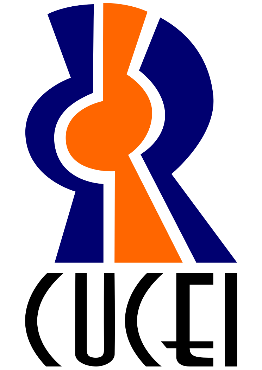
Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

Carrera: Ingeniería en Computación

**Estructuras de Datos**

NRC: 200219

Clave de la materia: IL354

Sección D01

Lunes, miércoles y viernes de 7 a 9

**Proyecto final.** Recetario digital

**Maestro:**

Alfredo Gutiérrez Hernández

**Datos personales:**

Alumno: Octavio Salvador Villegas Navarro

Código: 221977608

Fecha: 7 de mayo de 2023

|  |  |  |  |
| --- | --- | --- | --- |
| **Autoevaluación** | | | |
| **Concepto** | **Sí** | **No** | **Acumulación** |
| Bajé el trabajo de internet o alguien me lo pasó (aunque sea de forma parcial) | -100 pts | 0 pts | 0 |
| Incluí el código fuente **en formato de texto (sólo si funciona cumpliento todos los requerimientos)** | +25 pts | 0 pts | 25 |
| Incluí las **impresiones de pantalla (sólo si funciona cumpliento todos los requerimientos** | +25 pts | 0 pts | 50 |
| Incluí una **portada** que identifica mi trabajo (nombre, código, materia, fecha, título) | +25 pts | 0 pts | 75 |
| Incluí una **descripción y conclusiones** de mi trabajo | +25 pts | 0 pts | 100 |
| Suma: | | | 100 |

**Introducción**

*Notas para la entrega final:*

Para la entrega final se realizó una modificación sobre el proyecto, de tal forma que ahora el programa trabaja con listas ligadas. Se utilizaron dos tipos de listas, la lineal simplemente ligada y la circular doblemente ligada. Recordemos que en el programa se tiene composición de clases, por lo que se utilizó la lista simplemente ligada para el atributo ingredientes de la clase receta, mientras que la doblemente ligada se utilizó para crear la lista que representa el recetario.

*Original (entrega preliminar):*

Para el proyecto final se abordará el siguiente problema:

Un Chef necesita un programa en el cual pueda capturar las recetas de los platillos que ofrece a sus clientes. El programa deberá almacenar los nombres de las recetas con su respectiva lista de ingredientes, tiempo de preparación, procedimiento y nombre del autor de la receta.

Este programa consiste en implementar todo lo que hemos aprendido durante el curso, pues utilizaremos la lista para manejar el sistema que desea el chef. De esta forma, el chef podrá hacer uso de todos los métodos de la lista para poder manipular su recetario, pues a través del modelo base de la lista se manejará cada función del programa.

**Código Fuente**

#ifndef NAME\_H\_INCLUDED

#define NAME\_H\_INCLUDED

#include <string>

#include <iostream>

**class** Name{

**private**:

**std**::**string** first;

**std**::**string** last;

**public**:

Name();

Name(**const** Name&);

**std**::**string** getFirst() **const**;

**std**::**string** getLast() **const**;

**void** setFirst(**const std**::**string**&);

**void** setLast(**const std**::**string**&);

**std**::**string** toString() **const**;

Name& **operator** = (**const** Name&);

**bool operator** ==(**const** Name&) **const**;

**bool operator** !=(**const** Name&) **const**;

**bool operator** <(**const** Name&) **const**;

**bool operator** >(**const** Name&) **const**;

**bool operator** <=(**const** Name&) **const**;

**bool operator** >=(**const** Name&) **const**;

**static int** compareByFirst(**const** Name&, **const** Name&);

**static int** compareByLast(**const** Name&, **const** Name&);

**friend std**::ostream& **operator** << (**std**::ostream&, Name&);

**friend std**::**istream**& **operator** >> (**std**::**istream**&, Name&);

};

#endif // NAME\_H\_INCLUDED

#include "name.h"

**using namespace std**;

Name::Name(){}

Name::Name(**const** Name& n) : first(n.first), last(n.last){}

**string** Name::getFirst() **const**

{

**return** first;

}

**string** Name::getLast() **const**

{

**return** last;

}

**void** Name::setFirst(**const std**::**string**& s)

{

first = s;

}

**void** Name::setLast(**const std**::**string**& s)

{

last = s;

}

**string** Name::toString() **const**

{

**string** result;

result += first;

result += " ";

result += last;

**return** result;

}

Name& Name::**operator**=(**const** Name& n)

{

first = n.first;

last = n.last;

**return** \***this**;

}

**bool** Name::**operator**==(**const** Name& n) **const**

{

**return** \***this** == n;

}

**bool** Name::**operator**!=(**const** Name& n) **const**

{

**return** \***this** != n;

}

**bool** Name::**operator**<(**const** Name& n) **const**

{

**return** \***this** < n;

}

**bool** Name::**operator**>(**const** Name& n) **const**

{

**return** \***this** > n;

}

**bool** Name::**operator**<=(**const** Name& n) **const**

{

**return** (\***this** < n) or (\***this** == n);

}

**bool** Name::**operator**>=(**const** Name& n) **const**

{

**return** (\***this** > n) or (\***this** == n);

}

**int** Name::compareByFirst(**const** Name& n1, **const** Name& n2)

{

**int** result;

result = n1.first.compare(n2.first);

**return** result;

}

**int** Name::compareByLast(**const** Name& n1, **const** Name& n2)

{

**int** result;

result = n1.last.compare(n2.last);

**return** result;

}

ostream& **operator** << (ostream& os, Name& n)

{

os << n.first << **endl**;

os << n.last;

**return** os;

}

**istream**& **operator** >> (**istream**& is, Name& n)

{

getline(is, n.first);

getline(is, n.last);

**return** is;

}

#ifndef INGREDIENT\_H\_INCLUDED

#define INGREDIENT\_H\_INCLUDED

#include <string>

#include <iostream>

**class** Ingredient{

**private**:

**std**::**string** name;

**int** quantity;

**std**::**string** unit;

**public**:

Ingredient();

Ingredient(**const** Ingredient&);

**void** setName(**const std**::**string**&);

**void** setQuantity(**const int**&);

**void** setUnit(**const std**::**string**&);

**std**::**string** getName() **const**;

**int** getQuantity() **const**;

**std**::**string** getUnit() **const**;

**std**::**string** txtString() **const**;

**std**::**string** toString() **const**;

Ingredient& **operator** = (**const** Ingredient&);

**bool operator** == (**const** Ingredient&) **const**;

**bool operator** != (**const** Ingredient&) **const**;

**bool operator** < (**const** Ingredient&) **const**;

**bool operator** > (**const** Ingredient&) **const**;

**bool operator** <= (**const** Ingredient&) **const**;

**bool operator** >= (**const** Ingredient&) **const**;

**static int** compareByName(**const** Ingredient&, **const** Ingredient&);

**static int** compareByQuantity(**const** Ingredient&, **const** Ingredient&);

**static int** compareByUnit(**const** Ingredient&, **const** Ingredient&);

**friend std**::ostream& **operator** << (**std**::ostream&, Ingredient&);

**friend std**::**istream**& **operator** >> (**std**::**istream**&, Ingredient&);

};

#endif // INGREDIENT\_H\_INCLUDED

#include "ingredient.h"

**using namespace std**;

Ingredient::Ingredient(){}

Ingredient::Ingredient(**const** Ingredient& i) : name(i.name), quantity(i.quantity), unit(i.unit){}

**void** Ingredient::setName(**const std**::**string**& s)

{

name = s;

}

**void** Ingredient::setQuantity(**const int**& i)

{

quantity = i;

}

**void** Ingredient::setUnit(**const std**::**string**& s)

{

unit = s;

}

**string** Ingredient::getName() **const**

{

**return** name;

}

**int** Ingredient::getQuantity() **const**

{

**return** quantity;

}

**string** Ingredient::getUnit() **const**

{

**return** unit;

}

**string** Ingredient::txtString() **const**

{

**string** result;

result += to\_string(quantity);

result += "\n";

result += unit;

result += "\n";

result += name;

**return** result;

}

**string** Ingredient::toString() **const**

{

**string** result;

result += to\_string(quantity);

result += " ";

result += unit;

result += " de ";

result += name;

**return** result;

}

Ingredient& Ingredient::**operator**=(**const** Ingredient& i)

{

name = i.name;

quantity = i.quantity;

unit = i.unit;

**return** \***this**;

}

**bool** Ingredient::**operator**==(**const** Ingredient& i) **const**

{

**return** \***this** == i;

}

**bool** Ingredient::**operator**!=(**const** Ingredient& i) **const**

{

**return** \***this** != i;

}

**bool** Ingredient::**operator**<(**const** Ingredient& i) **const**

{

**return** \***this** < i;

}

**bool** Ingredient::**operator**>(**const** Ingredient& i) **const**

{

**return** \***this** > i;

}

**bool** Ingredient::**operator**<=(**const** Ingredient& i) **const**

{

**return** (\***this** < i) or (\***this** == i);

}

**bool** Ingredient::**operator**>=(**const** Ingredient& i) **const**

{

**return** (\***this** > i) or (\***this** == i);

}

**int** Ingredient::compareByName(**const** Ingredient& i1, **const** Ingredient& i2)

{

**int** result;

result = i1.name.compare(i2.name);

**return** result;

}

**int** Ingredient::compareByQuantity(**const** Ingredient& i1, **const** Ingredient& i2)

{

**return** i1.quantity - i2.quantity;

}

**int** Ingredient::compareByUnit(**const** Ingredient& i1, **const** Ingredient& i2)

{

**int** result;

result = i1.unit.compare(i2.unit);

**return** result;

}

ostream& **operator** << (ostream& os, Ingredient& i)

{

os << i.name << **endl**;

os << i.quantity << **endl**;

os << i.unit;

**return** os;

}

**istream**& **operator** >> (**istream**& is, Ingredient& i)

{

**string** myStr;

getline(is, myStr);

i.quantity = stoi(myStr);

getline(is, i.unit);

getline(is, i.name);

**return** is;

}

#ifndef SLIST\_H\_INCLUDED

#define SLIST\_H\_INCLUDED

#include <exception>

#include <string>

#include <iostream>

#include <fstream>

**template** <**class** T>

**class** SLList{

**private**:

**class** Node{

**private**:

T data;

Node\* next;

**public**:

Node();

Node(**const** T&);

T getData() **const**;

Node\* getNext() **const**;

**void** setData(**const** T&);

**void** setNext(Node\*);

};

Node\* anchor;

**void** copyAll(**const** SLList<T>&);

**bool** isValidPos(Node\*) **const**;

**public**:

**typedef** Node\* Position;

**class** Exception : **public std**::**exception**{

**private**:

**std**::**string** msg;

**public**:

**explicit** Exception(**const char**\* message) : msg(message){}

**explicit** Exception(**const std**::**string**& message) : msg(message){}

**virtual** ~Exception() **throw**(){}

**virtual const char**\* what() **const throw**(){

**return** msg.c\_str();

}

};

SLList();

SLList(**const** SLList<T>&);

~SLList();

**bool** isEmpty() **const**;

**void** insertData(Node\*, **const** T&);

**void** deleteData(Node\*);

Node\* getFirstPos() **const**;

Node\* getLastPos() **const**;

Node\* getPrevPos(Node\*) **const**;

Node\* getNextPos(Node\*) **const**;

Node\* findData(**const** T&, **int** cmp(**const** T&, **const** T&)) **const**;

T getElement(Node\*) **const**;

**std**::**string** txtString() **const**;

**std**::**string** toString() **const**;

**std**::**string** categoricToString(**const** T&, **int** cmp(**const** T&, **const** T&)) **const**;

**void** nullify();

SLList<T>& **operator** = (**const** SLList<T>&);

};

//Nodo

**template** <**class** T>

SLList<T>::Node::Node() : next(**nullptr**){}

**template** <**class** T>

SLList<T>::Node::Node(**const** T& e) : data(e), next(**nullptr**){}

**template** <**class** T>

T SLList<T>::Node::getData() **const**

{

**return** data;

}

**template** <**class** T>

**typename** SLList<T>::Node\* SLList<T>::Node::getNext() **const**

{

**return** next;

}

**template** <**class** T>

**void** SLList<T>::Node::setData(**const** T& e)

{

data = e;

}

**template** <**class** T>

**void** SLList<T>::Node::setNext(Node\* p)

{

next = p;

}

//Lista

**template** <**class** T>

**void** SLList<T>::copyAll(**const** SLList<T>& l)

{

Node\* aux(l.anchor);

Node\* last(**nullptr**);

Node\* newNode;

**while**(aux != **nullptr**){

newNode = **new** Node(aux->getData());

**if**(newNode == **nullptr**){

**throw** Exception("Memoria no disponible, no se pudo insertar el dato");

}

**if**(last == **nullptr**){

anchor = newNode;

}

**else**{

last->setNext(newNode);

}

last = newNode;

aux = aux->getNext();

}

}

**template** <**class** T>

**bool** SLList<T>::isValidPos(Node\* p) **const**

{

Node\* aux(anchor);

**while**(aux != **nullptr**){

**if**(aux==p){

**return true**;

}

aux = aux->getNext();

}

**return false**;

}

**template** <**class** T>

SLList<T>::SLList() : anchor(**nullptr**){}

**template** <**class** T>

SLList<T>::SLList(**const** SLList<T>& l) : anchor(**nullptr**)

{

copyAll(l);

}

**template** <**class** T>

SLList<T>::~SLList()

{

nullify();

}

**template** <**class** T>

**bool** SLList<T>::isEmpty() **const**

{

**return** anchor == **nullptr**;

}

**template** <**class** T>

**void** SLList<T>::insertData(Node\* p, **const** T& e)

{

**if**(p != **nullptr** and !isValidPos(p)){

**throw** Exception("Posicion Invalida, no se pudo insertar el dato");

}

Node\* aux(**new** Node(e));

**if**(aux == **nullptr**){

**throw** Exception("Memoria no disponible, no se pudo insertar el dato");

}

**if**(p == **nullptr**){

aux->setNext(anchor);

anchor = aux;

}

**else**{

aux->setNext(p->getNext());

p->setNext(aux);

}

}

**template** <**class** T>

**void** SLList<T>::deleteData(Node\* p)

{

**if**(!isValidPos(p)){

**throw** Exception("Posicion invalida, no se pudo eliminar el dato");

}

**if**(p == anchor){

anchor = anchor->getNext();

}

**else**{

getPrevPos(p)->setNext(p->getNext());

}

**delete** p;

}

**template** <**class** T>

**typename** SLList<T>::Node\* SLList<T>::getFirstPos() **const**

{

**return** anchor;

}

**template** <**class** T>

**typename** SLList<T>::Node\* SLList<T>::getLastPos() **const**

{

**if**(isEmpty()){

**return nullptr**;

}

Node\* aux(anchor);

**while**(aux->getNext() != **nullptr**){

aux = aux->getNext();

}

**return** aux;

}

**template** <**class** T>

**typename** SLList<T>::Node\* SLList<T>::getPrevPos(Node\* p) **const**

{

**if**(p == anchor){

**return nullptr**;

}

Node\* aux(anchor);

**while**(aux != **nullptr** and aux->getNext() != p){

aux = aux->getNext();

}

**return** aux;

}

**template** <**class** T>

**typename** SLList<T>::Node\* SLList<T>::getNextPos(Node\* p) **const**

{

**if**(!isValidPos(p)){

**return nullptr**;

}

**return** p->getNext();

}

**template** <**class** T>

**typename** SLList<T>::Node\* SLList<T>::findData(**const** T& e, **int** cmp(**const** T&, **const** T&)) **const**

{

Node\* aux(anchor);

**while**(aux != **nullptr** and cmp(aux->getData(), e) != 0){

aux = aux->getNext();

}

**return** aux;

}

**template** <**class** T>

T SLList<T>::getElement(Node\* p) **const**

{

**if**(!isValidPos(p)){

**throw** Exception("Posicion invalida, no se pudo obtener el elemento");

}

**return** p->getData();

}

**template** <**class** T>

**std**::**string** SLList<T>::txtString() **const**

{

**std**::**string** result;

Node\* aux(anchor);

**while**(aux != **nullptr**){

result += aux->getData().txtString();

**if**(aux->getNext() == **nullptr**){

result += " \n";

}

**else**{

result += "\n";

}

aux = aux->getNext();

}

**return** result;

}

**template** <**class** T>

**std**::**string** SLList<T>::toString() **const**

{

**std**::**string** result;

Node\* aux(anchor);

**while**(aux != **nullptr**){

result += aux->getData().toString() + "\n";

aux = aux->getNext();

}

**return** result;

}

**template** <**class** T>

**std**::**string** SLList<T>::categoricToString(**const** T& e, **int** cmp(**const** T&, **const** T&)) **const**

{

**std**::**string** result;

Node\* pos = anchor;

Node\* last = getLastPos();

**do**{

**if**(cmp(e, pos->getData()) == 0){

result += pos->getData().toString() + "\n";

}

pos = pos->getNext();

}**while**(pos != last);

**return** result;

}

**template** <**class** T>

**void** SLList<T>::nullify()

{

Node\* aux;

**while**(anchor != **nullptr**){

aux = anchor;

anchor = anchor->getNext();

**delete** aux;

}

}

**template** <**class** T>

SLList<T>& SLList<T>::**operator**=(**const** SLList<T>& l)

{

nullify();

copyAll(l);

**return** \***this**;

}

#endif // SLIST\_H\_INCLUDED

#ifndef RECIPE\_H\_INCLUDED

#define RECIPE\_H\_INCLUDED

#include <string>

#include <iostream>

#include "slist.h"

#include "ingredient.h"

#include "name.h"

**class** Recipe{

**private**:

**std**::**string** name;

SLList<Ingredient> ingredients;

**int** prepTime;

**std**::**string** procedure;

Name author;

**std**::**string** category;

**public**:

Recipe();

Recipe(**const** Recipe&);

**void** setName(**const std**::**string**&);

**void** setIngredients(**const** SLList<Ingredient>&);

**void** setPrepTime(**const int**&);

**void** setProcedure(**const std**::**string**&);

**void** setAuthor(**const** Name&);

**void** setCategory(**const std**::**string**&);

**std**::**string** getName() **const**;

SLList<Ingredient> getIngredients() **const**;

**int** getprepTime() **const**;

**std**::**string** getProcedure() **const**;

Name getAuthor() **const**;

**std**::**string** getCategory() **const**;

**std**::**string** toString() **const**;

**std**::**string** displayRecipe() **const**;

Recipe& **operator** = (**const** Recipe&);

**bool operator** == (**const** Recipe&) **const**;

**bool operator** != (**const** Recipe&) **const**;

**bool operator** < (**const** Recipe&) **const**;

**bool operator** >(**const** Recipe&) **const**;

**bool operator** <= (**const** Recipe&) **const**;

**bool operator** >= (**const** Recipe&) **const**;

**static int** compareByName(**const** Recipe&, **const** Recipe&);

**static int** compareByPrepTime(**const** Recipe&, **const** Recipe&);

**static int** compareByCategory(**const** Recipe&, **const** Recipe&);

**friend std**::ostream& **operator** << (**std**::ostream&, Recipe&);

**friend std**::**istream**& **operator** >> (**std**::**istream**&, Recipe&);

};

#endif // RECIPE\_H\_INCLUDED

#include "recipe.h"

**using namespace std**;

Recipe::Recipe(){}

Recipe::Recipe(**const** Recipe& r) : name(r.name), ingredients(r.ingredients), prepTime(r.prepTime), procedure(r.procedure), author(r.author), category(r.category){}

**void** Recipe::setName(**const std**::**string**& s)

{

name = s;

}

**void** Recipe::setIngredients(**const** SLList<Ingredient>& i)

{

ingredients = i;

}

**void** Recipe::setPrepTime(**const int**& t)

{

prepTime = t;

}

**void** Recipe::setProcedure(**const std**::**string**& s)

{

procedure = s;

}

**void** Recipe::setAuthor(**const** Name& n)

{

author = n;

}

**void** Recipe::setCategory(**const std**::**string**& s)

{

category = s;

}

**string** Recipe::getName() **const**

{

**return** name;

}

SLList<Ingredient> Recipe::getIngredients() **const**

{

**return** ingredients;

}

**int** Recipe::getprepTime() **const**

{

**return** prepTime;

}

**string** Recipe::getProcedure() **const**

{

**return** procedure;

}

Name Recipe::getAuthor() **const**

{

**return** author;

}

**string** Recipe::getCategory() **const**

{

**return** category;

}

**string** Recipe::toString() **const**

{

**string** result;

result += "Nombre: ";

result += name;

result += " | ";

result += "Categoria: ";

result += category;

result += " | ";

result += "Autor: ";

result += author.toString();

result += " | ";

result += "Tiempo de preparacion: ";

result += to\_string(prepTime);

result += " minutos";

**return** result;

}

**string** Recipe::displayRecipe() **const**

{

**string** result;

result += toString();

result += "\nIngredientes:\n";

result += ingredients.toString();

result += "\nProcedimiento:\n";

result += procedure;

**return** result;

}

Recipe& Recipe::**operator**=(**const** Recipe& r)

{

name = r.name;

ingredients = r.ingredients;

prepTime = r.prepTime;

procedure = r.procedure;

author = r.author;

category = r.category;

**return** \***this**;

}

**bool** Recipe::**operator**==(**const** Recipe& r) **const**

{

**return** \***this** == r;

}

**bool** Recipe::**operator**!=(**const** Recipe& r) **const**

{

**return** \***this** != r;

}

**bool** Recipe::**operator**<(**const** Recipe& r) **const**

{

**return** \***this** < r;

}

**bool** Recipe::**operator**>(**const** Recipe& r) **const**

{

**return** \***this** > r;

}

**bool** Recipe::**operator**<=(**const** Recipe& r) **const**

{

**return** (\***this** < r) or (\***this** == r);

}

**bool** Recipe::**operator**>=(**const** Recipe& r) **const**

{

**return** (\***this** > r) or (\***this** == r);

}

**int** Recipe::compareByName(**const** Recipe& r1, **const** Recipe& r2)

{

**int** result;

result = r1.name.compare(r2.name);

**return** result;

}

**int** Recipe::compareByPrepTime(**const** Recipe& r1, **const** Recipe& r2)

{

**return** r1.prepTime - r2.prepTime;

}

**int** Recipe::compareByCategory(**const** Recipe& r1, **const** Recipe& r2)

{

**int** result;

result = r1.category.compare(r2.category);

**return** result;

}

ostream& **operator** << (ostream& os, Recipe& r)

{

os << r.name << **endl**;

os << r.category << **endl**;

os << r.author << **endl**;

os << r.prepTime << **endl**;

os << r.ingredients.txtString();

os << r.procedure;

**return** os;

}

**istream**& **operator** >> (**istream**& is, Recipe& r)

{

**string** auxNameIngredient, myStr;

Ingredient auxIngredient;

**int** entero;

**char** last;

r.ingredients.nullify();

getline(is, r.name);

**if**(r.name.**empty**()){

**return** is;

}

getline(is, r.category);

is >> r.author;

getline(is, myStr);

entero = stoi(myStr);

r.setPrepTime(entero);

**do**{

is >> auxIngredient;

r.ingredients.insertData(r.ingredients.getLastPos(), auxIngredient);

myStr = auxIngredient.getName();

last = myStr[myStr.length()-1];

}**while**(last != ' ');

getline(is, r.procedure);

**return** is;

}

#ifndef DLIST\_H\_INCLUDED

#define DLIST\_H\_INCLUDED

#include <exception>

#include <string>

#include <iostream>

#include <fstream>

**template** <**class** T>

**class** DLList{

**private**:

**class** Node{

**private**:

T\* dataPtr;

Node\* prev;

Node\* next;

**public**:

**class** Exception : **public std**::**exception**{

**private**:

**std**::**string** msg;

**public**:

**explicit** Exception(**const char**\* message) : msg(message){}

**explicit** Exception(**const std**::**string**& message) : msg(message){}

**virtual** ~Exception() **throw**(){}

**virtual const char**\* what() **const throw**(){

**return** msg.c\_str();

}

};

Node();

Node(**const** T&);

~Node();

T\* getDataPtr() **const**;

T getData() **const**;

Node\* getPrev() **const**;

Node\* getNext() **const**;

**void** setDataPtr(T\*);

**void** setData(**const** T&);

**void** setPrev(Node\*);

**void** setNext(Node\*);

};

Node\* header;

**void** copyAll(**const** DLList<T>&);

**bool** isValidPos(Node\*) **const**;

**void** swapData(T\*, T\*);

Node\* sortingPartition(Node\* l, Node\* h, **int** cmp(**const** T&, **const** T&));

**void** sortList(Node\*, Node\*, **int** cmp(**const** T&, **const** T&));

**public**:

**typedef** Node\* Position;

**class** Exception : **public std**::**exception**{

**private**:

**std**::**string** msg;

**public**:

**explicit** Exception(**const char**\* message) : msg(message){}

**explicit** Exception(**const std**::**string**& message) : msg(message){}

**virtual** ~Exception() **throw**(){}

**virtual const char**\* what() **const throw**(){

**return** msg.c\_str();

}

};

DLList();

DLList(**const** DLList<T>&);

~DLList();

**void** sortList(**int** (\*cmp)(**const** T&, **const** T&));

**bool** isEmpty() **const**;

**void** insertData(Node\*, **const** T&);

**void** deleteData(Node\*);

Node\* getFirstPos() **const**;

Node\* getLastPos() **const**;

Node\* getPrevPos(Node\*) **const**;

Node\* getNextPos(Node\*) **const**;

Node\* findData(**const** T&, **int** cmp(**const** T&, **const** T&)) **const**;

T\* getElement(Node\*) **const**;

**std**::**string** toString() **const**;

**std**::**string** categoricToString(**const** T&, **int** cmp(**const** T&, **const** T&)) **const**;

**void** nullify();

DLList<T>& **operator** = (**const** DLList<T>&);

**void** readFromDisk(**const std**::**string**&);

**void** writeToDisk(**const std**::**string**&);

};

//Nodo

**template** <**class** T>

DLList<T>::Node::Node() : dataPtr(**nullptr**), prev(**nullptr**), next(**nullptr**){}

**template** <**class** T>

DLList<T>::Node::Node(**const** T& e) : dataPtr(**new** T(e)), prev(**nullptr**), next(**nullptr**)

{

**if** (dataPtr == **nullptr**){

**throw** Exception("Memoria insuficiente, no se pudo crear nuevo nodo");

}

}

**template** <**class** T>

DLList<T>::Node::~Node()

{

**delete** dataPtr;

}

**template** <**class** T>

T\* DLList<T>::Node::getDataPtr() **const**

{

**return** dataPtr;

}

**template** <**class** T>

T DLList<T>::Node::getData() **const**

{

**if** (dataPtr == **nullptr**){

**throw** Exception("Dato inexistente, getData");

}

**return** \*dataPtr;

}

**template** <**class** T>

**typename** DLList<T>::Node\* DLList<T>::Node::getPrev() **const**

{

**return** prev;

}

**template** <**class** T>

**typename** DLList<T>::Node\* DLList<T>::Node::getNext() **const**

{

**return** next;

}

**template** <**class** T>

**void** DLList<T>::Node::setDataPtr(T\* p)

{

dataPtr = p;

}

**template** <**class** T>

**void** DLList<T>::Node::setData(**const** T& e)

{

**if**(dataPtr == **nullptr**){

**if**((dataPtr = **new** T(e)) == **nullptr**){

**throw** Exception("Memoria no disponible, setData");

}

}

**else**{

\*dataPtr = e;

}

}

**template** <**class** T>

**void** DLList<T>::Node::setPrev(Node\* p)

{

prev = p;

}

**template** <**class** T>

**void** DLList<T>::Node::setNext(Node\* p)

{

next = p;

}

//Lista

**template** <**class** T>

**void** DLList<T>::copyAll(**const** DLList<T>& l)

{

Node\* aux(l.header->getNext());

Node\* newNode;

**while**(aux != l.header){

**try**{

**if**((newNode = **new** Node(aux->getData())) == **nullptr**){

**throw** Exception("Memoria no disponible, no se pudo copiar la lista");

}

} **catch** (**typename** Node::Exception ex){

**throw** Exception(ex.what());

}

newNode->setPrev(header->getPrev());

newNode->setNext(header);

header->getPrev()->setNext(newNode);

header->setPrev(newNode);

aux = aux->getNext();

}

}

**template** <**class** T>

**bool** DLList<T>::isValidPos(Node\* p) **const**

{

Node\* aux(header->getNext());

**while** (aux != header){

**if**(aux == p){

**return true**;

}

aux = aux->getNext();

}

**return false**;

}

**template** <**class** T>

DLList<T>::DLList() : header(**new** Node)

{

**if**(header == **nullptr**){

**throw** Exception("Memoria no disponible, no se pudo inicializar la lista");

}

header->setPrev(header);

header->setNext(header);

}

**template** <**class** T>

DLList<T>::DLList(**const** DLList<T>& l) : DLList()

{

copyAll(l);

}

**template** <**class** T>

DLList<T>::~DLList()

{

nullify();

**delete** header;

}

**template** <**class** T>

**bool** DLList<T>::isEmpty() **const**

{

**return** header->getNext() == header;

}

**template** <**class** T>

**void** DLList<T>::insertData(Node\* p, **const** T& e)

{

**if** (p != **nullptr** and !isValidPos(p)){

**throw** Exception("Posicion invalida, no se pudo insertar el dato");

}

Node\* aux;

**try**{

aux = **new** Node(e);

}

**catch** (**typename** Node::Exception ex){

**throw** Exception(ex.what());

}

**if** (aux == **nullptr**){

**throw** Exception("Memoria no disponible, no se pudo insertar el dato");

}

**if**(p == **nullptr**){

p = header;

}

aux->setPrev(p);

aux->setNext(p->getNext());

p->getNext()->setPrev(aux);

p->setNext(aux);

}

**template** <**class** T>

**void** DLList<T>::deleteData(Node\* p)

{

**if**(!isValidPos(p)){

**throw** Exception("Posicion invalida, no se pudo borrar el dato");

}

p->getPrev()->setNext(p->getNext());

p->getNext()->setPrev(p->getPrev());

**delete** p;

}

**template** <**class** T>

**typename** DLList<T>::Node\* DLList<T>::getFirstPos() **const**

{

**if**(isEmpty()){

**return nullptr**;

}

**return** header->getNext();

}

**template** <**class** T>

**typename** DLList<T>::Node\* DLList<T>::getLastPos() **const**

{

**if**(isEmpty()){

**return nullptr**;

}

**return** header->getPrev();

}

**template** <**class** T>

**typename** DLList<T>::Node\* DLList<T>::getPrevPos(Node\* p) **const**

{

**if**(!isValidPos(p) or p==header->getNext()){

**return nullptr**;

}

**return** p->getPrev();

}

**template** <**class** T>

**typename** DLList<T>::Node\* DLList<T>::getNextPos(Node\* p) **const**

{

**if**(!isValidPos(p) or p==header->getPrev()){

**return nullptr**;

}

**return** p->getNext();

}

**template** <**class** T>

**typename** DLList<T>::Node\* DLList<T>::findData(**const** T& e, **int** cmp(**const** T&, **const** T&)) **const**

{

Node\* aux(header->getNext());

**while**(aux != header){

**if**(cmp(aux->getData(), e) == 0){

**return** aux;

}

aux = aux->getNext();

}

**return nullptr**;

}

**template** <**class** T>

T\* DLList<T>::getElement(Node\* p) **const**

{

**if**(!isValidPos(p)){

**throw** Exception("Posicion invalida, no se pudo recuperar el elemento");

}

**return** p->getDataPtr();

}

**template** <**class** T>

**std**::**string** DLList<T>::toString() **const**

{

Node\* aux(header->getNext());

**std**::**string** result;

**while**(aux != header){

result += aux->getData().toString() + "\n";

aux = aux->getNext();

}

**return** result;

}

**template** <**class** T>

**std**::**string** DLList<T>::categoricToString(**const** T& e, **int** cmp(**const** T&, **const** T&)) **const**

{

**std**::**string** result;

Node\* aux(header->getNext());

**while**(aux != header){

**if**(cmp(e, aux->getData()) == 0){

result += aux->getData().toString() + "\n";

}

aux = aux->getNext();

}

**return** result;

}

**template** <**class** T>

**void** DLList<T>::nullify()

{

Node\* aux;

**while**(header->getNext() != header){

aux = header->getNext();

header->setNext(aux->getNext());

**delete** aux;

}

header->setPrev(header);

}

**template** <**class** T>

DLList<T>& DLList<T>::**operator**=(**const** DLList<T>& l)

{

nullify();

copyAll(l);

**return** \***this**;

}

**template** <**class** T>

**void** DLList<T>::swapData(T\* a, T\* b)

{

T aux = \*a;

\*a = \*b;

\*b = aux;

}

**template** <**class** T>

**typename** DLList<T>::Node\* DLList<T>::sortingPartition(Node\* l, Node\* h, **int** cmp(**const** T&, **const** T&))

{

T x = h->getData();

Node\* i = l->getPrev();

**for**(Node\* j=l; j!=h; j=j->getNext()){

**if**(cmp(j->getData(), x) <= 0){

i = (i == header)? l : i->getNext();

swapData(i->getDataPtr(), j->getDataPtr());

}

}

i = (i==header)? l : i->getNext();

swapData(i->getDataPtr(), h->getDataPtr());

**return** i;

}

**template** <**class** T>

**void** DLList<T>::sortList(**int** (\*cmp)(**const** T&, **const** T&))

{

Node\* h = getLastPos();

Node\* first = getFirstPos();

sortList(first, h, cmp);

}

**template** <**class** T>

**void** DLList<T>::sortList(Node\* l, Node\* h, **int** cmp(**const** T&, **const** T&))

{

**if**(h!=header and l!=h and l!=h->getNext()){

Node\* p = sortingPartition(l, h, cmp);

sortList(l, p->getPrev(), cmp);

sortList(p->getNext(), h, cmp);

}

}

**template** <**class** T>

**void** DLList<T>::readFromDisk(**const std**::**string**& fileName)

{

**std**::**ifstream** myFile;

myFile.open(fileName);

**if**(!myFile.is\_open()){

**throw** Exception("No se pudo abrir el archivo para lectura");

}

nullify();

T myData;

**try**{

**while**(myFile >> myData){

insertData(getLastPos(), myData);

}

}

**catch**(Exception ex){

myFile.close();

**throw** Exception(ex.what());

}

myFile.close();

}

**template** <**class** T>

**void** DLList<T>::writeToDisk(**const std**::**string**& fileName)

{

**std**::**ofstream** myFile;

myFile.open(fileName, **std**::**ios\_base**::trunc);

**if**(!myFile.is\_open()){

**throw** Exception("No se pudo abrir el archivo para escritura");

}

Node\* aux = header->getNext();

T data;

**while**(aux != header){

data = aux->getData();

myFile << data << **std**::**endl**;

aux = aux->getNext();

}

myFile.close();

}

#endif // DLIST\_H\_INCLUDED

#ifndef MENU\_H\_INCLUDED

#define MENU\_H\_INCLUDED

#include <iostream>

#include "recipe.h"

#include "dlist.h"

**class** Menu{

**private**:

DLList<Recipe> recipeList;

**public**:

Menu();

**void** start();

**void** displayList();

**void** addRecipe();

**void** displayRecipe();

**void** removeRecipe();

**void** deleteList();

**void** sortList();

**void** modifyIngredients();

**void** modifyProcedure();

SLList<Ingredient> addIngredients() **const**;

SLList<Ingredient> addIngredients(**const** SLList<Ingredient>&) **const**;

};

#endif // MENU\_H\_INCLUDED

#include "menu.h"

**using namespace std**;

Menu::Menu(){}

**void** Menu::start()

{

**char** pick;

**cout** << "Bienvenido al programa!" << **endl** << **endl**;

**do**{

**cout** << "\nSeleccione una opcion:\n";

**cout** << "[a] Mostrar lista." << **endl**;

**cout** << "[b] Insertar receta." << **endl**;

**cout** << "[c] Mostrar receta." << **endl**;

**cout** << "[d] Borrar una receta." << **endl**;

**cout** << "[e] Ordenar lista." << **endl**;

**cout** << "[f] Modificar ingredientes de una receta." << **endl**;

**cout** << "[g] Modificar procedimiento de una receta." << **endl**;

**cout** << "[h] Borrar recetario." << **endl**;

**cout** << "[i] Leer recetario del disco." << **endl**;

**cout** << "[j] Escribir recetario al disco." << **endl**;

**cout** << "[s] Salir\n-> ";

**cin** >> pick;

**cin**.ignore();

**switch**(pick){

**case** 'a': displayList(); **break**;

**case** 'b': addRecipe(); **break**;

**case** 'c': displayRecipe(); **break**;

**case** 'd': removeRecipe(); **break**;

**case** 'e': sortList(); **break**;

**case** 'f': modifyIngredients(); **break**;

**case** 'g': modifyProcedure(); **break**;

**case** 'h': recipeList.nullify(); **cout** << "\nLista borrada" << **endl**; **break**;

**case** 'i': recipeList.readFromDisk("listado.txt"); **cout** << "\nRecetario leido" << **endl**; **break**;

**case** 'j': recipeList.writeToDisk("listado.txt"); **cout** << "\nRecetario escrito" << **endl**; **break**;

**case** 's': **cout** << "\nHasta pronto!"; getchar(); **break**;

}

} **while**(pick != 's');

}

**void** Menu::displayList()

{

**char** pick;

Recipe myRecipe;

**if**(recipeList.isEmpty()){

**cout** << "La lista se encuentra vacia" << **endl** << **endl**;

**return**;

}

**cout** << "\nDesplegar:\n[a]Toda la lista\n[b]Desayunos\n[c]Comidas\n[d]Cenas\n[e]Navidenos\n->";

**cin** >> pick;

**cin**.ignore();

**cout** << "Lista actual:" << **endl**;

**switch**(pick){

**case** 'a':

**cout** << recipeList.toString() << **endl** << **endl**;

**break**;

**case** 'b':

myRecipe.setCategory("Desayuno");

**cout** << recipeList.categoricToString(myRecipe, Recipe::compareByCategory) << **endl** << **endl**;

**break**;

**case** 'c':

myRecipe.setCategory("Comida");

**cout** << recipeList.categoricToString(myRecipe, Recipe::compareByCategory) << **endl** << **endl**;

**break**;

**case** 'd':

myRecipe.setCategory("Cena");

**cout** << recipeList.categoricToString(myRecipe, Recipe::compareByCategory) << **endl** << **endl**;

**break**;

**case** 'e':

myRecipe.setCategory("Navideno");

**cout** << recipeList.categoricToString(myRecipe, Recipe::compareByCategory) << **endl** << **endl**;

**break**;

}

}

**void** Menu::addRecipe()

{

Recipe myRecipe;

Name myName;

**string** myStr;

**char** pick;

DLList<Recipe>::Position pos = recipeList.getLastPos();

**cout** << "Nombre del platillo: ";

getline(**cin**, myStr);

myRecipe.setName(myStr);

**cout** << "\nCategoria:\n[a]Desayuno\n[b]Comida\n[c]Cena\n[d]Navideno\n-> ";

**cin** >> pick;

**cin**.ignore();

**switch**(pick){

**case** 'a': myStr = "Desayuno"; **break**;

**case** 'b': myStr = "Comida"; **break**;

**case** 'c': myStr = "Cena"; **break**;

**case** 'd': myStr = "Navideno"; **break**;

**default**: myStr = "s/c"; **break**;

}

myRecipe.setCategory(myStr);

**cout** << "Tiempo de preparacion (en minutos): ";

getline(**cin**, myStr);

myRecipe.setPrepTime(stoi(myStr));

**cout** << "Nombre del autor: ";

getline(**cin**, myStr);

myName.setFirst(myStr);

**cout** << "Apellido del autor: ";

getline(**cin**, myStr);

myName.setLast(myStr);

myRecipe.setAuthor(myName);

myRecipe.setIngredients(addIngredients());

**cout** << "Procedimiento: ";

getline(**cin**, myStr);

myRecipe.setProcedure(myStr);

**try**{

recipeList.insertData(pos, myRecipe);

}**catch**(**typename** DLList<Recipe>::Exception ex){

**cout** << "Error al ingresar receta: " << ex.what() << **endl**;

}

}

**void** Menu::displayRecipe()

{

Recipe myRecipe;

**string** myStr;

DLList<Recipe>::Position pos;

**char** pick;

**cout** << "\nSeleccione el criterio de busqueda:\n[a]Nombre de la receta\n[b]Categoria de la receta\n-> ";

**cin** >> pick;

**cin**.ignore();

**switch**(pick){

**case** 'a':

**cout** << "Nombre de la receta: ";

getline(**cin**, myStr);

myRecipe.setName(myStr);

pos = recipeList.findData(myRecipe, Recipe::compareByName);

**break**;

**case** 'b':

**cout** << "\nCategoria:\n[a]Desayuno\n[b]Comida\n[c]Cena\n[d]Navideno\n-> ";

**cin** >> pick;

**cin**.ignore();

**switch**(pick){

**case** 'a': myStr = "Desayuno"; **break**;

**case** 'b': myStr = "Comida"; **break**;

**case** 'c': myStr = "Cena"; **break**;

**case** 'd': myStr = "Navideno"; **break**;

**default**: myStr = "s/c"; **break**;

}

myRecipe.setCategory(myStr);

pos = recipeList.findData(myRecipe, Recipe::compareByCategory);

**break**;

}

**cout** << "\nLa busqueda de " << myStr << " ";

**if**(pos == **nullptr**){

**cout** << "no fue encontrada en la lista";

}

**else**{

**cout** << "\nSe muestra a continuacion: \n" << recipeList.getElement(pos)->displayRecipe() << **endl** << **endl**;

}

}

**void** Menu::removeRecipe()

{

Recipe myRecipe;

**string** myStr;

DLList<Recipe>::Position pos;

**cout** << "Nombre de la receta: ";

getline(**cin**, myStr);

myRecipe.setName(myStr);

pos = recipeList.findData(myRecipe, Recipe::compareByName);

**try**{

recipeList.deleteData(pos);

}**catch**(**typename** DLList<Recipe>::Exception ex){

**cout** << "Error al borrar receta: " << ex.what() << **endl**;

}

}

**void** Menu::deleteList()

{

recipeList.nullify();

**cout** << "Lista anulada" << **endl**;

}

**void** Menu::sortList()

{

**char** pick;

**cout** << "Seleccione el criterio de ordenamiento:\n[a]Nombre\n[b]Tiempo de preparacion\n->";

**cin** >> pick;

**cin**.ignore();

**if**(pick == 'b'){

recipeList.sortList(&Recipe::compareByPrepTime);

}**else**{

recipeList.sortList(&Recipe::compareByName);

}

}

**void** Menu::modifyIngredients()

{

Recipe myRecipe;

Ingredient myIngredient;

**string** myStr;

DLList<Recipe>::Position pos;

SLList<Ingredient>::Position internalPos;

**char** pick;

**cout** << "\nIngrese el nombre de la receta cuyos ingredientes desea modificar: ";

getline(**cin**, myStr);

myRecipe.setName(myStr);

pos = recipeList.findData(myRecipe, Recipe::compareByName);

**if**(pos == **nullptr**){

**cout** << "No se encontro la receta buscada" << **endl**;

**return**;

}

myRecipe = \*(recipeList.getElement(pos));

SLList<Ingredient> myIngredients = myRecipe.getIngredients();

**cout** << myRecipe.displayRecipe() << **endl**;

**cout** << "\nSeleccione una opcion:\n[a]Agregar un ingrediente\n[b]Eliminar un ingrediente\n[c]Modificar la cantidad de un ingrediente\n[d]Borrar todos los ingredientes\n->";

**cin** >> pick;

**cin**.ignore();

**switch**(pick){

**case** 'a':

myIngredients = addIngredients(myRecipe.getIngredients());

**break**;

**case** 'b':

**cout** << "\nIngrese el nombre del ingrediente que desea borrar: ";

getline(**cin**, myStr);

myIngredient.setName(myStr);

internalPos = myIngredients.findData(myIngredient, Ingredient::compareByName);

**try**{

myIngredients.deleteData(internalPos);

}**catch**(**typename** SLList<Recipe>::Exception ex){

**cout** << "Error al borrar receta: " << ex.what() << **endl**;

}

**break**;

**case** 'c':

**cout** << "\nIngrese el nombre del ingrediente que desea modificar: ";

getline(**cin**, myStr);

myIngredient.setName(myStr);

**cout** << "Nuevo nombre de la unidad: ";

getline(**cin**, myStr);

myIngredient.setUnit(myStr);

**cout** << "Nueva cantidad: ";

getline(**cin**, myStr);

myIngredient.setQuantity(stoi(myStr));

internalPos = myIngredients.findData(myIngredient, Ingredient::compareByName);

myIngredients.insertData(internalPos, myIngredient);

myIngredients.deleteData(internalPos);

**break**;

**case** 'd':

myIngredients.nullify();

**break**;

**default**:

**return**;

}

myRecipe.setIngredients(myIngredients);

\*(recipeList.getElement(pos)) = myRecipe;

}

**void** Menu::modifyProcedure()

{

Recipe myRecipe;

**string** myStr;

DLList<Recipe>::Position pos;

**cout** << "\nIngrese el nombre de la receta cuyo procedimiento desea modificar: ";

getline(**cin**, myStr);

myRecipe.setName(myStr);

pos = recipeList.findData(myRecipe, Recipe::compareByName);

**if**(pos == **nullptr**){

**cout** << "No se encontro la receta buscada" << **endl**;

**return**;

}

myRecipe = \*(recipeList.getElement(pos));

**cout** << "Nuevo procedimiento: ";

getline(**cin**, myStr);

myRecipe.setProcedure(myStr);

\*(recipeList.getElement(pos)) = myRecipe;

}

SLList<Ingredient> Menu::addIngredients() **const**

{

SLList<Ingredient> result;

**string** myStr;

**char** pick;

**bool** flag;

SLList<Ingredient>::Position pos;

**int** comparison;

**cout** << "A continuacion ingrese los ingredientes" << **endl**;

**do**{

Ingredient myIngredient;

**cout** << "Nombre del ingrediente: ";

getline(**cin**, myStr);

myIngredient.setName(myStr);

**cout** << "Nombre de la unidad: ";

getline(**cin**, myStr);

myIngredient.setUnit(myStr);

**cout** << "Cantidad: ";

getline(**cin**, myStr);

myIngredient.setQuantity(stoi(myStr));

pos = result.getFirstPos();

**if**(pos != **nullptr**){

**while**(pos != **nullptr**){

comparison = Ingredient::compareByName(myIngredient, result.getElement(pos));

**if**(comparison < 0){

**try**{

result.insertData(result.getPrevPos(pos), myIngredient);

}**catch**(**typename** SLList<Ingredient>::Exception ex){

**cout** << "Error al ingresar ingrediente: " << ex.what() << **endl**;

}

**break**;

}

**if**(result.getNextPos(pos) == **nullptr**){

**try**{

result.insertData(pos, myIngredient);

}**catch**(**typename** SLList<Ingredient>::Exception ex){

**cout** << "Error al ingresar ingrediente: " << ex.what() << **endl**;

}

**break**;

}

pos = result.getNextPos(pos);

}

}

**else**{

**try**{

result.insertData(pos, myIngredient);

}**catch**(**typename** SLList<Ingredient>::Exception ex){

**cout** << "Error al ingresar ingrediente: " << ex.what() << **endl**;

}

}

**cout** << "Ingresar otro ingrediente?(s/n): ";

**cin** >> pick;

**cin**.ignore();

**switch**(pick){

**case** 's': flag = **true**; **break**;

**case** 'n': flag = **false**; **break**;

}

}**while**(flag);

**return** result;

}

SLList<Ingredient> Menu::addIngredients(**const** SLList<Ingredient>& current) **const**

{

SLList<Ingredient> result = current;

**string** myStr;

**char** pick;

**bool** flag;

SLList<Ingredient>::Position pos;

**int** comparison;

**cout** << "A continuacion ingrese los ingredientes" << **endl**;

**do**{

Ingredient myIngredient;

**cout** << "Nombre del ingrediente: ";

getline(**cin**, myStr);

myIngredient.setName(myStr);

**cout** << "Nombre de la unidad: ";

getline(**cin**, myStr);

myIngredient.setUnit(myStr);

**cout** << "Cantidad: ";

getline(**cin**, myStr);

myIngredient.setQuantity(stoi(myStr));

pos = result.getFirstPos();

**if**(pos != **nullptr**){

**while**(pos != **nullptr**){

comparison = Ingredient::compareByName(myIngredient, result.getElement(pos));

**if**(comparison < 0){

**try**{

result.insertData(result.getPrevPos(pos), myIngredient);

}**catch**(**typename** SLList<Ingredient>::Exception ex){

**cout** << "Error al ingresar ingrediente: " << ex.what() << **endl**;

}

**break**;

}

**if**(result.getNextPos(pos) == **nullptr**){

**try**{

result.insertData(pos, myIngredient);

}**catch**(**typename** SLList<Ingredient>::Exception ex){

**cout** << "Error al ingresar ingrediente: " << ex.what() << **endl**;

}

**break**;

}

pos = result.getNextPos(pos);

}

}

**else**{

**try**{

result.insertData(pos, myIngredient);

}**catch**(**typename** SLList<Ingredient>::Exception ex){

**cout** << "Error al ingresar ingrediente: " << ex.what() << **endl**;

}

}

**cout** << "Ingresar otro ingrediente?(s/n): ";

**cin** >> pick;

**cin**.ignore();

**switch**(pick){

**case** 's': flag = **true**; **break**;

**case** 'n': flag = **false**; **break**;

}

}**while**(flag);

**return** result;

}

#include <iostream>

#include "menu.h"

**using namespace std**;

**int** main()

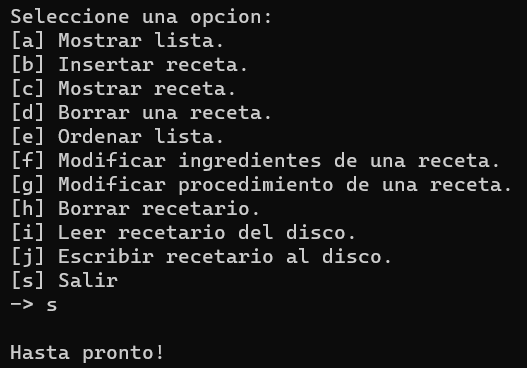
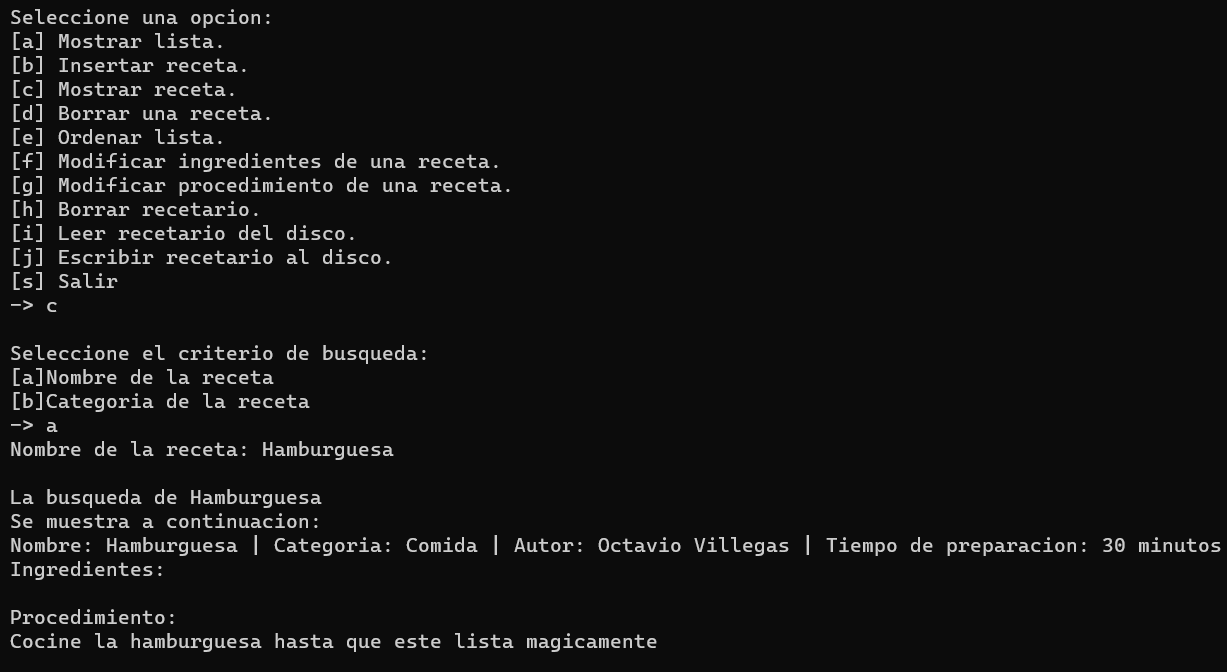
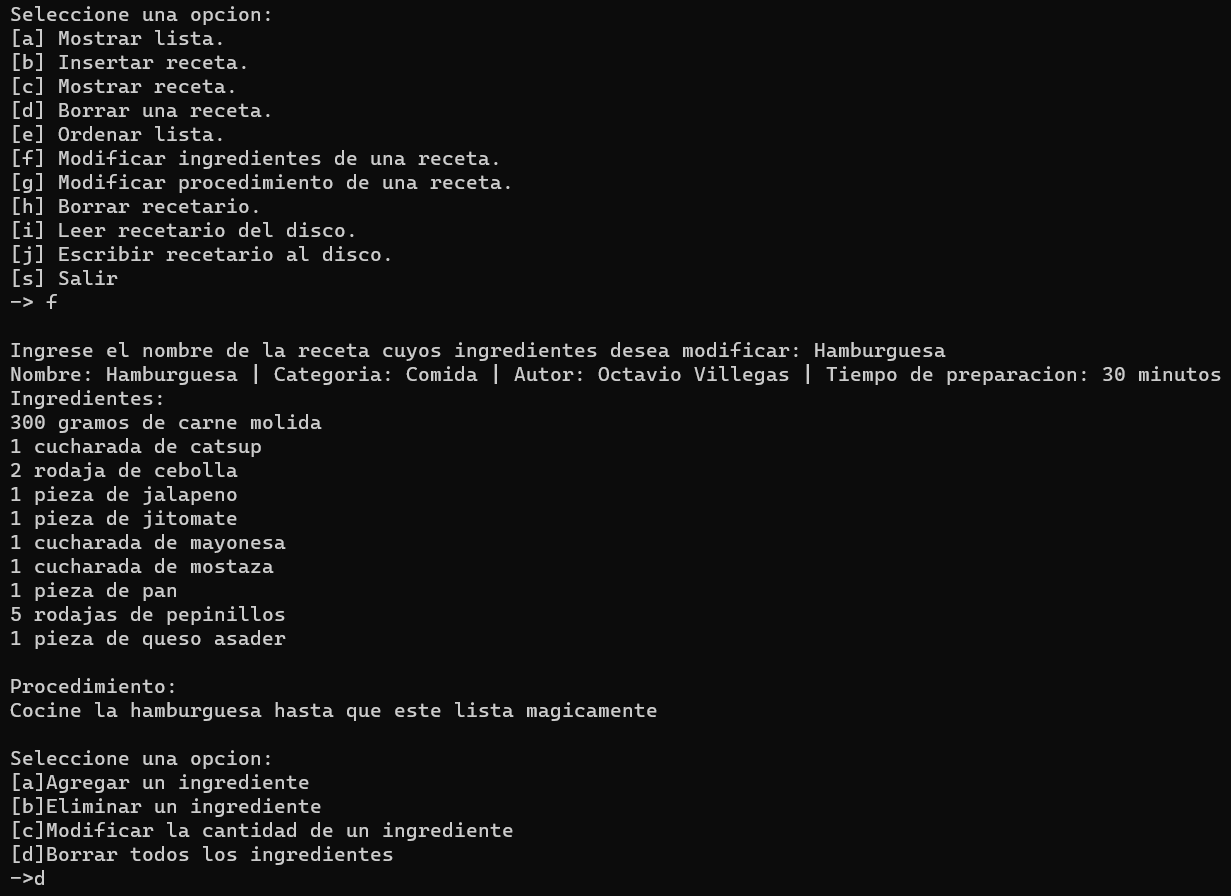
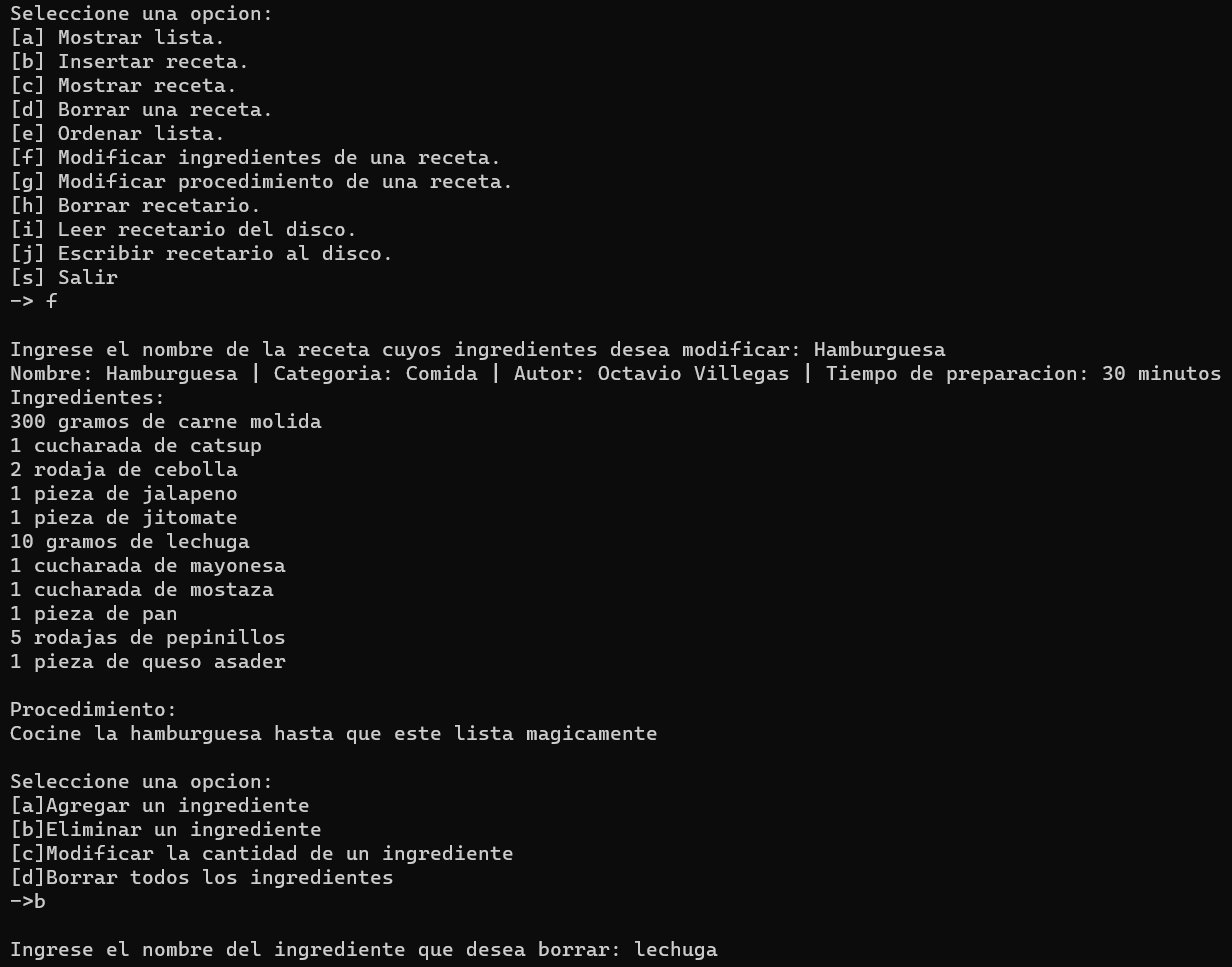
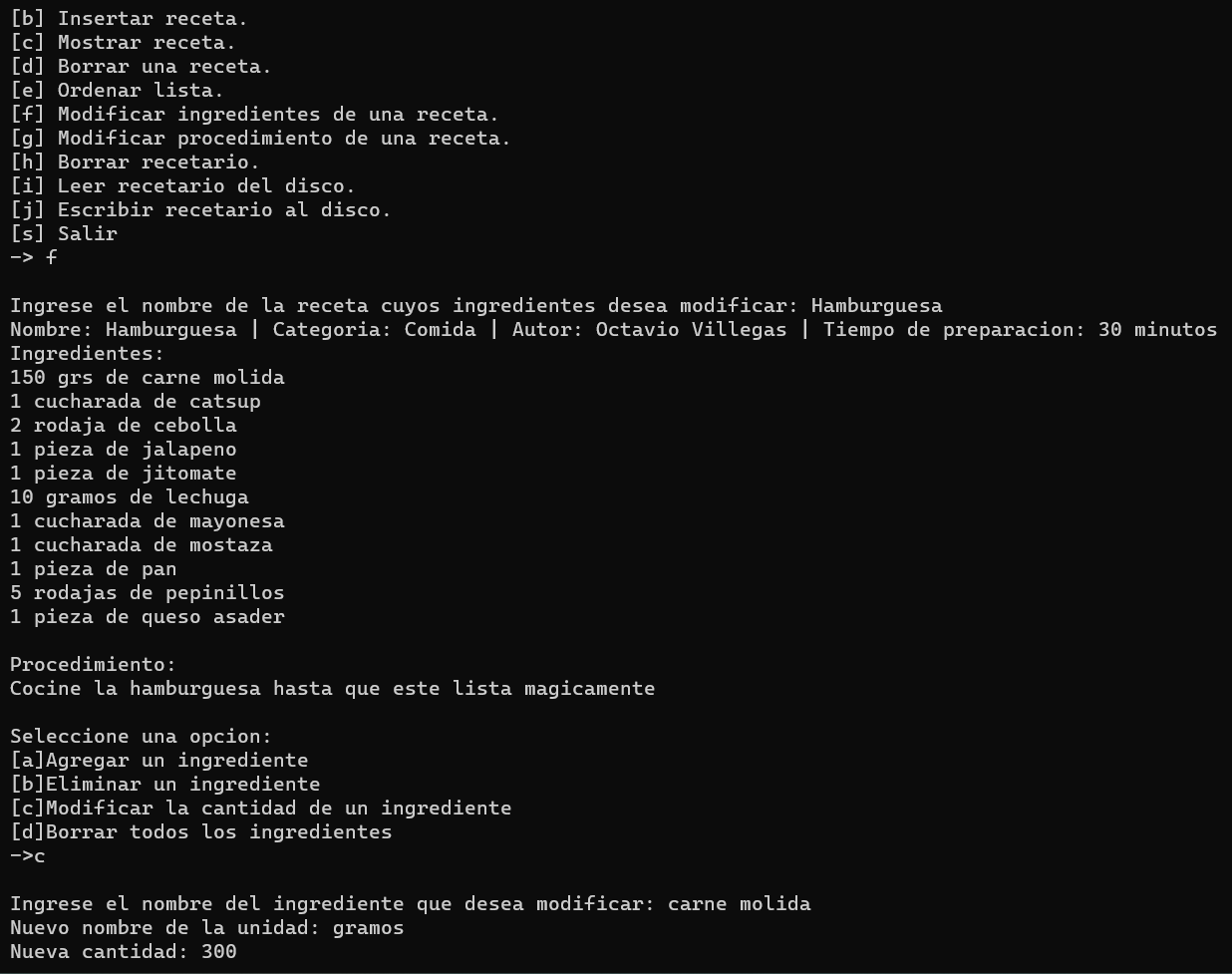
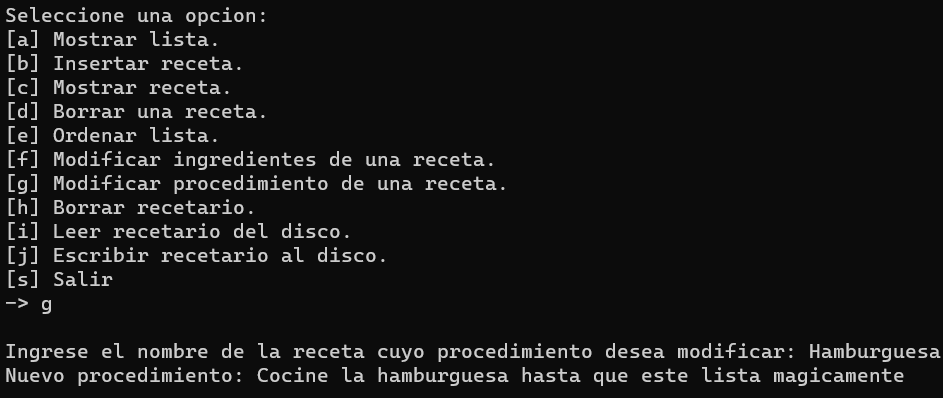
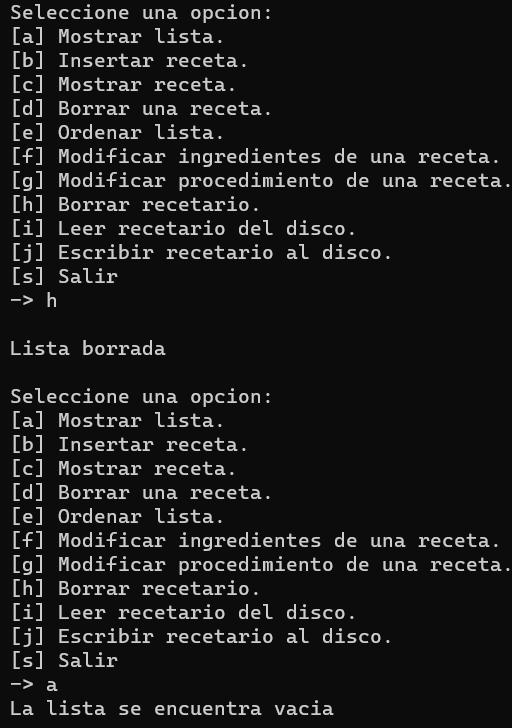
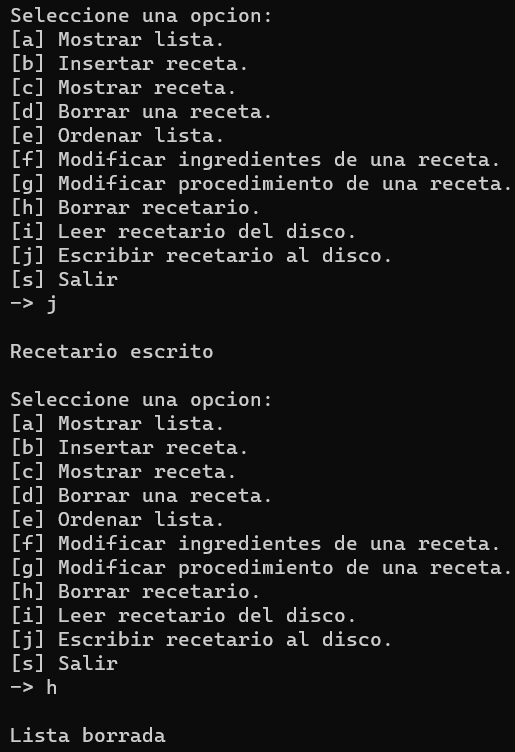
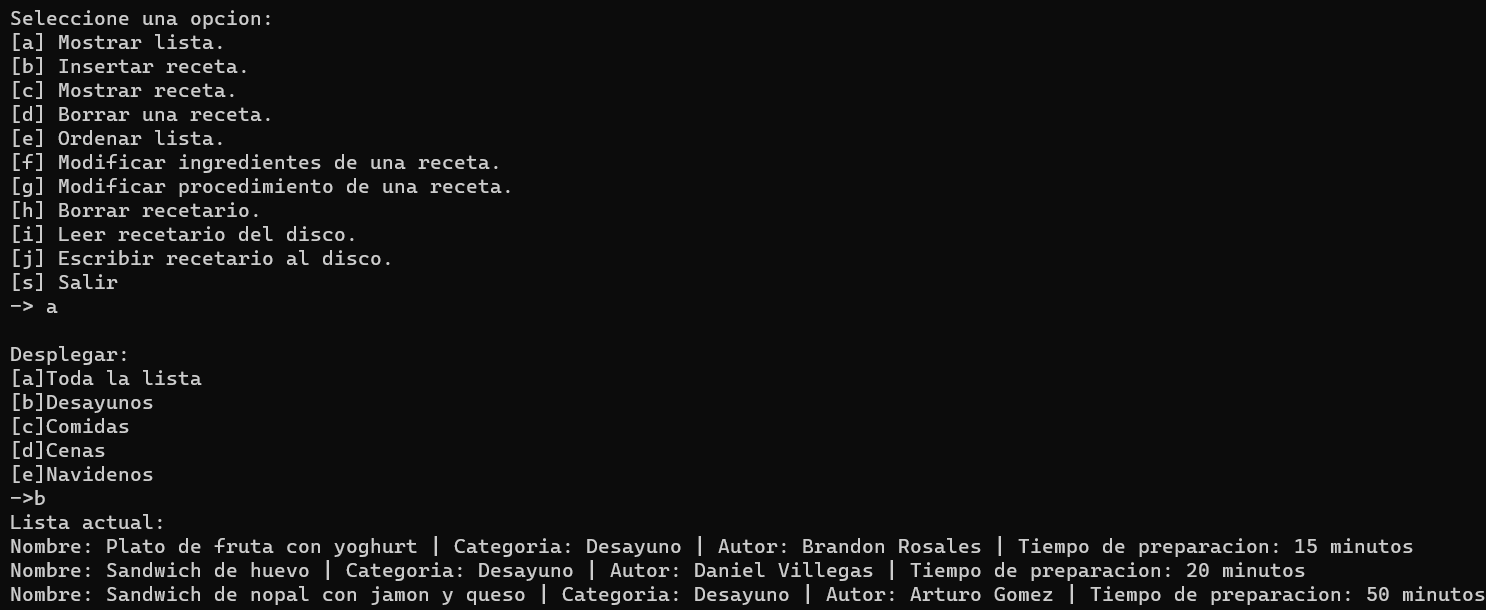
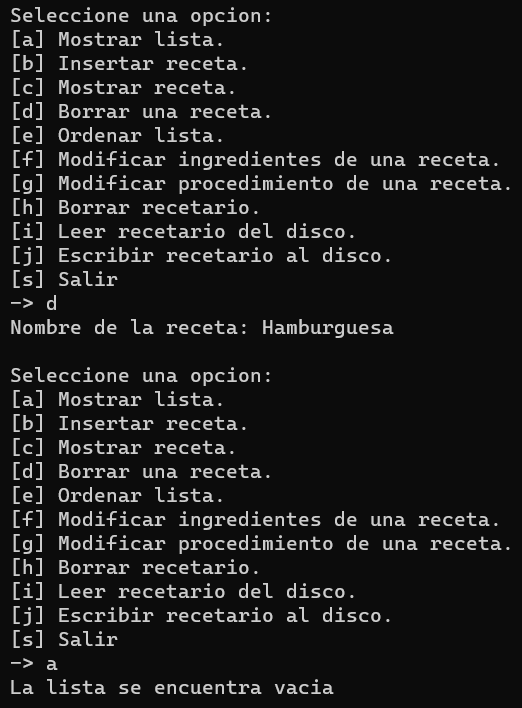
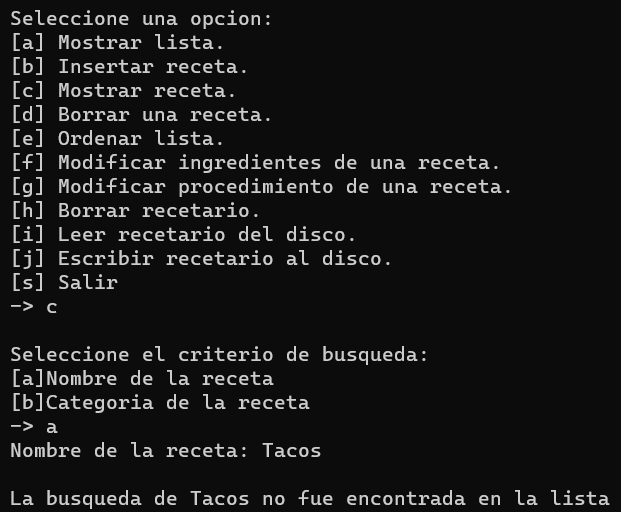
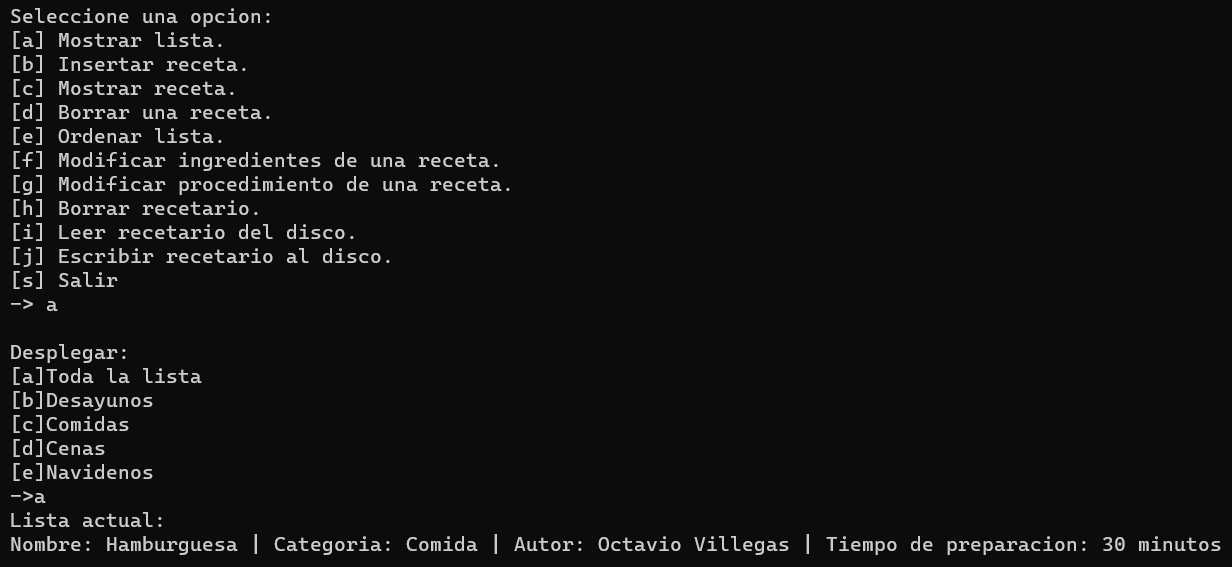
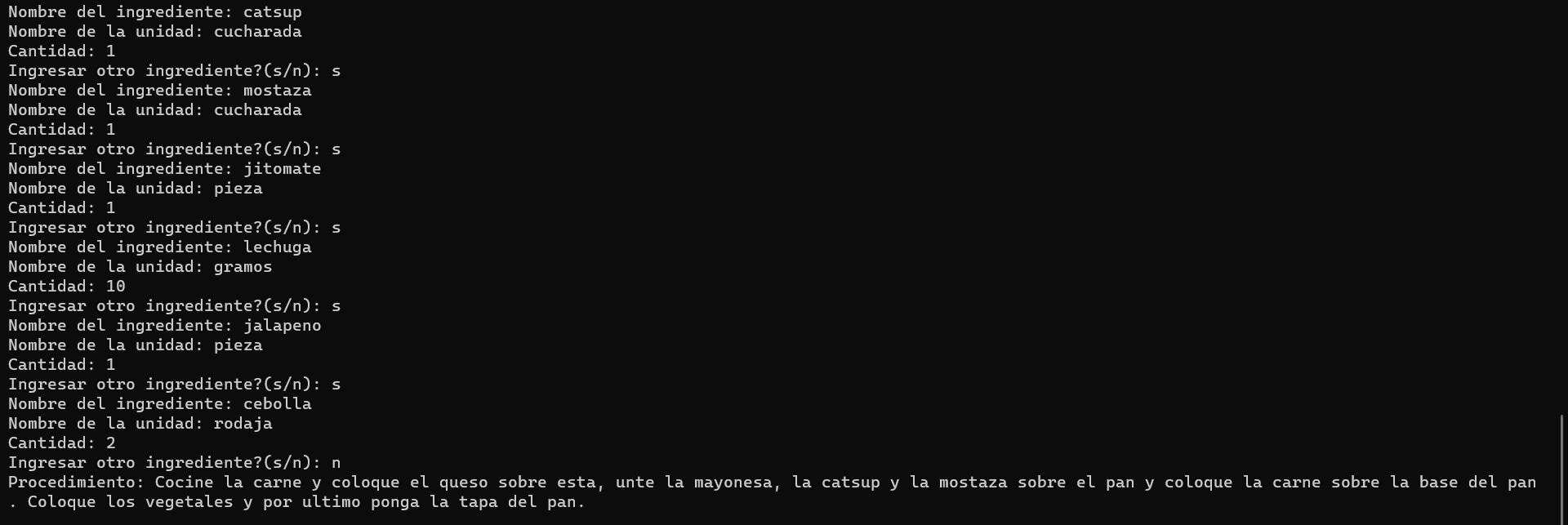
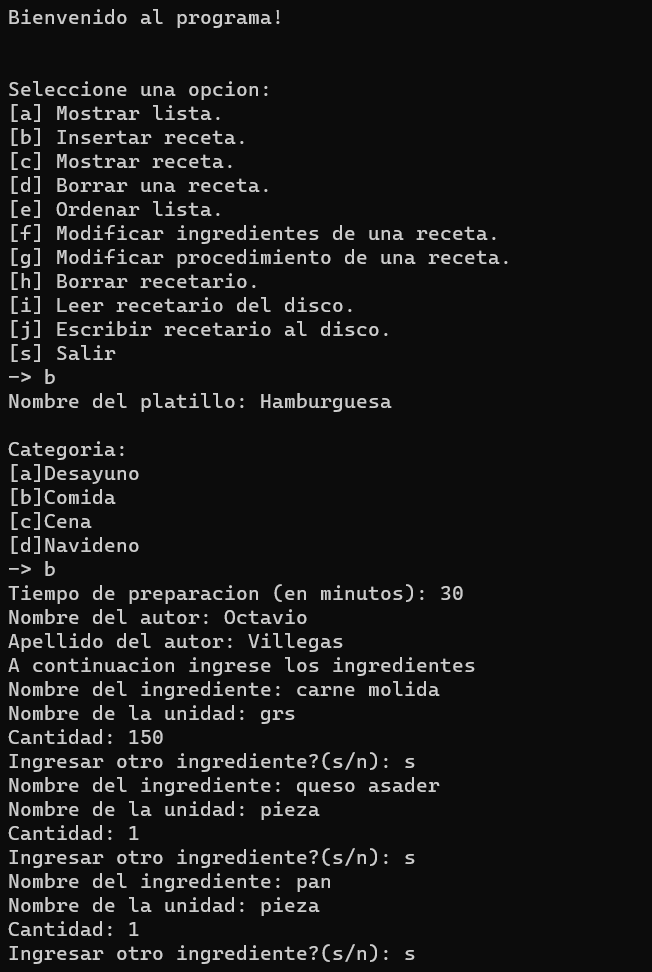
{

Menu myMenu;

myMenu.start();

}

**Capturas de pantalla**

****

**Resumen personal**

*Notas para la entrega final:*

Se realizaron algunos cambios respecto a la entrega preliminar. La clase nombre permaneció igual a la entrega preliminar. Añadí un nuevo método a la clase ingrediente llamada txtString que retorna una cadena con únicamente los datos correspondientes a cada atributo en una nueva línea, lo cual será de utilidad para la función de escribir al disco. La clase receta ahora tiene como atributo ingredientes una lista lineal simplemente enlazada. La clase menú ahora tiene como recetario una lista circular doblemente enlazada.

En cuanto a las listas, estas se implementaron como se trabajaron en las actividades realizadas en el semestre. Aunado a esto, a la lista simplemente ligada se añadió un método que también sirve para retornar un string con solo los datos necesarios para la función de escribir al disco. De esta forma, el operador de flujo de la clase recetas manda a llamar esta función para escribir a un archivo de texto, y al final del último elemento de esta lista se añade un carácter en blanco para poder indicarle a la función de lectura del disco cuando terminar de leer. La lista simplemente ligada no cuenta con métodos para escribir y leer al disco ni un método de ordenamiento, puesto que la inserción de ingredientes de realiza de forma ordenada.

La lista doblemente ligada contiene los mismos métodos del modelo de la lista simplemente ligada, pero con los ajustes necesarios para utilizar los nodos doblemente enlazados. Además, esta contiene los métodos para leer y escribir al disco, así como una función llamada swapData que sirve para intercambiar dos datos recibiendo los apuntadores de dichos datos. Esta función es de utilidad para el método sortList, el cual utiliza el método quickSort para ordenar nuestra lista. También se cuenta con un método llamado sortingPartition el cual nos sirve para asignar un pivote y mandar a llamar el método de ordenamiento de manera recursiva con esta posición.

En la clase menú se realizaron algunos cambios respecto a la entrega preliminar, sin embargo, esencialmente sigue siendo la misma estructura con las mismas funciones. El principal cambio es al momento de la inserción de ingredientes, pues en la entrega preliminar no implementé esta función para que se ingresaran ordenados. Esta vez, el método realiza un recorrido de la lista para encontrar la posición adecuada, comparando el elemento a insertar con el elemento de la posición actual.

Como última observación, el método de la lista doblemente ligada para obtener un dato regresa el apuntador del dato, en lugar del dato. Esto es de ayuda para modificar directamente el dato desde el menú.

*Original (entrega preliminar):*

Para mi trabajo cree las clases nombre, ingrediente y receta además de la clase lista y la clase menú. Estas tres primeras clases se construyeron de acuerdo a los requerimientos, incluyendo sus constructores, métodos interfaz, método toString y sus operadores. Además, se crearon métodos para comparar las instancias de cada clase, los cuales reciben dos objetos y regresan un entero mayor, menor o igual a cero según el resultado de la comparación.

La clase nombre cuenta con los atributos para primer nombre y para el apellido. La clase ingrediente cuenta con tres atributos: uno para el nombre del ingrediente, otro para el nombre de la unidad y otro para la cantidad del ingrediente.

La clase receta será la que almacene la mayor cantidad de atributos, pues cuenta con atributos para el nombre de la receta, el nombre del autor, la categoría del platillo, el tiempo de preparación, la lista de ingredientes y el procedimiento.

También añadí un método toString especial en la lista que tome como parámetro una función para comparar los objetos y que solo añada a la cadena resultado aquellos objetos que cumplan con el criterio deseado.

Para la función para desplegar la receta se utiliza una sentencia switch que manda a llamar este método y despliega los elementos de la lista de la categoría deseada. También se cuenta con la opción de desplegar la lista completa con el método toString clásico. En esta función se despliega una cadena básica de cada receta, sin mostrar los ingredientes y el procedimiento.

La función añadir receta crea instancias de la lista de ingredientes y de cada clase creada para poder settear cada atributo de una instancia de la clase receta. Finalmente se añade este objeto receta a la lista de recetas.

Para encontrar recetas se utilizan nuevamente los métodos para comparar cada objeto para poder encontrar recetas según su nombre o por su categoría. Una vez encontrada la posición de la receta deseada se utiliza el método para desplegar una receta con la posición que se encontró.

Para eliminar la receta se realiza algo similar sin embargo en lugar de utilizar el método para desplegar la receta, se elimina la posición que se encontró. Para eliminar la totalidad de recetas simplemente se utiliza el método anular lista de la lista de recetas.

El ordenamiento de recetas utiliza el ordenamiento quick sort, recibiendo como parámetro la función correspondiente a la comparación que se desea utilizar ya sea por nombre o por tiempo de preparación.

Los métodos que modifican una receta utilizan el mismo principio de encontrar la posición deseada primero y después se copia la receta a una instancia interna de dicha clase. Las modificaciones se hacen primero sobre este objeto. Una vez listo el objeto, sobre la lista de recetas se elimina la posición que se encontró al inicio y se inserta en la misma posición el objeto receta creado internamente en la función. El método para modificar el procedimiento trabaja de manera similar.