МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ

ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)

Кафедра

«Автоматизированные системы управления»

Курсовая работа по дисциплине

«Объектно-ориентированное программирование»

Вариант № 21

Тема:

«Разработка информационной системы «Хлебозавод»

Выполнил: студент группы \_\_\_\_\_\_

Проверил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2019

Оглавление

[Задание 3](#_Toc12101787)

[Введение 5](#_Toc12101788)

[1 Основной раздел 6](#_Toc12101789)

[1.1. Классы программы 6](#_Toc12101790)

[1.2. Коллекции программы 7](#_Toc12101791)

[1.3. Описание запросов LINQ 7](#_Toc12101792)

[2 Текст программы 11](#_Toc12101793)

[3 Файлы инициализации списков 21](#_Toc12101794)

[Заключение 25](#_Toc12101795)

[Список используемых источников 31](#_Toc12101796)

# Задание

Требуется разработать программную систему, предназначенную для обработки данных о хлебозаводах. Такая система должна обеспечивать хранение сведений о производимых изделиях и их стоимости.

Сведения о производимых изделиях включают в себя: наименование изделия, объем производства, дату производства, срок годности, ингредиенты, отпускную цену. Каждый состав характеризуется наименованием, количеством, датой поставки, сроком годности. Состав изделия содержит список возможных ингредиентов: вода, мука пшеничная, мука ржаная, отруби, сахар, масло подсолнечное, соль, дрожжи, добавки. В составе могут отсутствовать некоторые ингредиенты (сахар, масло).

**Хлебозавод Изделие Состав**

Ключ Ключ Ключ

Название Название Ингредиент 1

Дата производства Вес изделия Ингредиент 2

Текущая дата Срок годности Ингредиент 3

Объем производства Ингредиент 4

Цена Ингредиент 5

Ингредиент 6

Вес

Дата поставки

Срок годности

Должны быть созданы обобщенные списки:

• Сведения о хлебозаводе.

• Сведения об изделии.

• Сведения о складе.

Руководству комбината могут понадобиться следующие сведения:

• Какие изделия выпускает заданный хлебозавод.

• Какова суммарная стоимость всех изделий, выпускаемых заданным хлебозаводом.

• Не было ли изделий, производство которых было выполнено после истечения срока какого-либо ингредиента.

• Отсортировать хлебозаводы в порядке убывания объема производства.

• В каком изделии больше всего ингредиентов.

• Какое изделие обеспечит при реализации максимальную прибыль.

Должна быть предусмотрена возможность добавления и удаления изделия для заданного хлебозавода.

# Введение

Целью курсовой работы является разработка в среде Microsoft Visual Studio на языке программирования C# консольного приложения Windows для работы с локальными обобщенными списками. Информация для списков должна сохраняться в файлах, реализованных с помощью приложения «Блокнот». Каждый из списков должен содержать не менее 10 инициализированных объектов класса.

Задачей курсовой работы является закрепление изученного материала и освоение методов обработки информации помощью запросов LINQ.

Программа, реализованная в результате работы, предназначена для управления хлебопекарным бизнесом и позволяет анализировать информацию о деятельности нескольких предприятий.

Основные действия, доступные пользователю программной системы:

1. Просмотр списка изделий, выпускаемых заданным хлебозаводом.
2. Подсчёт суммарной стоимости изделий, выпускаемых хлебозаводом.
3. Просмотр списка изделий с просроченными ингредиентами.
4. Сортировка хлебозаводов в порядке убывания объёма производства.
5. Выявление изделий с наибольшим количеством ингредиентов.
6. Выявление изделий с наибольшей прибылью при реализации.
7. Добавление изделий в список производимой продукции.
8. Удаление изделий из списка производимой продукции.

# 1 Основной раздел

## Классы программы

Класс Factory содержит информацию об одном хлебозаводе. Члены класса приведены в Таблице\_1.

Таблица\_1. Поля класса Factory

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Key | int | Уникальный ключ записи |
| Name | string | Название хлебозавода |
| ProdDate | DateTime | Дата начала производства |

Класс Product содержит данные об одном выпуске конкретной продукции – иначе говоря, речь идёт о целой партии, наименование может совпадать, а дата, вес и другие поля могут отличаться. Члены класса приведены в Таблице\_2.

Таблица\_2. Поля класса Product

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Key | int | Уникальный ключ записи |
| FactoryKey | int | Ключ записи хлебозавода |
| Name | string | Наименование продукции |
| Weight | double | Вес выпущенной продукции |
| ExpiryDate | DateTime | Дата истечения срока годности |
| Count | int | Объём выпущенной партии |
| Price | double | Цена единицы продукции |

Класс Ingredient содержит данные об одной поставке на завод какого-либо ингредиента, используемого в производстве. Члены класса приведены в Таблице\_3.

Таблица\_3. Поля класса Ingredient

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Key | Int | Уникальный ключ записи |
| Name | String | Наименование ингредиента |
| Weight | double | Вес закупленной партии |
| DeliveryDate | DateTime | Дата поступления |
| ExpiryDate | DateTime | Дата истечения срока годности |

Класс Program – стандартный класс приложения на C#, содержащий точку входа программы – статическую функцию Main, с которой начинается выполнение приложения. Этот класс инициализирует обобщённые списки, управляет взаимодействием с пользователем и формирует запросы LINQ.

## Коллекции программы

Все коллекции в программе являются статическими членами класса Program и представлены обобщённым списочным контейнером List, параметризируемым соответствующим классом:

* List<Factory> factories – список хлебозаводов;
* List<Product> products – список изделий;
* List<Ingredient> ingredients – список ингредиентов.

## Описание запросов LINQ

Каждое действие пользователя, кроме добавления и удаления изделий, реализовано при помощи соответствующего запроса LINQ. Каждое действие вынесено в отдельную статическую функцию класса Program.

**Запрос списка изделий, выпускаемых хлебозаводом**

Фильтруем таблицу изделий по запрошенному у пользователя ключу хлебозавода:

products.Where(p => p.FactoryKey == id);

**Запрос суммарной стоимости изделий, выпускаемых хлебозаводом**

Фильтруем таблицу изделий по запрошенному у пользователя ключу хлебозавода и группируем результирующую проекцию по этому ключу, подсчитывая для каждого из них суммарную стоимость выпускаемых изделий:

var result = products.Where(p => p.FactoryKey == id)

.GroupBy(p => p.FactoryKey)

.Select(g => new {

Id = g.Key,

Total = g.Sum(p => p.Price \* p.Count)

});

**Запрос списка изделий с просроченными ингредиентами.**

Сперва получаем объединение по внешнему ключу связующей таблицы consists и таблицы ингредиентов ingredients, включая в результат запроса как значения связующих ключей, так и информационные поля:

var ingr = consists.Join(

ingredients,

c => c.IngredientKey,

i => i.Key,

(c, i) => new {

c.ProductKey,

c.IngredientKey,

i.Name,

i.Weight,

i.ExpiryDate

}

);

Затем объединяем таблицу изделий products и полученную на предыдущем шаге проекцию, группируя результат по изделиям и собирая для каждого из них список просроченных ингредиентов:

var view = products.GroupJoin(

ingr,

p => p.Key,

c => c.ProductKey,

(prod, cons) => new

{

prod.Key,

prod.Name,

Ingredients = cons.Where(w => w.ExpiryDate < DateTime.Now)

}

);

При выводе информации на экран фильтруем результат по наличию просроченных продуктов:

view.Where(v => v.Ingredients.Count() > 0)

**Запрос списка хлебозаводов в порядке убывания объёма производства.**

Объединяем по внешнему ключу FactoryKey таблицы хлебозаводов и изделий и группируем результат, подсчитывая для каждого хлебозавода объем выпущенной продукции. В конце сортируем результат по этому полю:

var result = factories.GroupJoin(

products,

f => f.Key,

p => p.FactoryKey,

(fact, prods) => new

{

fact.Key,

fact.Name,

Count = prods.Sum(prod => prod.Count)

}

).OrderByDescending(f => f.Count);

**Запрос списка изделий с наибольшим количеством ингредиентов.**

Сперва получаем объединение по внешнему ключу связующей таблицы consists и таблицы ингредиентов ingredients, включая в результат запроса значение ключа и наименование ингредиента:

var ingr = consists.Join(

ingredients,

c => c.IngredientKey,

i => i.Key,

(c, i) => new {

c.ProductKey,

i.Name

}

);

Затем объединяем таблицу изделий products и полученную на предыдущем шаге проекцию, группируя результат по изделиям и собирая для каждого из них список ингредиентов:

var view = products.GroupJoin(

ingr,

p => p.Key,

c => c.ProductKey,

(prod, cons) => new

{

prod.Key,

prod.Name,

IngredCount = cons.Count(),

Ingredients = cons.Select(w => w)

}

);

В получившейся проекции вычисляем максимальное количество ингредиентов среди всех изделий:

int maxCount = view.Max(v => v.IngredCount);

При отображении на экране, фильтруем эту проекцию, оставляя только те изделия, у которых количество ингредиентов равно максимальному значению:

view.Where(v => v.IngredCount == maxCount)

**Запрос списка изделий с наибольшей прибылью при реализации.**

Делаем проекцию таблицы изделий, выбирая из неё наименование и вычисляя прибыль как произведение цены за единицу изделия на объём выпуска:

var view = products.Select(p => new {

p.Name,

Revenue = p.Price \* p.Count

});

Вычисляем максимальное значение прибыли среди всех изделий:

var maxVal = view.Max(v => v.Revenue);

Фильтруем эту проекцию, оставляя только те изделия, у которых прибыль равна максимальному значению:

var result = view.Where(v => v.Revenue == maxVal);

# 2 Текст программы

Factory.cs

using System;

namespace bakery

{

// Хлебозавод

class Factory

{

// уникальный ключ

public int Key { set; get; } = 0;

// название предприятия

public string Name { set; get; }

// начало производства

public DateTime ProdDate { set; get; }

}

}

Product.cs

using System;

namespace bakery

{

// Изделие (партия изделий)

class Product

{

// уникальный ключ

public int Key { set; get; } = 0;

// внешний ключ - хлебозавод

public int FactoryKey { set; get; } = 0;

// наименование

public string Name { set; get; }

// вес (кг)

public double Weight { set; get; }

// дата истечения срока годности

public DateTime ExpiryDate { set; get; }

// объём партии

public int Count { set; get; } = 0;

// цена одного изделия

public double Price { set; get; } = 0.0;

}

}

Ingredient.cs

using System;

namespace bakery

{

// Ингредиент

class Ingredient

{

// уникальный ключ

public int Key { set; get; } = 0;

// наименование

public string Name { set; get; }

// вес (кг)

public double Weight { set; get; }

// дата поставки

public DateTime DeliveryDate { set; get; }

// дата истечения срока годности

public DateTime ExpiryDate { set; get; }

}

}

Constitution.cs

namespace bakery

{

// Связь между изделием и ингрединентом

class Constitution

{

// уникальный ключ

public int Key { set; get; } = 0;

// внешний ключ - изделие

public int ProductKey { set; get; } = 0;

// внешний ключ - ингредиент

public int IngredientKey { set; get; } = 0;

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Globalization;

namespace bakery

{

// Основной класс приложения

class Program

{

// список хлебозаводов

static List<Factory> factories = new List<Factory>();

// список изделий

static List<Product> products = new List<Product>();

// список ингредиентов

static List<Ingredient> ingredients = new List<Ingredient>();

// список связей изделий с ингредиентами

static List<Constitution> consists = new List<Constitution>();

// настройки локализации

static CultureInfo \_cultureInfo = (CultureInfo)CultureInfo.CurrentCulture.Clone();

// имена файлов с данными

static string[] paths = new string[]{

"factories",

"products",

"ingredients",

"consists"

};

// основная функция - точка входа приложения

static void Main(string[] args)

{

// дробная часть в файлах отделена точкой, а не запятой

\_cultureInfo.NumberFormat.NumberDecimalSeparator = ".";

Console.Title = "Сеть хлебозаводов";

// загрузить список хлебзаводов

ReadList(0, 2);

// загрузить список изделий

ReadList(1, 6);

// загрузить список ингредиентов

ReadList(2, 5);

// загрузить список, связывающий изделия и ингредиенты

ReadList(3, 2);

// цикл взаимодействия с пользователем

while (true)

{

// напечатать меню и запросить выбор пользователя

char choice = UserChoice();

Console.WriteLine("Вы выбрали действие: " + choice);

if (choice == '0')

{

// 0 - завершаем работу

break;

}

switch(choice)

{

case '1': // Перечень изделий, выпускаемых хлебозаводом

FactoryProductsList();

break;

case '2': // Суммарная стоимость изделий, выпускаемых хлебозаводом

TotalCost();

break;

case '3': // Изделия с просроченными ингредиентами

ExpiredProductsList();

break;

case '4': // Список хлебозаводов в порядке убывания объёма производства

DescendingFactoriesList();

break;

case '5': // Изделие с наибольшим количеством ингредиентов

MaxIngredientsProduct();

break;

case '6': // Изделие с наибольшей прибылью при реализации

MaxRevenueProduct();

break;

case '7': // Добавить изделие, выпускаемое хлебозаводом

AddProduct();

break;

case '8': // Удалить изделие, выпускаемое хлебозаводом

RemoveProduct();

break;

default:

Console.WriteLine("Такого действия нет в списке");

break;

}

Console.Write("Нажмите любую клавишу для продолжения работы ");

Console.ReadKey(true);

}

}

// Вывод пользовательского меню и запрос выбора

static char UserChoice()

{

Console.Clear();

Console.WriteLine("0 - Выход");

Console.WriteLine("1 - Перечень изделий, выпускаемых хлебозаводом");

Console.WriteLine("2 - Суммарная стоимость изделий, выпускаемых хлебозаводом");

Console.WriteLine("3 - Изделия с просроченными ингредиентами");

Console.WriteLine("4 - Список хлебозаводов в порядке убывания объёма производства");

Console.WriteLine("5 - Изделие с наибольшим количеством ингредиентов");

Console.WriteLine("6 - Изделие с наибольшей прибылью при реализации");

Console.WriteLine("7 - Добавить изделие, выпускаемое хлебозаводом");

Console.WriteLine("8 - Удалить изделие, выпускаемое хлебозаводом");

return Console.ReadKey(true).KeyChar;

}

// Запрос у пользователя номера интересующего его хлебозавода

static int RequestFactoryKey()

{

Console.WriteLine("Список заводов:");

Console.WriteLine("№\tНазвание");

foreach (var item in factories)

{

Console.WriteLine("{0}\t{1}", item.Key, item.Name);

}

Console.Write("Введите № завода: ");

return Convert.ToInt32(Console.ReadLine().Trim());

}

// Перечень изделий, выпускаемых хлебозаводом

// choose - запросить номер у пользователя

// если false, то просто выводим список изделий

static int FactoryProductsList(bool choose = false)

{

// выводим список всех хлебозаводов

// и запрашиваем у пользователя номер

int id = RequestFactoryKey();

// фильтруем все изделий по номеру выбранного хлебозавода

var result = products.Where(p => p.FactoryKey == id);

// выводим список всех изделий в этой проекции

Console.WriteLine("№\tВес\tЦена\tВыпуск\tГоден до\t\tНазвание");

foreach (var p in result)

{

Console.WriteLine("{0}\t{1}\t{2}\t{3}\t{4}\t{5}",

p.Key, p.Weight, p.Price, p.Count, p.ExpiryDate, p.Name);

}

if (choose)

{

// запрашиваем у пользователя номер хлебозавода

Console.Write("Введите № партии: ");

return Convert.ToInt32(Console.ReadLine().Trim());

}

return 0;

}

// Суммарная стоимость изделий, выпускаемых хлебозаводом

static void TotalCost()

{

// запросить у пользователя номер хлебозавода

int id = RequestFactoryKey();

// фильтруем изделия по выбранному номеру хлебозавода

var result = products.Where(p => p.FactoryKey == id)

// группируем изделия по номеру хлебозавода

.GroupBy(p => p.FactoryKey)

// суммируем стоимость партии в каждой группе

.Select(g => new {

Id = g.Key,

Total = g.Sum(p => p.Price \* p.Count)

});

// вывод результата

foreach (var group in result)

{

Console.WriteLine("Суммарная стоимость выпущенных изделий: {0} руб.", group.Total);

}

}

// Изделия с просроченными ингредиентами

static void ExpiredProductsList()

{

// сперва получаем объединение по внешнему ключу

// таблицы consists и таблицы ингредиентов ingredients,

// включая в результат запроса как значения связующих

// ключей, так и информационные поля

var ingr = consists.Join(

ingredients,

c => c.IngredientKey,

i => i.Key,

(c, i) => new {

c.ProductKey,

c.IngredientKey,

i.Name,

i.Weight,

i.ExpiryDate

}

);

// затем объединяем таблицу изделий products и полученную

// на предыдущем шаге проекцию, группируя результат по изделиям

// и собирая для каждого из них список просроченных ингредиентов

var view = products.GroupJoin(

ingr,

p => p.Key,

c => c.ProductKey,

(prod, cons) => new

{

prod.Key,

prod.Name,

Ingredients = cons.Where(w => w.ExpiryDate < DateTime.Now)

}

);

Console.WriteLine("Изделия с просроченным ингредиентами");

Console.WriteLine("№\tНазвание");

// вывод результата (только позиции с просроченными ингредиентами)

foreach (var p in view.Where(v => v.Ingredients.Count() > 0))

{

Console.WriteLine("{0}\t{1} (просроченных ингредиентов: {2}):",

p.Key, p.Name, p.Ingredients.Count());

foreach (var i in p.Ingredients)

{

Console.WriteLine("\t - {0} ({1}) {2}", i.ExpiryDate, i.IngredientKey, i.Name);

}

}

}

// Список хлебозаводов в порядке убывания объёма производства

static void DescendingFactoriesList()

{

// объединяем по внешнему ключу FactoryKey таблицы

// хлебозаводов и изделий и группируем результат,

// подсчитывая для каждого хлебозавода объем выпущенной продукции.

// В конце сортируем результат по этому полю.

var result = factories.GroupJoin(

products,

f => f.Key,

p => p.FactoryKey,

(fact, prods) => new

{

fact.Key,

fact.Name,

Count = prods.Sum(prod => prod.Count)

}

).OrderByDescending(f => f.Count);

// вывод результата

Console.WriteLine("№\tНазвание\tОбъём производства");

foreach (var f in result)

{

Console.WriteLine("{0}\t{1}\t\t{2}", f.Key, f.Name, f.Count);

}

}

// Изделие с наибольшим количеством ингредиентов

static void MaxIngredientsProduct()

{

// сперва получаем объединение по внешнему ключу таблицы consists

// и таблицы ингредиентов ingredients, включая в результат запроса

// значение ключа и наименование ингредиента

var ingr = consists.Join(

ingredients,

c => c.IngredientKey,

i => i.Key,

(c, i) => new {

c.ProductKey,

i.Name

}

);

// затем объединяем таблицу изделий products и полученную

// на предыдущем шаге проекцию, группируя результат по изделиям

// и собирая для каждого из них список ингредиентов

var view = products.GroupJoin(

ingr,

p => p.Key,

c => c.ProductKey,

(prod, cons) => new

{

prod.Key,

prod.Name,

IngredCount = cons.Count(),

Ingredients = cons.Select(w => w)

}

);

// в получившейся проекции вычисляем максимальное

// количество ингредиентов среди всех изделий

int maxCount = view.Max(v => v.IngredCount);

// вывод результата (только позиции с наибольшим числом ингредиентов)

Console.WriteLine("Изделия с наибольшим числом ингредиентов");

Console.WriteLine("№\tНазвание");

foreach (var p in view.Where(v => v.IngredCount == maxCount))

{

Console.WriteLine("{0}\t{1} (ингредиентов: {2}):",

p.Key, p.Name, p.IngredCount);

foreach (var i in p.Ingredients)

{

Console.WriteLine("\t - {0}", i.Name);

}

}

}

// Изделие с наибольшей прибылью при реализации

static void MaxRevenueProduct()

{

// делаем проекцию таблицы изделий, выбирая из неё

// наименование и вычисляя прибыль как произведение

// цены за единицу изделия на объём выпуска

var view = products.Select(p => new {

p.Name,

Revenue = p.Price \* p.Count

});

// вычисляем максимальное значение прибыли среди всех изделий

var maxVal = view.Max(v => v.Revenue);

// фильтруем эту проекцию, оставляя только те изделия,

// у которых прибыль равна максимальному значению

var result = view.Where(v => v.Revenue == maxVal);

// вывод результата

Console.WriteLine("Изделия с наибольшей прибылью:");

foreach (var item in result)

{

Console.WriteLine("{0} ({1} руб).", item.Name, item.Revenue);

}

}

// Добавить изделие, выпускаемое хлебозаводом

static void AddProduct()

{

// выводим на экран список заводов хлебозаводов

// и запрашиваем у пользователя номер

int factoryKey = RequestFactoryKey();

// формируем ключ

int key = products.Count + 1;

// запрашиваем у пользователя остальные поля

Console.Write("Введите наименование продукции: ");

string name = Console.ReadLine().Trim();

Console.Write("Введите вес единицы продукции (кг): ");

double weight = double.Parse(Console.ReadLine().Trim(), \_cultureInfo);

Console.Write("Введите дату окончания срока годности (ГГГГ-ММ-ДД): ");

DateTime date = DateTime.Parse(Console.ReadLine().Trim());

Console.Write("Введите объём партии (шт): ");

int count = Convert.ToInt32(Console.ReadLine().Trim());

Console.Write("Введите цену единицы продукции: ");

double price = double.Parse(Console.ReadLine().Trim(), \_cultureInfo);

// создаём объект и добавляем в таблицу

Product product = new Product();

product.Key = key;

product.FactoryKey = factoryKey;

product.Name = name;

product.Weight = weight;

product.ExpiryDate =date ;

product.Count = count;

product.Price = price;

products.Add(product);

}

// Удалить изделие, выпускаемое хлебозаводом

static void RemoveProduct()

{

// выводим на экран список заводов изделий

// и запрашиваем у пользователя номер

int key = FactoryProductsList(true);

// находим изделие с таким номером (ключом)

var item = products.SingleOrDefault(p => p.Key == key);

// если нашли - удаляем

if (item != null)

{

products.Remove(item);

}

}

// Загрузка списка из файла

static void ReadList(int index, int fieldCount)

{

if(index >= paths.Length)

return;

// открываем файл на чтение

StreamReader stream = new StreamReader(paths[index], Encoding.Default);

string line;

int key = 1;

switch(index)

{

case 0:

// читаем построчно файл со списком хлебозаводов

while ((line = stream.ReadLine()) != null)

{

string[] fields = line.Split(';');

if(fields.Length == fieldCount)

{

Factory item = new Factory();

item.Key = key++;

item.Name = fields[0];

item.ProdDate = DateTime.Parse(fields[1]);

factories.Add(item);

}

}

break;

case 1:

// читаем построчно файл со списком изделий

while ((line = stream.ReadLine()) != null)

{

string[] fields = line.Split(';');

if (fields.Length == fieldCount)

{

Product item = new Product();

item.Key = key++;

item.FactoryKey = Convert.ToInt32(fields[0]);

item.Name = fields[1];

item.Weight = double.Parse(fields[2], \_cultureInfo);

item.ExpiryDate = DateTime.Parse(fields[3]);

item.Count = Convert.ToInt32(fields[4]);

item.Price = double.Parse(fields[5], \_cultureInfo);

products.Add(item);

}

}

break;

case 2:

// читаем построчно файл со списком иигредиентов

while ((line = stream.ReadLine()) != null)

{

string[] fields = line.Split(';');

if (fields.Length == fieldCount)

{

Ingredient item = new Ingredient();

item.Key = key++;

item.Name = fields[0];

item.Weight = double.Parse(fields[1], \_cultureInfo);

item.DeliveryDate = DateTime.Parse(fields[2]);

item.ExpiryDate = DateTime.Parse(fields[3]);

ingredients.Add(item);

}

}

break;

case 3:

// читаем построчно файл со списком связей

while ((line = stream.ReadLine()) != null)

{

string[] fields = line.Split(';');

if (fields.Length == fieldCount)

{

Constitution item = new Constitution();

item.Key = key++;

item.ProductKey = Convert.ToInt32(fields[0]);

item.IngredientKey = Convert.ToInt32(fields[1]);

consists.Add(item);

}

}

break;

} // switch

Console.WriteLine(paths[index]);

}

}

}

# 3 Файлы инициализации списков

Сразу после запуска приложения происходит инициализация списков. Для этого используются обычные текстовые файлы, которые можно редактировать с помощью обычного «Блокнота» Windows. Каждой записи – элементу списка – соответствует одна строка. Поля в пределах одной строки отделяются символом точки с запятой – «;». Дробная часть действительных чисел отделена точкой.

Уникальные ключи в файлах не указываются – они формируются программой автоматически, в процессе считывания файлов и заполнения списков.

Список хлебозаводов содержится в файле factories:

Каравай;2005-05-08

Колос;2001-08-20

Эвертон;2003-05-21

Армхлеб;2004-03-05

Лимак;2001-02-22

Рускон;2015-09-02

Ирмень;2010-04-07

Орфей;2012-08-08

Фортуна;2008-08-08

Восход;2003-11-12

Первое значение – название хлебозаводы, далее указана дата запуска производства.

Список хлебобулочных и кондитерских изделий содержится в файле products:

1;Батон нарезной;0.2;2019-06-23;1000;38.0

1;Батон нарезной;0.2;2019-06-25;1500;38.0

2;Рогалик;0.1;2019-06-19;1500;22.5

1;Ржаной хлеб;0.22;2019-06-21;1200;35.2

1;Фермерский хлеб;0.18;2019-06-20;1100;38.0

2;Пирожное;0.12;2019-06-18;1000;55.8

2;Пирожное;0.12;2019-06-24;1100;55.8

1;Булочка;0.15;2019-06-20;1300;40.0

1;Лаваш;0.25;2019-06-20;1000;45.4

3;Фруктовая корзинка;0.2;2019-06-25;1000;73.0

Первым значением указано значение внешнего ключа, равно соответствующему значению уникального ключа хлебозавода. Далее последовательно идут следующие поля: наименование, вес (кг), дата истечения срока годности, объём партии и цена одного изделия.

Список ингредиентов, используемых при изготовлении продукции, содержится в файле products:

Вода;100;2019-06-15;2019-07-15;

Вода;200;2019-06-18;2019-07-18;

Вода;300;2019-06-19;2019-07-19;

Мука пшеничная;250;2019-06-01;2019-10-01;

Мука пшеничная;100;2019-03-10;2019-06-10;

Мука ржаная;100;2019-06-15;2019-09-15;

Отруби;80;2019-06-05;2019-11-05;

Сахар;200;2019-05-20;2020-05-20;

Сахар;100;2019-06-07;2020-06-07;

Масло подсолнечное;20;2019-05-20;2020-08-20;

Масло подсолнечное;100;2019-05-25;2020-08-25;

Масло подсолнечное;100;2019-06-01;2020-09-01;

Масло подсолнечное;50;2019-06-20;2020-09-20;

Соль;10;2019-04-10;2022-04-10;

Соль;50;2019-05-20;2022-05-20;

Дрожжи;50;2019-05-30;2019-06-30;

Каждая строка начинается с наименования, далее следуют вес партии, хранящейся на складе (кг), дата доставки и дата окончания срока годности.

Таблица, определяющая связь между изделиями и ингредиентами – т.е. какие ресурсы были использованы при изготовлении какой партии продукции, содержится в файле consists:

1;1

1;4

1;8

1;10

2;2

2;5

2;8

2;11

3;1

3;4

3;9

3;13

3;14

3;16

4;3

4;6

4;7

4;12

4;15

5;2

5;5

5;6

5;7

5;12

5;15

5;16

6;1

6;5

6;9

6;10

6;14

6;16

7;3

7;5

7;9

7;11

7;14

7;16

8;2

8;4

8;7

8;8

8;10

8;14

8;16

9;2

9;4

9;11

9;14

9;16

10;1

10;6

10;9

10;10

10;14

10;16

Первым значением указан ключ партии готового изделия, вторым – ключ партии ингредиента. Таким образом реализуется отношение «многие ко многим» между таблицами изделий и ингредиентов, поскольку, например, одна и та же партия муки могла использоваться при изготовлении нескольких изделия (например, батоны и булочки), а одно и то же изделие (батон) изготавливается из нескольких ингредиентов (вода, мука и т.д.).

# Заключение

В результате работы была спроектирована и разработана программа, позволяющая анализировать хозяйственную деятельность нескольких хлебозаводов – просматривать выпускаемую продукцию, фильтровать, сортировать и группировать данные в соответствии с различными критериями.

В ходе работы над проектом получены практические навыки разработки программ на языке высокого уровня C#. Изучены особенности создания приложений с помощью этого языка программирования и среды разработки Microsoft Visual Studio. Кроме того, усвоены методы обработки данных с использованием механизма запросов LINQ.

В целом, на базе практического применения, закреплены базовые знания, полученные ранее, при изучении теории.

В заключение продемонстрируем результаты запросов в работающем приложении. Главное окно программы с пользовательским меню приведено на рисунке 1.

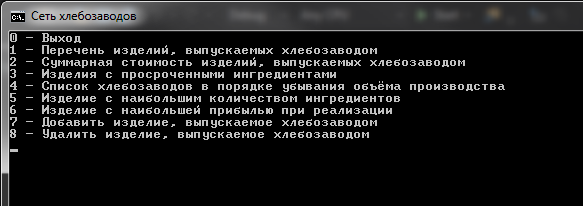


Рис. 1. Основное меню программы.

Список выпускаемых выбранным хлебозаводом изделий показан на рисунке 2, выбран завод «Колос» со значением ключа (в интерфейсе программы он назван «Номером») равным 2.

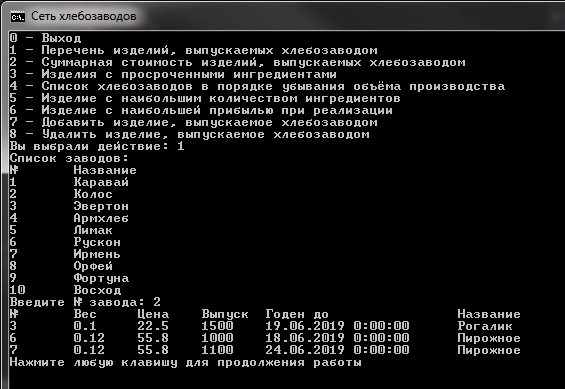


Рис. 2. Список выпускаемых хлебозаводом изделий.

Расчёт суммарной стоимости изделий, выпускаемых выбранным хлебозаводом, показан на рисунке 3. Выбран завод «Каравай» со значением уникального ключа, равным 1.

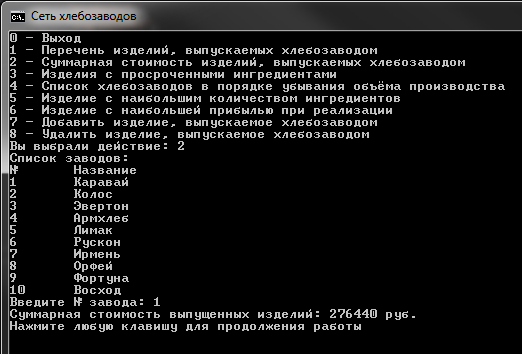


Рис. 3. Суммарная стоимость выпускаемых заводом изделий.

Список изделий, у которых закончился срок годности одного или нескольких ингредиентов, изображён на рисунке 4.

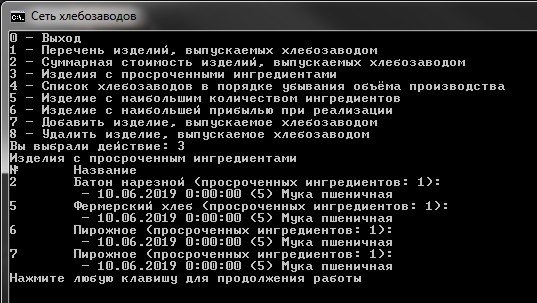


Рис. 4. Список изделий с просроченными ингредиентами.

Как видно из рисунка, программа под каждым изделием уточняет, какие именно ингредиенты просрочены.

Список хлебозаводов, отсортированный в порядке убывания объёма производства, показан на рисунке 5.

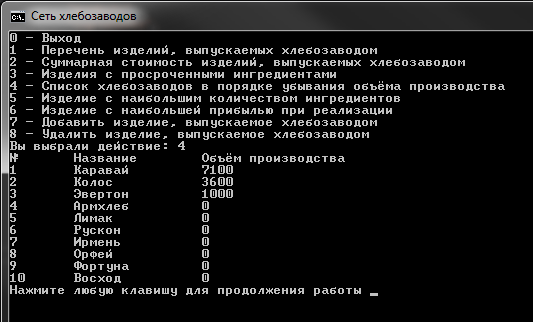


Рис. 5. Отсортированный список хлебозаводов.

Список изделий, для производства которых требуется наибольшее число ингредиентов, показан на рисунке 6.

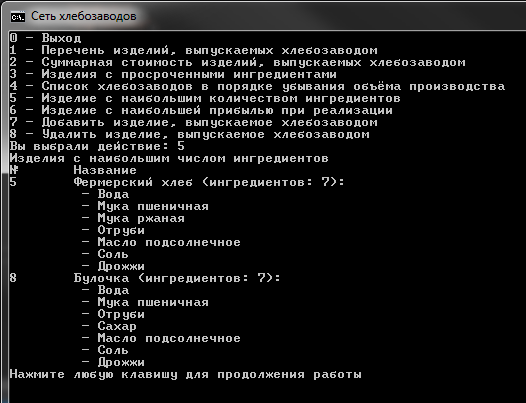


Рис. 6. Изделие с наибольшим количеством ингредиентов.

Как видно из рисунка, таких изделий может быть несколько – в случае, если для них требуется одинаковое количество ингредиентов.

Список изделий с максимальной прибылью, ожидаемой при реализации партии, изображён на рисунке 7.

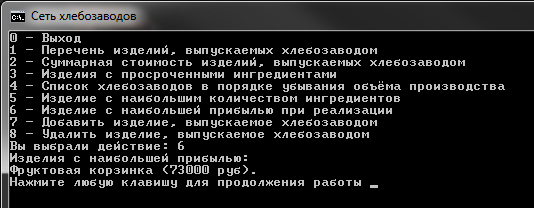


Рис. 7. Изделия с наибольшей прибылью.

В данном случае список состоит из одного элемента.

Процедура добавления нового изделия показана на рисунке 8. Для проверки того, что изделие действительно добавлено в список, используем действие №1 – «Перечень изделий, выпускаемых хлебозаводом» - выбирая то же предприятие, что и при добавлении.

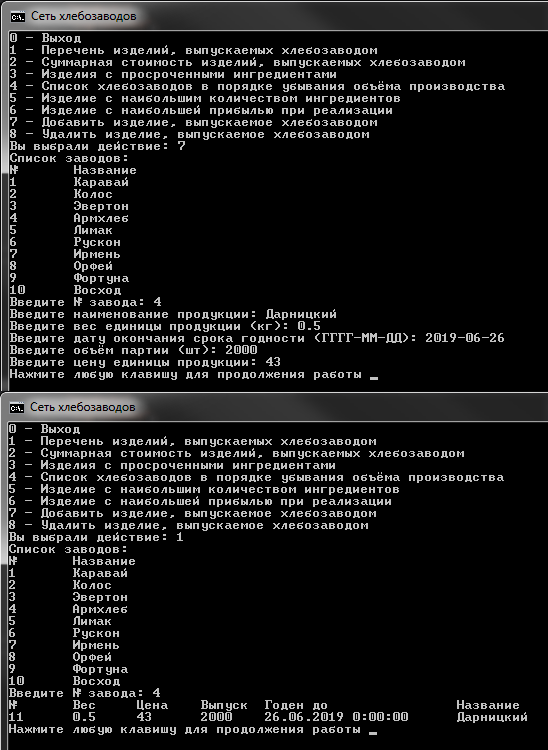


Рис. 8. Добавление изделия.

Порядок действий при удалении изделия показан на рисунке 9.

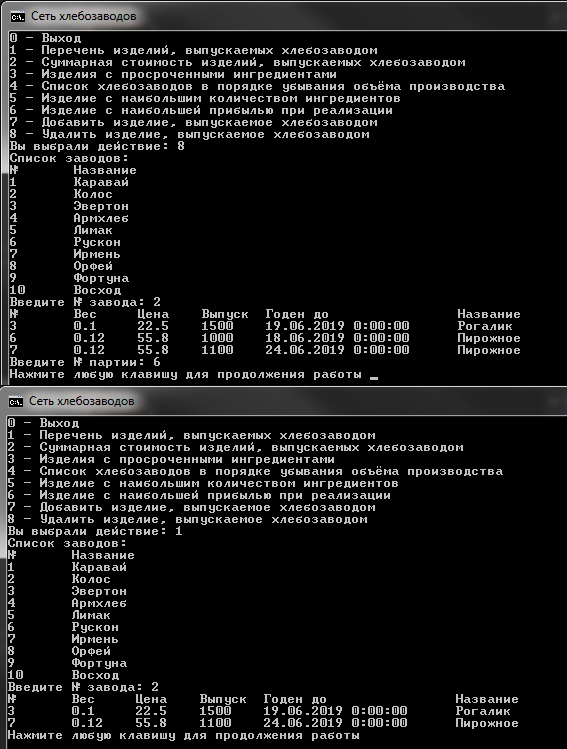


Рис. 9. Удаление изделия.

Как видно из рисунков 8 и 9, операции добавления и удаления выполняются успешно. В задании, файлы предполагается использовать только для инициализации обобщённых списков, поэтому сохранения нет.

# Список используемых источников

1. Вагнер, Билл С# Эффективное программирование / Билл Вагнер. – М.: ЛОРИ, 2013. – 320 c.

2. Котов, О.М. Язык C#. Краткое описание и введение в технологии программирования / О.М. Котов. – Екатеринбург: Издательствово Урал. унта, 2014. – 208 с.

3. Чернов, Э.А. Основы программирования на языке C# / Э.А. Чернов. – Дюссельдорф, Германия: LAP Lambert Academic Publishing, 2015. – 310 с.

4. Абрамян, Михаил Visual C# на примерах / Михаил Абрамян. – М.: БХВ-Петербург, 2015. – 572 c.

5. Культин, Н. С# в задачах и примерах / Н. Культин. – М.: БХВ-Петербург, 2016. – 952 c.

6. Ишкова, Э. А. Самоучитель С#. Начала программирования / Э.А. Ишкова. – М.: Наука и техника, 2017. – 496 c