ПЯВУ. Лекция 7.

Основы программирования.

А.М. Задорожный

Контрольные вопросы

1. Что такое массив?
2. Как объявить массив?
3. Как узнать количество элементов массива?
4. Как нумеруются элементы массива?
5. Где применяется (в C#) и что означает слово **break**? **continue**?

Содержание

Алгоритм подсчета количества

Задача. Подсчитать количество элементов массива целых, делящихся на 3.

```
//Подготовим пример исходных данных
Random r = new Random();
int [] a = new int[10];

for(int i = 0; i < a.Length; i++)
      a[i] = r.Next(25);

for(int i = 0; i < a.Length; i++)
      Console.Write("{0}, ", a[i]);
Console.WriteLine();
```

Подсчет количества

```
int n = 0;
for(int i = 0; i < a.Length; i++)
    if(a[i] % 3 == 0)
        n++;

//Вывод результата для контроля
Console.WriteLine(“N = {0}”, n);
```

Среднее арифметическое

Задача. Найти среднее арифметическое элементов массива действительных чисел.

```
//Подготовим пример исходных данных
Random r = new Random();
double [] a = new double[10];

for(int i = 0; i < a.Length; i++)
    a[i] = 10*r.NextDouble() - 5;

for(int i = 0; i < a.Length; i++)
    Console.Write("{0:0.##}, ", a[i]);
Console.WriteLine();
```

Среднее арифметическое

```
double sum = 0, avrg;
for(int i = 0; i < a.Length; i++)
    sum += a[i];
avrg = sum/a.Length;

//Вывод результата для контроля
Console.WriteLine("Среднее = {0:0.##}", avrg);
```

Пояснение форматов вывода

Формат	3	0.3333	3.3333	33.3333
#.##	3	.33	3.33	33.33
0.##	3	0.33	3.33	33.33
0.00	3.00	0.33	3.33	33.33

Алгоритм подсчета количества слов в строке

Будем считать, что строка имеет вид:

__XXXXXXX_____XXXXXXX__...

Т.е., что состоит из слов и разделителей.

Что считать, начала слов или окончания?

__**X**XXXXXX__ __XXXXXX**X**__

Удобнее начала.

Алгоритм подсчета количества слов.
Флаг состояния

Введем булевскую переменную, которая будет определять, находимся ли мы в слове или вне слова:

bool outOfWord = **true**;

0. Занулим счетчик слов n.
1. Цикл по всем символам строки c_i
   1. Если c_i – разделитель, то outOfWord = true;
   2. Иначе, если outOfWord истино (нашли начало слова), то outOfWord = false и увеличить счетчик слов.
2. Конец цикла.

Композиция алгоритмов

- Известные алгоритмы могут комбинироваться в совместный алгоритм для решения новой задачи.

Задача. Подсчитать количество чисел массива больших среднего.

1. Вычислить среднее (avrg).
2. Подсчитать количество чисел, больших avrg.

Поиск заданного элемента массива

Задача. Найти номер первого элемента массива удовлетворяющего заданному условию, или установить, что такого элемента нет.

Задача уже решалась, как часть задачи “Поиск условного наибольшего”

Поиск заданного элемента массива

Задача. Найти номер первого элемента массива делящегося на 3 или установить, что такого элемента нет.

```
int iFound = -1;
for(int i = 0; i < a.Length; i++)
    if(a[i] % 3 == 0)
    {
        iFound = i;
        break;
    }

// Здесь в iFound содержится номер искомого элемента массива // или -1, если такого элемента нет
```

Упражнения

1. Какие из изученных алгоритмов являются композицией других алгоритмов?
   a) Подсчет количество слов в строке.
   b) Поиск наибольшего/наименьшего элемента массива.
   c) Поиск элемента массива, удовлетворяющего заданному условию.
   d) Подсчет количества элементов, удовлетворяющих заданному условию.
   e) Поиск условно наибольшего/наименьшего элемента массива.
   f) Поиск элементов массива больших среднего.

2. Какие из задач являются конкретизациями алгоритмов, приведенных выше?
   a) Подсчитать количество пробелов в строке.
   b) Найти первый наибольший среди элементов массива, которые делятся на 3.
3. Поясните, что означают коды форматирования # и 0.

Сортировка массива

- **Сортировка** – расположение элементов массива в порядке возрастания или убывания.

- **Сортировка** – важная задача обработки информации. (Проще искать в отсортированном массиве: телефонный справочник, список группы и т.п.)

- Имеются более 100 алгоритмов сортировки.

Алгоритм сортировки пузырьком

Однократный проход

```
for(int i = 0; i < a.Length - 1; i++)
        if(a[i]>a[i + 1])
        {
                double    x = a[i];
                a[i] = a[i + 1];
                a[i + 1] = x;
        }
```

1. Самый большой элемент массива окажется последним. Т.е. он будет находиться на своем месте.

2. Останется отсортировать только первые a.Length-1 элементов.

Иллюстрация прохода

3
6
1
4
5
9
2
8
7

Иллюстрация прохода

3
6
1
4
5
9
8
2
7

Иллюстрация прохода

3
6
1
4
9
5
8
2
7

Иллюстрация прохода

3
6
1
9
4
5
8
2
7

Иллюстрация прохода

3
6
9
1
4
5
2
8
7

Иллюстрация прохода

3
9
6
1
4
5
2
8
7

Иллюстрация прохода

9

3

6

1

4

5

2

8

7

Сортировка пузырьком

Всего однократный проход нужно применить N-1 раз, где N – длина массива. (Массив из 1 элемента всегда отсортирован!)

```
for(int j = 1; j < a.Length; j++)
      for(int i = 0; i < a.Length - 1; i++)
       if(a[i]>a[i + 1])
       {
              double    x = a[i];
              a[i] = a[i + 1];
              a[i + 1] = x;
       }
```

Вложенные циклы

Сортировка пузырьком. Оптимизация.

- Количество итераций **внутреннего** цикла не зависит от номера итерации **внешнего** цикла.
- Было установлено, что неотсортированная часть массива после каждого прохода сокращается на 1 элемент.

- Рассмотрим:

```
for(int j = 1; j < a.Length; j++)
      for(int i = 0; i < a.Length - j; i++)
       if(a[i]>a[i + 1])
       {
              double      x = a[i];
              a[i] = a[i + 1];
              a[i + 1] = x;
       }
```

Улучшенная сортировка пузырьком

- Количество итераций оригинального алгоритма не зависит от исходного порядка элементов массива. (Даже если массив изначально упорядочен!)

Улучшенный пузырек:

```
for(int j = 1; j < a.Length; j++)
{
      bool sorted = true;
      for(int i = 0; i < a.Length - j; i++)
       if(a[i]>a[i + 1])
       {
              double       x = a[i];
              a[i] = a[i + 1];
              a[i + 1] = x;
               sorted = false;
       }
      if(sorted)
       break;
}
```

Анализ сортировки пузырьком

- Количество итераций = (N-1)+(N-2)+…+1
  = N*(N-1)/2
- Количество сравнений = N*(N-1)/2
- Среднее количество перестановок
  = N*(N-1)/4

Циклические перестановки массива

1, 2, 3, 4, 5

По часовой стрелке:

5, 1, 2, 3, 4

Против часовой стрелки:

2, 3, 4, 5, 1

Алгоритм циклической перестановки

Против часовой стрелки:

```
int x = a[0];
for(int i = 1; i < a.Length; i++)
      a[i-1]=a[i];
a[a.Length-1] = x;
```

По часовой стрелке:

```
int x = a[a.Length-1];
for(int i = a.Length-1; i > 0; i--)
      a[i]=a[i-1];
a[0] = x;
```

Контрольные вопросы

1. Что означает термин “Сортировка” в информатике?
2. Зачем применяется сортировка?
3. От чего и как зависит количество операций при сортировке пузырьком?
4. Как мог бы выглядеть алгоритм решения задачи: “Найти 5 наибольших элементов массива чисел”?
5. Как можно доработать алгоритм циклической перестановки, что бы перестановка осуществлялась на N позиций?

Задачи

1. Разработайте алгоритм инвертирования массива целых? (Инвертирование – расположение элементов в обратном порядке, например, 1,2,3=>3,2,1)

2. Разработайте алгоритм “перетасовки” массива (как колоды карт). Например:1,2,3,4,5,6,7 после перетасовки должен с равной вероятностью принимать значение любой перестановки.

Поясните, почему любой элемент массива может в конце оказаться на любом месте.

3. Разработайте алгоритм, который для произвольной перестановки чисел 1,2,3,4,5,6,7 подсчитывает количество инверсий.

Говорят, что перестановка содержит инверсию, если больший элемент расположен левее меньшего. Например, перестановка 1,2,3,4,7, 5,6 содержит 2 инверсии: 7-5 и 7-6.

Замечание. Четность числа инверсий определяет четность перестановки!