МАССИВЫ. ТИПОВЫЕ АЛГОРИТМЫ ОБРАБОТКИ МАССИВОВ

Массив – это структура данных, содержащая несколько значений одного типа, обозначаемая одним именем. Доступ к элементам массива осуществляется по индексу. Изменяя индексы, можно переходить от одного элемента массива к другому и таким образом обрабатывать единообразно большие наборы данных, используя циклы. Индексация массивов в C# начинается с нуля. Массив может быть одномерным, многомерным, вложенным. Здесь будут рассмотрены только одномерные и двухмерные массивы. Одномерный массив представляет собой линейную структуру. Положение элемента определяется одним индексом. Двухмерный массив можно представить себе как таблицу. Положение элемента определяется двумя индексами: номером строки и номером столбца. Начнем рассмотрение с одномерных массивов.

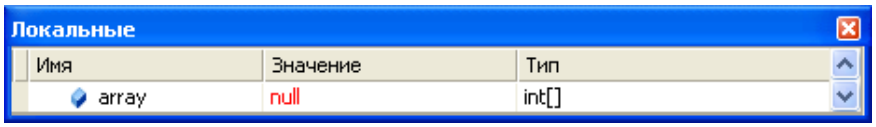
Объявление и инициализация массива

Все массивы должны быть объявлены и инициализированы перед их использованием. При объявлении массива нужно указать тип элементов массива (элементы массива могут быть любых типов), далее следуют пустые квадратные скобки и имя массива (переменная массива). Например,

int[] array;

(Пустые квадратные скобки указывают на то, что переменная `array` является массивом.)

В соответствии с этим объявлением под переменную массива `array` выделяется ячейка памяти для хранения ссылки на одномерный массив элементов типа `int`. Переменной `array` присваивается значение `null`.

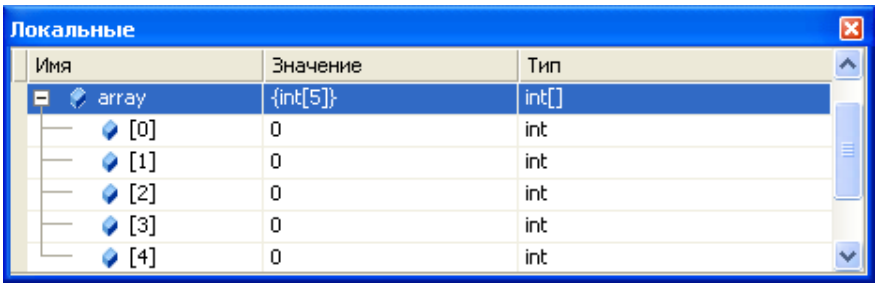


Далее объявленной переменной `array` можно присвоить массив конкретного размера, используя оператор `new`:

```
int[] array;
array = new int[5];
```

Оператор `new` служит для создания массива (выделения блока памяти для размещения элементов массива и передачи в переменную `array` адреса этого блока) и инициализации элементов массива со значением по умолчанию. По умолчанию все элементы числовых массивов инициализируются значением 0.

Индексация элементов массива начинается с 0. Массив размера `n` содержит элементы с индексами от 0 до `n–1`.Описанный выше массив array будет содержать элементы с array `[0]` по array `[4]`.



Инициализацию массива можно выполнить и при объявлении переменной массива:

```
int[] array = new int[5];
```

Действие этого оператора полностью аналогично приведенным выше двум операторам.

При объявлении массива можно явно задать значения элементов. В этом случае спецификация ранга (указание количества элементов массива) необязательна, поскольку она уже предоставлена по числу элементов в списке инициализации. Например,

```
int[] array1 = new int[] { 1, 3, 5, 7, 9 };
```

Доступ к отдельному элементу массива осуществляется заданием индекса, указываемого в квадратных скобках после имени массива. Например,

```
int[] array = new int[5];
…
array[3] = 8;
…
```

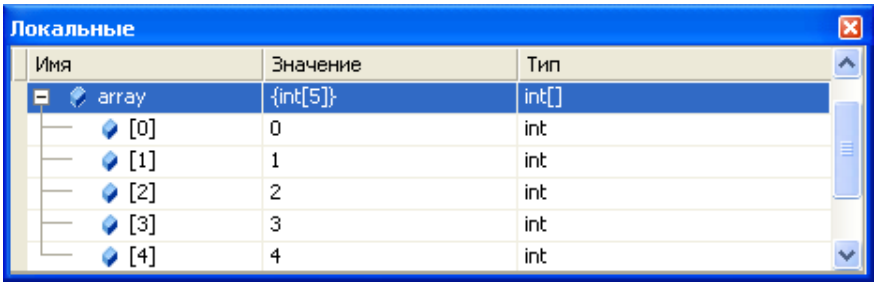
Здесь описан массив `array`, содержащий 5 элементов от `array[0]` до `array[4]`; 4-му элементу этого массива присваивается значение 8.

Заполнение массива

Использование переменной в качестве индекса позволяет организовывать циклы для задания элементов массива или их обработки с использованием индекса в качестве управляющей переменной цикла. Например,

```
int[] array = new int[5];
for (int i = 0; i < 5; i++)
{
 array[i] = i;
}
```

Здесь каждому элементу массива `array` присваивается значение, равное его индексу.



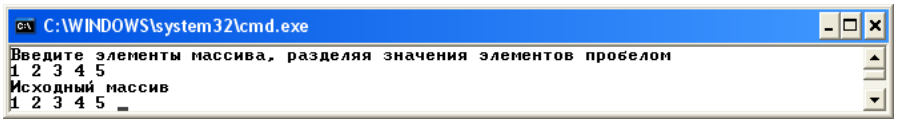
Ввод массива, т.е. заполнение элементов массива заданными значениями, можно осуществлять в цикле. Например, для ввода массива `a` размера 5 с элементами типа `double` необходимо использовать цикл.

```
double[] a = new double[5];
string s;
for (int i = 0; i < 5; i++)
{
   s = Console.ReadLine();
   a[i] = double.Parse(s);
}
```

Здесь при каждом вызове статического метода `ReadLine()` класса `Console` нужно вводить один элемент массива, после набора на клавиатуре значения очередного элемента нужно нажимать клавишу [Enter]. Введенный элемент помещается методом `ReadLine()` в переменную s типа `string`. Для того чтобы получить число из строкового представления числа необходимо вызвать статический метод `Parse`, который осуществляет разбор строки и возвращает экземпляр типа, который находится в строке. Если в строке храниться число типа `double`, то надо вызвать статический **метод** `Parse` **типа** `double`.

Можно набирать значения элементов массива на клавиатуре в одной строке, разделяя их пробелами (или любым другим разделителем). Например,

```
int[] a = new int[5];
Console.WriteLine("Введите элементы массива, разделяя значения элементов пробелом");
string s = Console.ReadLine();
string[] c = s.Split(" ");
for (int i = 0; i < 5; i = i + 1)
{
   a[i] = int.Parse(c[i]);
}
Console.WriteLine("Исходный массив");
for (int i = 0; i < 5; i++)
   Console.Write(a[i]+" ");
Console.ReadKey();
```



Здесь последовательно набираются на клавиатуре значения элементов массива, разделяемые пробелами (см. окно вывода). Введенная строка помещается в строковую переменную `s`. Далее производится **разбор этой строки с помощью метода** `Split` (см. гл. 6). Определяется положение первого пробела в строке `s` и символы, расположенные перед ним пересылаются в первый элемент строкового массива `c`. Далее определяется положение следующего пробела в строке `s`, и символы, расположенные между текущим и предыдущим пробелами, пересылаются в следующий элемент строкового массива `c` и до тех пор, пока не будет закончен разбор всей строки. В результате будет сформирован строковый массив `c`, каждый элемент которого содержит одно число в строковом представлении, Далее эти числа после преобразования по очереди пересылаются в массив `a`.

Вывод массива в строку

В строке присутствует пробел, чтобы выводимые значения зрительно отделялись друг от друга. Например,

```
Console.Write(a[i]+" ");
```

Можно также выводить пробел отдельным оператором после вывода очередного элемента массива. Например,

```
for (int i = 0; i < 5; i++)
{
   Console.Write(a[i]);
   Console.Write(" ");
}
```

При выводе массива необходимо обеспечить наглядность и удобство восприятия выводимых данных. Вывод одномерного массива целесообразно осуществлять в строку, отделяя элементы массива друг от друга пробелами. При этом используется оператор `Write`, после выполнения которого перевода строки не происходит (см. предыдущий пример).

Вывод одномерного массива в столбец

Вывод осуществляется с использованием оператора `WriteLine`, после выполнения которого каждый раз происходит переход на новую строку. Например,

```
for (int i = 0; i < 5; i++)
{
   Console.WriteLine(array[i]);
}
```

Вывод одномерного массива размера `n` по `m` элементов в строке

Код приведен для `n = 25, m = 5`:

```
const int n = 25;
int m = 5;
int[] array = new int[n];
for (int i = 0; i < n; i++)
{
   array[i] = i*i;
   Console.Write("{0,3:d} ", array[i]);
   if ((i+1) % m == 0) Console.WriteLine();
}
Console.WriteLine();
```

Если номер очередного выведенного элемента (`i+1`) кратен 5 (результат его деления нацело на 5 равен 0), то выполняется оператор `WriteLine` и происходит перевод строки.

Вычислить сумму элементов массива:

```
int[] a = new int[5]{ 2, 3, 4, 5, 7};
int s = 0;
for (int i = 0; i < 5; i++)
{
   s = s + a[i];
}
Console.WriteLine(s);
Console.ReadKey();
```

Объявляется массив целых чисел, состоящий из пяти элементов. Элементам массива присваиваются значения из списка инициализации. На каждом шаге к сумме добавляется очередной элемент массива.

Для обработки последовательно всех элементов массива язык C# предлагает вместо обычного `for` оператор `foreach`. Пользоваться им намного проще. Например, приведенный выше фрагмент будет выглядеть так:

```
int[] a = new int[5]{ 2, 3, 4, 5, 7};
int s = 0;
foreach (int x in a)
{
   s = s + x;
}
Console.WriteLine(s);
Console.ReadKey();
```

В скобках после ключевого слова `foreach` объявляем переменную типа `int` (того же типа, что и элементы массива), которой при каждой итерации будут последовательно присваиваться значения элементов массива и ее можно будет использовать в цикле вместо переменной с индексами. При использовании `foreach` отпадает необходимость в использовании управляющей переменной цикла (ее не нужно объявлять, изменять и проверять, не вышло ли значение индекса за допустимые пределы).

Вычислить среднее арифметическое положительных элементов массива размера 5:

```
double[] a = new double[5]{ 4.0, 3.0, 4.0, -5.0, -7.0};
double s= 0;
int m = 0;
for (int i = 0; i < 5; i++)
{
   if (a[i] > 0)
   {
      s = s + a[i];
      m += 1;//m = m + 1
   }
}
double sr = s / m;
Console.WriteLine(sr);
Console.ReadKey();
```

Здесь в переменной s накапливается сумма положительных элементов, а в переменной m – их количество. После выхода из цикла остается разделить сумму на количество.

С использованием оператора `foreach` код будет следующим:

```
double[] a = new double[5] { 4.0, 3.0, 5.0, -5.0, -7.0 };
double s = 0;
int m = 0;
foreach (double x in a)
{
   if (x > 0)
   {
     s = s + x;
     m += 1;//m = m + 1
   }
}
double sr = s / m;
Console.WriteLine(sr);
Console.ReadKey();
```

Найти сумму элементов, расположенных до первого отрицательного элемента массива

```
double[] a = new double[5] { 4.0, 3.0, 4.0, -5.0, -7.0 };
double s = 0;
foreach (double x in a)
{
   if (x > 0)
   {
      s = s + x;
 }

 else
 {
 break;
 }
}
Console.WriteLine(s);
Console.ReadKey();
```

Оператор `break` завершает цикл, если не будет выполнено условие `x > 0` (т.е. встретится первый отрицательный элемент). Произойдет выход из цикла и суммирование закончится.

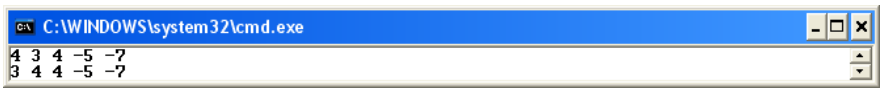
Возможен другой вариант программы:

```
double[] a = new double[5]{ 4.0, 3.0, 4.0, -5.0, -7.0};
double s = 0;
foreach(double x in a)
{
   if (x < 0)
   {
      break;
   }
   s = s + x;
}
Console.WriteLine(s);
Console.ReadKey();
```

***Перестановка элементов в векторе* (*вектор* в программировании – то же, что одномерный массив). В примере переставлены нулевой и первый.**

```
double[] a = new double[5] { 4.0, 3.0, 4.0, -5.0, -7.0 };
double p;
foreach (double x in a)
   Console.Write("{0} ", x);
Console.WriteLine();
p = a[0]; a[0] = a[1]; a[1] = p;
foreach (double x in a)
   Console.Write("{0} ", x);
Console.WriteLine();
Console.ReadKey();
```

При перестановке используется вспомогательная переменная (в данном случае `p`).

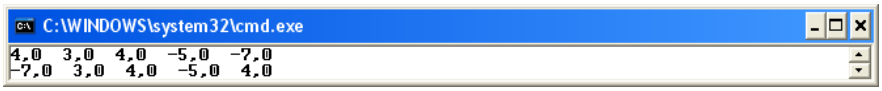


В общем виде перестановка `i`-го и `j`-го элементов осуществляется следующим образом:

```
p = a[i]; a[i] = a[j]; a[j] = p;
```

Последний отрицательный элемент массива поменять с первым элементом массива:

```
double[] a = new double[5] { 4.0, 3.0, 4.0, -5.0, -7.0 };
double p;
int i;
for (i = 0; i < 5; i++)
   Console.Write("{0:f1} ", a[i]);
Console.WriteLine();
for (i = 4; i >= 0; i--)
   if (a[i] < 0) break;
p = a[i]; a[i] = a[0]; a[0] = p;
for (i = 0; i < 5; i++)
   Console.Write("{0:f1} ", a[i]);
Console.WriteLine();
Console.ReadKey();
```



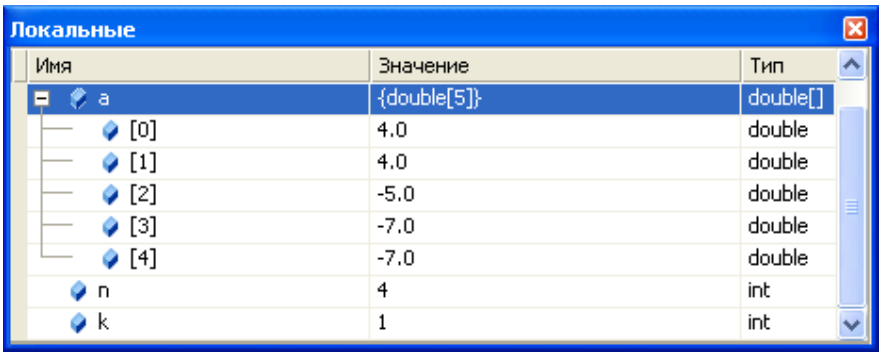
Здесь массив просматривается с конца (начиная с последнего элемента, с шагом 1), и первый встретившийся отрицательный элемент будет последним отрицательным элементом в массиве. Для того чтобы переменная цикла сохранила свое значение при принудительном выходе из цикла, ее надо объявить до использования оператора итерации `for`, иначе она будет локальной по отношению к циклу и потеряет свое значение вне цикла. Перестановка элементов выполняется с использованием вспомогательной переменной `p`. Предложить самостоятельно вариант кода с использованием оператора `foreach` вместо `for`.

Удалить элемент с заданным индексом из массива. Массив сжать.

Удалить элемент с заданным индексом k означает сдвинуть все следующие за ним элементы на одну позицию влево, т.е. выполнить операцию $a_i = a_{i+1}$ для $I = k, k + 1,\ldots, n - 1$:

```
double[] a = new double[5] { 4.0, 3.0, 4.0, -5.0, -7.0 };
for (int i = 0; i < 5; i++)
   Console.Write("{0:f1} ", a[i]);
Console.WriteLine();
int n = 5, k = 1;
n = n - 1;
for (int i = k; i < n; i++)
   a[i] = a[i + 1];
for (int i = 0; i < n; i++)
   Console.Write("{0:f1} ", a[i]);
Console.WriteLine();
Console.ReadKey();
```

Размер массива уменьшился на 1. При этом на месте последнего элемента исходного массива остается старое значение, но оно не будет нас интересовать, так как формально не входит в состав результирующего массива (см. приведенное ниже окно «Локальные», в котором отображаются локальные переменные текущего контекста, имена переменных, их значения и тип).

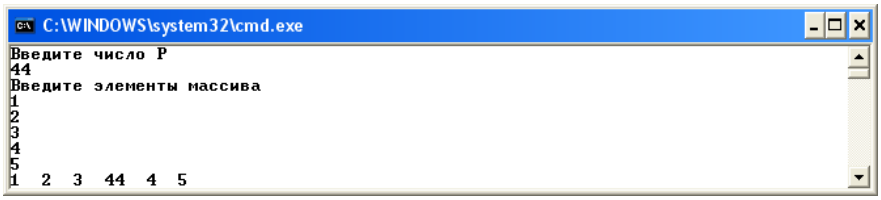


Включить заданное значение p после k-го элемента массива.

Размер массива при этом увеличится на 1. Это необходимо предусмотреть при описании массива, указав его размер на 1 больше исходного.

Чтобы не потерять элемент исходного массива, необходимо освободить место для нового значения, передвинув вправо на одну позицию все элементы, начиная с $(k + 1)$-го, т.е. выполнить операцию $a_{i+1} = a_i$ для $i = n, n - 1, \ldots, k + 1$. Необходимо также учитывать, что нумерация индексов массива начинается от нуля:

```
const int n = 6;
int[] a = new int[n];
Console.WriteLine("Введите число P");
string s = Console.ReadLine(); ;
int p = int.Parse(s);
Console.WriteLine("Введите элементы массива");
for (int i = 0; i < n - 1; i++)
{
   s = Console.ReadLine();
   a[i] = int.Parse(s);
}
int k = 2;
for (int i = n - 2; i >= k + 1; i--)
{
   a[i + 1] = a[i];
}
a[k + 1] = p;
for (int i = 0; i < n; i++)
   Console.Write("{0:d} ", a[i]);
Console.WriteLine();
Console.ReadKey();
```



Перемещать элементы нужно начиная с последнего. В противном случае элементы массива, расположенные после $(k + 1)$ элемента, будут иметь одинаковое значение, равное значению $(k + 1)$-го элемента. Убедитесь в этом самостоятельно.

Сформировать массив из элементов другого массива, удовлетворяющих заданному условию.

Пусть требуется переслать в массив *b* положительные элементы массива *a*. Сразу заметим, что индексы одинаковых по значению элементов массивов *a* и *b* не будут совпадать. Цикл организуем по индексам элементов массива *a*, индекс пересылаемого элемента в массиве *b* будем каждый раз увеличивать на 1:

```
double[] a = new double[5] { 4.0, -3.0, 4.0, -5.0, 7.0 };
double[] b = new double[5];
int k = 0;
foreach (double x in a)
{
   if (x > 0)
   {
      b[k] = x;
      k = k + 1;
   }
}
```

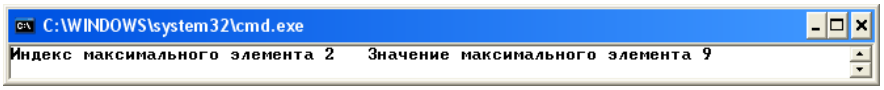
После выхода из цикла в переменной `k` будет содержаться количество элементов массива `b` (значение индекса последнего элемента будет при этом на 1 меньше) и для вывода массива `b` необходимо организовать следующий цикл

```
for (int i = 0; i < k; i++)
   Console.Write("{0:f1} ", b[i]);
Console.WriteLine();
Console.ReadKey();
```

9. *Найти максимальный элемент массива и его индекс*.

Сохраним значения первого элемента и его индекса в переменных `amax = a[0], imax = 0`. Далее будем просматривать остальные элементы массива последовательно, начиная со второго, сравнивать очередной элемент `a[i]` со значением, сохраненным в переменной amax и заменять значение `amax` и индекс `imax` , если очередной элемент оказался больше.

```
int[] a = new int[5] { 4, 3, 9, 5, 7 };
int amax = a[0];
int imax = 0;
for (int i = 1; i < 5; i++)
{
   if (a[i] > amax)
   {
      amax = a[i];
      imax = i;
   }
}
Console.WriteLine("Индекс максимального элемента {0:d} Значение максимального
элемента {1:d} ", imax, a[imax]);
Console.ReadKey();
```



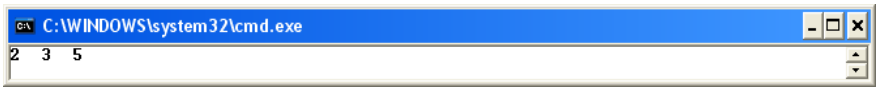
Поиск минимального элемента массива осуществляется аналогично с той лишь разницей, что в операторе `if` нужно использовать оператор отношения <, а не >.

Если в массиве имеется *несколько максимальных элементов*, у которых значения одинаковые, а индексы разные, то в переменной, в которой сохраняется индекс, будет сохранен индекс первого из них. Чтобы получить индекс последнего, необходимо в операторе `if` проверить условие:

```
if (a[i] >= amax).
```

Если нужно запомнить индексы всех максимальных элементов, то необходимо предусмотреть массив для хранения их индексов, например `im`, в котором будут запоминаться индексы одинаковых максимальных элементов. При изменении значения `amax` массив `im` начнет заполняться сначала. Если очередной элемент `a[i]` равен максимальному, то в следующий элемент массива `im` помещается текущий индекс `i`:

```
int[] a = new int[7] { 4, 7, 9, 9, 3, 9, 2 };
int[] im = new int[7];
int amax = a[0];
int k = 0;
for (int i = 0; i < 7; i++)
{
   if (a[i] > amax)
   {
      amax = a[i];
      k = 0;
      im[k] = i;
   }
   else if (a[i] == amax)
   {
      k = k + 1;
      im[k] = i;
   }
}
for (int i = 0; i <= k; i++)
   Console.Write("{0:d} ", im[i]);
Console.WriteLine();
Console.ReadKey();
```



После выхода из цикла количество элементов массива `im` находится в переменной `k`.

Упорядочить массив. Сортировка выбором

Требуется расположить элементы массива *a* в определенном порядке, например, по убыванию.

Рассмотрим алгоритм упорядочения, базирующийся на двух уже рассмотренных алгоритмах, а именно на алгоритмах поиска максимального элемента и перестановки элементов.

В цикле при $i = 0, 1, 2, \ldots, n - 2$ будем устанавливать на нужное место i-й элемент. Для этого найдем максимальный элемент и его индекс в части массива, начиная с i-го элемента, и далее поменяем местами максимальный элемент с i-м. После окончания цикла элементы массива будут расположены в порядке убывания, при этом на последнем n-м месте окажется самый маленький элемент:

```
const int n = 7;
int[] a = new int[n] { 4, 7, 9, 10, 3, 12, 2 };
for (int i = 0; i < n - 2; i++)
{
    int amax = a[i];
    int imax = i;
    for (int j = i + 1; j < n - 1; j++)
    {
        if (a[j] > amax)
        {
            amax = a[j];
            imax = j;
        }
    }
    a[imax] = a[i];
    a[i] = amax;
}
for (int i = 0; i < n; i++)
    Console.Write("{0:d}  ", a[i]);
Console.WriteLine();
Console.ReadKey();
```

Типовые алгоритмы 11 – 15 приводятся без соответствующих кодов. Рекомендуется разработать их самостоятельно, используя приведенные схемы алгоритмов, и проверить их на компьютере.

Поиск заданного элемента в упорядоченном массиве (бинарный поиск).

Поиск в упорядоченном массиве осуществляется существенно быстрее, чем в неупорядоченном. Поэтому часто целесообразно предварительно произвести упорядочение массива, особенно в случае необходимости многократного поиска и больших массивов.

Требуется в массиве *A* размером *n*, упорядоченном по возрастанию, определить наличие заданного элемента *X* и его индекс, если такой элемент найден.

Схема алгоритма

- *x* сравнивается со средним элементом массива.
- Индекс среднего элемента `i = (i1+i2)/2, где i1 = 0, i2 = n – 1.`
- Если $x = a(i)$, задача решена.
- Если $x<a(i)$, то поиск продолжается в левой половине: `i2 = i – 1`, `i1` не изменяется.
- Если $x>a(i)$, то поиск продолжается в правой половине: `i1 = i + 1, i2` не изменяется.
- Искомый элемент отсутствует в массиве, если выполнится условие `i2<i1`.

Реализовать алгоритм самостоятельно.

Объединение двух массивов с чередованием элементов.

Требуется объединить два массива одинакового размера $A = (a_0, a_1, \ldots, a_{n-1})$ и $B = (b_0, b_1, \ldots, b_{n-1})$ в один массив $C = (a_0, b_0, a_1, b_1, \ldots, a_{n-1}, b_{n-1})$.

Элементами массива С с четными индексами являются элементы массива А:

$$c_0 = a_0;\ c_2 = a_1;\ \ldots;\ c_{2i} = a_i,\ \ldots,$$

элементами с нечетными индексами – элементы массива В:

$$c_1 = b_0,\ c_3 = b_1 \ldots,\ \ldots\ ,\ c_{2i-1} = b_i, \ldots$$

Таким образом, требуется организовать цикл и выполнить операции

$$c_{2i} = a_i,\ c_{2i-1} = b_{i-1},\ \text{для } i = 0, 1, 2, , n - 1.$$

Если массивы имеют разные размеры, то больший массив обрезается по размеру меньшего и меньший массив и урезанный большой объединяются в соответствии с предложенным алгоритмом. Далее оставшиеся элементы большего массива пересылаются подряд.

13. *Объединение двух упорядоченных массивов в один с сохранением упорядоченности*.

Требуется объединить два упорядоченных по убыванию массива *A* размером *n* и *B* размера *m* в один массив *C* размером *n* + *m*, также упорядоченный.

Схема алгоритма

Начиная с первых элементов массивов *A* и *B*, сравниваем элементы `A[i]` и `B[j]`. В массив *C* пересылаем больший из них, например, `A[i]` (если выполняется условие `A[i]>=B[j]`).

- Далее продолжаем пересылать в массив *C* элементы массива *A* (индекс *j* при этом не меняется), пока очередной элемент массива *A* **не будет меньше** `B[j]` (условие `A[i] >=B[j]` не выполняется).
- Тогда **начинаем пересылать в массив *C* элементы массива *B*** (индекс *i* при этом не меняется), пока для какого-либо элемента массива *B* не будет снова выполнено условие (`A[i] >=B[j]`).
- Если один из массивов исчерпан (выполнено условие $i > n$ или $j > m$), то оставшиеся элементы другого массива пересылаются друг за другом без всяких проверок.

Реализовать алгоритм самостоятельно.

Инвертирование массива

- Требуется изменить порядок следования элементов массива размером n на обратный, т.е. поменять местами нулевой элемент массива с $(n-1)$-м, первый с $(n-2)$-м, i-й с $(n-(i+1))$-м. Последними нужно поменять местами средние элементы, т.е. $(n/2)$-й и $(n/2+1)$-й.

Циклический сдвиг

- Требуется переместить элементы массива A размера n на m позиций вправо. При этом m элементов из конца массива перемещаются в начало:

$A = (a_1, a_2, a_3, a_4, a_5)$ – исходный массив;

$A = (a_4, a_5, a_1, a_2, a_3)$ – после циклической перестановки на 2 позиции вправо.

Вариант 1. Используется вспомогательный массив для временного хранения m последних элементов массива A. Далее оставшиеся элементы с нулевого по $(n - m - 1)$-й смещаются вправо на m позиций. (Заметим, что перемещение нужно начинать с последнего из перемещаемых элементов, чтобы не испортить элементы массива A.) После этого в первые m элементов пересылаются элементы вспомогательного массива.

Вариант 2. Циклический сдвиг осуществляется с использованием одной вспомогательной переменной, в которую каждый раз пересылается последний элемент массива A, после чего все элементы сдвигаются вправо на 1 позицию (начиная с конца), и на место первого элемента помещается значение вспомогательной переменной. Эта процедура повторяется m раз.