Постановка задачи

Однонаправленный кольцевой список. Упорядочить список так, чтобы фамилии вкладчиков располагались в алфавитном порядке. Подсчитать общую сумму всех вкладов.

Список вкладчиков Сбербанка (фамилия и инициалы вкладчика, номер счета и сумма вклада).

Исходные данные

Исходные данные задачи размещаются в файле input.txt, расположенном в рабочей папке программы. Формат файла текстовый в кодировке UTF-8, каждая строка файла соответствует одной записи вкладчика. Строка представляет собой набор последовательностей символов, разделенных символом «|», каждая из последовательностей соответствует одному полю. Порядок следования полей в строке следующий: «Фамилия и инициалы вкладчика» (строковый), «Номер счета» (строковый), «Сумма вклада» (числовой). В качестве разделителя дробной части в числовом поле используется символ «,».

Текст программы

```
using System;
using System.Globalization;
using System.IO;

namespace RingList
{
    class Program
    {
        static void Main(string[] args)
        {
            // Создаем кольцевой список вкладчиков
            AccountRingList accounts = new AccountRingList();
            // Открываем на чтение текстовый файл input.txt
            using (StreamReader reader = new StreamReader("input.txt"))
            {
                string line;
                // Создаем объект-правило преобразования строк в числа
                NumberFormatInfo provider = new NumberFormatInfo();
                // Разделитель дробной части - запятая
                provider.NumberDecimalSeparator = ",";
                // Считываем файл в цикле, пока есть строки
                while ((line = reader.ReadLine()) != null)
                {
                    // Делим считанную строку на части
                    string[] parts = line.Split('|');
                    // Добавляем в кольцевой список вкладчика
                    // Первая часть - ФИО, вторая - номер счета, третья - сумма вклада
                    accounts.Add(
                        new Account(parts[0], parts[1], Convert.ToDecimal(parts[2])));
                }
                // Сортируем список вкладчиков
                accounts.Sort();
                // Выводим список вкладчиков
                Console.WriteLine("Список вкладчиков:");
                accounts.Print();
                // Выводим общую сумму вкладов
```

```
            Console.WriteLine(
                string.Format(
                    "Общая сумма вкладов: {0:0.00}", accounts.CalcTotalBalance()));
        }
    }
}

// Структура, описывающая вклад
struct Account
{
    // Инициализирующий конструктор
    public Account(string owner, string number, decimal balance)
    {
        this.owner = owner;
        this.number = number;
        this.balance = balance;
    }

    // ФИО вкладчика
    public string owner { get; }
    // Номер счета
    public string number { get; }
    // Сумма вклада
    public decimal balance { get; }
}

// Элемент кольцевого списка
class AccountRingListItem
{
    // Инициализирующий конструктор
    public AccountRingListItem(Account data)
    {
        this.data = data;
    }

    // Указатель на следующий элемент списка
    public AccountRingListItem next;
    // Данные элемента (вклад)
    public Account data { get;  }
}

// Кольцевой список вкладчиков
class AccountRingList
{
    // Первый элемент списка
    private AccountRingListItem first;
    // Последний элемент списка (для быстрого добавления в конец)
    private AccountRingListItem last;

    // Метод добавления в конец списка
    public void Add(Account account)
    {
        // Создаем новый элемент списка с вкладом
        AccountRingListItem item = new AccountRingListItem(account);
        if (first == null)
            // Если список пуст, то назначаем новый элемент первым
            first = item;
        else
```

```
            // Иначе добавляем новый элемент к последнему
            last.next = item;
        // Теперь новый элемент становится последним
        last = item;
        // Закольцовываем список: следующий элемент у последнего - первый в списке
        item.next = first;
    }

    // Метод сортировки списка
    public void Sort()
    {
        // Устанавливаем объект для сравнения строк - в порядке следованию по
        // алфавиту кириллицы
        var ru = new System.Globalization.CultureInfo("ru-RU");
        // Отсортируем элементы списка выбором. Для этого для каждой позиции в списке
        // с самого начала выберем минимальный элемент из оставшейся части списка и
        // поместим его в эту позицию. Таким образом, на первой позиции будет
        // минимальный элемент из всех, на второй - минимальный из оставшихся и т.д.,
        // т.е. список станет упорядочен по неубыванию элементов, в нашем случае -
        // в лексикографическом порядке ФИО

        // Для перемещения элемента нам нужно две позиции: на сам элемент, чтобы
        // переместить его, и на предыдущий элемент, чтобы поменять указатель в нем
        // на следующий элемент
        AccountRingListItem prev = last;
        AccountRingListItem curr = first;
        // Идем в цикле от первого элемента до последнего
        while (curr != last)
        {
            // Назначаем текущий минимальный элемент - элемент в рассматриваемой
            // позиции
            AccountRingListItem prevBest = prev;
            AccountRingListItem best = curr;

            // Пробегаемся по оставшейся части списка в поисках меньшего элемента,
            // начиная с элемента, следующего за рассматриваемой позицией
            AccountRingListItem prevCheck = curr;
            AccountRingListItem check = curr.next;
            while (check != first)
            {
                // Сравниваем текущий элемент с лучшим минимумом
                if (String.Compare(check.data.owner, best.data.owner, true, ru) < 0)
                {
                    // Если меньше - назначим новый минимум и его предыдущий
                    best = check;
                    prevBest = prevCheck;
                }
                // Продвигаемся дальше по списку
                prevCheck = check;
                check = check.next;
            }
            if (best != curr)
            {
                // Нашли в оставшейся части списка меньший элемент - вставляем
                // перед рассматриваемым

                // Извлекаем минимальный элемент из его текущей позиции, для
                // этого в предшествующем ему элементе ставим указатель на следующий
```

```
                // через один
                prevBest.next = best.next;
                if (best == last)
                    // Если минимальный элемент был последним, обновляем последний
                    last = prevBest;
                // Вставляем минимальный элемент перед рассматриваемым, т.е.
                // следующий за минимальным - рассматриваемый
                best.next = curr;
                // следующий за предшествующий рассматриваемому - минимальный
                prev.next = best;
                if (first == curr)
                    // Если у нас первым элементом списка был рассматриваемый,
                    // то теперь первый элемент списка - минимальный
                    first = best;
                // Рассматриваемый элемент теперь минимальный
                curr = best;
            }
            // Продвигаемся дальше по списку
            prev = curr;
            curr = curr.next;
        }
    }

    // Метод расчета суммы вкладов
    public decimal CalcTotalBalance()
    {
        // Инициализируем сумму
        decimal total = 0;
        // Начинаем просмотр списка с первого элемента
        AccountRingListItem curr = first;
        if (curr != null)
            // Если список не пуст, то запускаем цикл
            do
            {
                // Добавляем сумму вклада к общей сумме
                total += curr.data.balance;
                // Переходим к следующему вкладу
                curr = curr.next;
                // Продолжаем так в цикле, пока снова не придем к первому вкладу
            } while (curr != first);
        return total;
    }

    // Метод вывода списка на экран
    public void Print()
    {
        // Начинаем просмотр списка с первого элемента
        AccountRingListItem curr = first;
        if (curr != null)
            // Если список не пуст, то запускаем цикл
            do
            {
                // Выводим данные вклада на экран с форматированием
                Console.WriteLine(
                    string.Format("{0,20}|{1,12}|{2,12:0.00}", curr.data.owner,
                    curr.data.number, curr.data.balance)
                );
                // Переходим к следующему вкладу
```

```
                    curr = curr.next;
                    // Продолжаем так в цикле, пока снова не придем к первому вкладу
                } while (curr != first);
            }
        }
    }
```

Результаты

В ходе выполнения работы была написана программа, которая осуществляет чтение списка вкладчиков из файла, выводит их на экран в алфавитном порядке, а также подсчитывает и выводит общую сумму вкладов. Для упорядочивания вкладов был выбран алгоритм сортировки выбором, применяющийся для массивов, и адаптирован для использования с однонаправленным кольцевым списком.

Выводы

Преимуществами использованной в работе структуры данных «однонаправленный кольцевой список» перед аналогичным решением с использованием массива являются:

- для размещения в оперативной памяти не требуется непрерывного свободного блока;
- при сортировке выбором во время операции перемещения минимального элемента не требуется проводить операцию обмена, что придает сортировке свойство устойчивости (элементы с одинаковыми ключами следуют в отсортированной структуре в том же порядке, что и в исходной).

Недостатки структуры:

- для размещения требуется больше оперативной памяти из-за необходимости хранения указателя на следующий элемент, а также накладных расходов менеджера памяти;
- время доступа к элементам может быть выше на оборудовании с системой кэширования непрерывных блоков памяти;
- неэффективно использовать алгоритмы сортировки, требующие константную вычислительную сложность доступа к элементу по индексу, например, быструю сортировку.