
        }
        return 0;
}

int ProductsCollection::getLastPos()
{
        return last;
}
```

```
int ProductsCollection::getPrevPos(int p)
{
	if(isEmpty() or !isValidPos(p) or p==0){
		return -1;
	}
	else{
		return p-1;
	}
}

int ProductsCollection::getNextPos(int p)
{
	if(isEmpty() or !isValidPos(p) or p == last - 1){
		return -1;
	}
	return p-1;
}

int ProductsCollection::findData(const string& prod) {
	for(int i = 0; i <= last;    i++) {
		if(data[i].barCode == prod) {
			return i;
			}
		}
	return -1;
	}

Product ProductsCollection::retrieve(int p)
{
	if(isEmpty()){
		throw ProductsCollectionException("DATA UNDERRUN, list empty, trying to get
data");
	}
	if(!isValidPos(p)){
		throw ProductsCollectionException("INVALID POSITION, trying to get data");
	}
	return data[p];
}

void ProductsCollection::sortData()
{

}
```

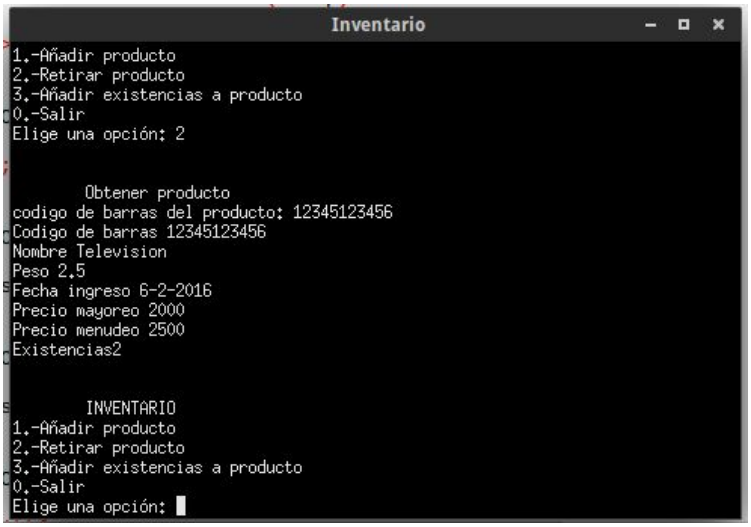
```
void ProductsCollection::printData()
{

}

void ProductsCollection::deleteAll()
{
	last=-1;
}
```

Pruebas

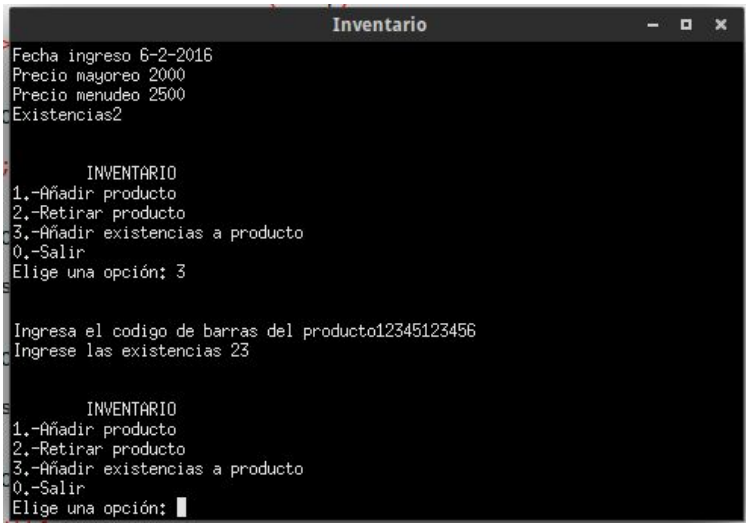
Añadiendo un producto



Sacandolo de inventario.



Añadiendo existencias



Retirando otro producto