ЛЕКЦИЯ № 3. Алгоритмы обработки одномерных массивов.

Цель лекции: *Знакомство с понятием массива. Приобретение навыков построения алгоритмов предназначенных для обработки одномерных массивов.*

6. Алгоритмы обработки массивов

Часто для работы с множеством однотипных данных (целочисленными значениями, строками, датами и т.п.) оказывается удобным использовать массивы. Например, можно создать массив для хранения списка студентов, обучающихся в одной группе. Вместо создания переменных для каждого студента, например `Студент1`, `Студент2` и т.д., достаточно создать один массив, где каждой фамилии из списка будет присвоен порядковый номер. Таким образом, можно дать следующее определение. *Массив* – структурированный тип данных, состоящий из фиксированного числа элементов одного типа.

Массив на рисунке 6.1 имеет 8 элементов, каждый элемент сохраняет число вещественного типа. Элементы в массиве пронумерованы от 1 до 8. Такого рода массив, представляющий собой просто список данных одного и того же типа, называют *простым* или *одномерным* массивом. Для доступа к данным, хранящимся в определенном элементе массива, необходимо указать имя массива и порядковый номер этого элемента, называемый *индексом*.

12.1	0.13	-1,5	0	21.9	-3.7	5.0	121.7
1-й элемент массива	2-й элемент массива	3-й элемент массива	4-й элемент массива	5-й элемент массива	6-й элемент массива	7-й элемент массива	8-й элемент массива

Рис. 6.1 Одномерный числовой массив

Если возникает необходимость хранения данных в виде таблиц, в формате строк и столбцов, то необходимо использовать *многомерные* массивы. На рисунке 6.2 приведен пример массива, состоящего из четырех строк и четырех столбцов. Это *двумерный* массив. Строки в нем можно считать первым измерением, а столбцы вторым. Для доступа к данным, хранящимся в этом массиве, необходимо указать имя массива и *два индекса*, первый должен соответствовать номеру строки, а второй номеру столбца в которых хранится необходимый элемент.

		Номера столбцов			
		1	2	3	4
Номера строк	1	3.5	7.8	1.3	0.6
	2	-1.4	0.3	0	12.1
	3	-5.7	-0.78	5.0	6.9
	4	45.1	124.0	-24.7	0.96

Рис. 6.2 Двумерный числовой массив

6.1. Ввод-вывод элементов одномерного массива

При вводе массива необходимо последовательно вводить 1-й, 2-й, 3-й и т.д. элементы массива, аналогичным образом поступить и при выводе. Следовательно, необходимо организовать цикл.

Блок-схемы алгоритмов ввода элементов массива изображены на рис. 6.3-6.4.

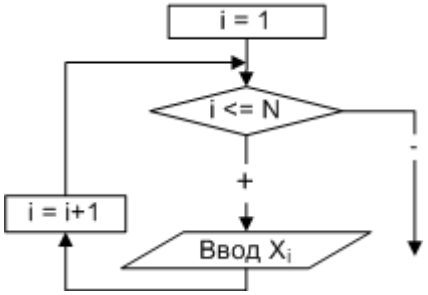


Рис 6.3 Алгоритм ввода массива с использованием цикла с предусловием

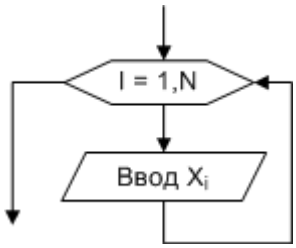


Рис. 6.4. Алгоритм ввода массива с использованием безусловного цикла

Как видно, безусловный цикл удобно использовать для обработки всего массива, и в дальнейшем при выполнении таких операций будем применять именно его. Вывод массива организуется аналогично вводу.

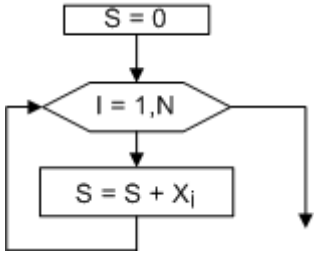
Рассмотрим несколько примеров обработки массивов. Алгоритмы, с помощью которых обрабатывают одномерные массивы, похожи на обработку последовательностей (вычисление суммы, произведения, поиск элементов по определенному признаку, выборки и т. д.). Отличие заключается в том, что в массиве одновременно доступны все его компоненты, поэтому становится возможной, например, сортировка его элементов и другие, более сложные преобразования.

6.2. Вычисление суммы элементов массива

Дан массив `X`, состоящий из `n` элементов. Найти сумму элементов этого массива. Процесс накапливания суммы элементов массива достаточно прост и практически ничем не отличается от суммирования значений некоторой числовой последовательности. Переменной `S` присваивается значение равное нулю, затем последовательно суммируются элементы массива `X`. Блок-схема алгоритма расчета суммы приведена на рис. 6.5.

6.3. Вычисление произведения элементов массива

Дан массив `X`, состоящий из `n` элементов. Найти произведение элементов этого массива. Решение этой задачи сводится к тому, что значение переменной `P`, в которую предварительно была записана единица, последовательно умножается на значение `i`–го элемента массива. Блок-схема алгоритма приведена на рис. 6.6.



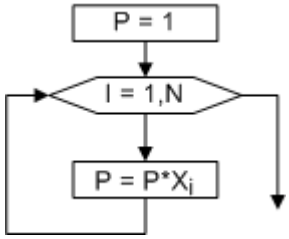


Рис. 6.5. Алгоритм вычисления суммы элементов массива

Рис. 6.6. Вычисление произведения элементов массива

6.4. Поиска максимального элемента в массиве и его номера

Дан массив X, состоящий из n элементов. Найти максимальный элемент массива и номер, под которым он хранится в массиве.

Алгоритм решения задачи следующий. Пусть в переменной с именем Max хранится значение максимального элемента массива, а в переменной с именем Nmax – его номер. Предположим, что первый элемент массива является максимальным, и запишем его в переменную Max**,** а в Nmax занесем его номер, то есть – 1. Затем все элементы, начиная со второго, сравниваем в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, то записываем его в переменную Max**,** а в переменную Nmax – текущее значение индекса i. Процесс определения максимального элемента в массиве приведен в таблице 6.1 и изображен при помощи блок-схемы на рис. 6.7.

Таблица 6.1. Определение максимального элемента и его номера в массиве

Номера элементов	1	2	3	4	5	6	7
Исходный массив	4	7	3	8	9	2	5
Значение переменной **Max**	4	7	7	8	9	9	9
Значение переменной **Nmax**	1	2	2	4	5	5	5

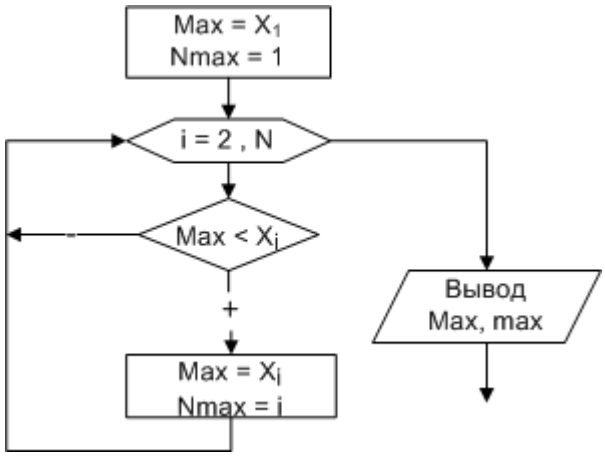


Рис. 6.7. Поиск максимального элемента и его номера в массиве

Совет. *Алгоритм поиска минимального элемента в массиве будет отличаться от приведенного выше лишь тем, что в условном блоке знак поменяется с > на <.*

6.5. Сортировка элементов в массиве

Сортировка представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений. Например, массив X из n элементов будет отсортирован в порядке возрастания значений его элементов, если $X_1 \le X_2 \le \ldots \le X_n$, и в порядке убывания, если $X_1 \ge X_2 \ge \ldots \ge X_n$.

Существует большое количество алгоритмов сортировки, но все они базируются на трех основных:

- сортировка обменом;

- сортировка выбором;
- сортировка вставкой.

Представим, что нам необходимо разложить по порядку карты в колоде. Для сортировки карт *обменом* можно разложить карты на столе лицевой стороной вверх и менять местами те карты, которые расположены в неправильном порядке, делая это до тех пор, пока колода карт не станет упорядоченной.

Для сортировки *выбором* из разложенных на столе карт выбирают самую младшую (старшую) карту и держат ее в руках. Затем из оставшихся карт вновь выбрать наименьшую (наибольшую) по значению карту и помещают ее позади той карты, которая была выбрана первой. Этот процесс повторяется до тех пор, пока вся колода не окажется в руках. Поскольку каждый раз выбирается наименьшая (наибольшая) по значению карта из оставшихся на столе карт, по завершению такого процесса карты будут отсортированы по возрастанию (убыванию).

Для сортировки *вставкой* из колоды берут две карты и располагают их в необходимом порядке по отношению друг к другу. Каждая следующая карта, взятая из колоды, должна быть установлена на соответствующее место по отношению к уже упорядоченным картам.

Итак, решим следующую задачу. Задан массив `Y` из `n` целых чисел. Расположить элементы массива в порядке возрастания их значений.

6.5.1. Сортировка методом «пузырька»

Сортировка пузырьковым методом является наиболее известной. Ее популярность объясняется запоминающимся названием, которое происходит из-за подобия процессу движения пузырьков в резервуаре с водой, когда каждый пузырек находит свой собственный уровень, и простотой алгоритма.

Сортировка методом «пузырька» использует метод обменной сортировки и основана на выполнении в цикле операций сравнения и при необходимости обмена соседних элементов. Рассмотрим алгоритм пузырьковой сортировки более подробно.

Сравним первый элемент массива со вторым, если первый окажется больше второго, то поменяем их местами. Те же действия выполним для второго и третьего, третьего и четвертого, `i`–го и (`i+1`)–го, (`n−1`)–го и ~~n~~–го элементов. В результате этих действий самый большой элемент станет на последнее (`n`-е) место. Теперь повторим данный алгоритм сначала, но последний (`n`-й) элемент, рассматривать не будем, так как он уже занял свое место. После проведения данной операции самый большой элемент оставшегося массива станет на (`n−1`)-е место. Так повторяем до тех пор, пока не упорядочим весь массив.

В табл.6.2 подробно расписан процесс упорядочивания элементов в массиве. Нетрудно заметить, что для преобразования массива, состоящего из ~~n~~ элементов, необходимо просмотреть его `n−1` раз, каждый раз уменьшая диапазон просмотра на один элемент. Блок–схема описанного алгоритма приведена на рис. 6.8. Обратите внимание на то, что для перестановки элементов (блок 4) используется буферная переменная `b`, в которой временно хранится значение элемента, подлежащего замене.

Таблица 6.2. Процесс упорядочивания элементов в массиве по возрастанию

Номер элемента	**1**	**2**	**3**	**4**	**5**
Исходный массив	7	3	5	4	2
Первый просмотр	3	5	4	2	7
Второй просмотр	3	4	2	5	7
Третий просмотр	3	2	4	5	7
Четвертый просмотр	2	3	4	5	7

Совет. *Для перестановки элементов в массиве по убыванию их значений необходимо при сравнении элементов массива заменить знак > на <.*

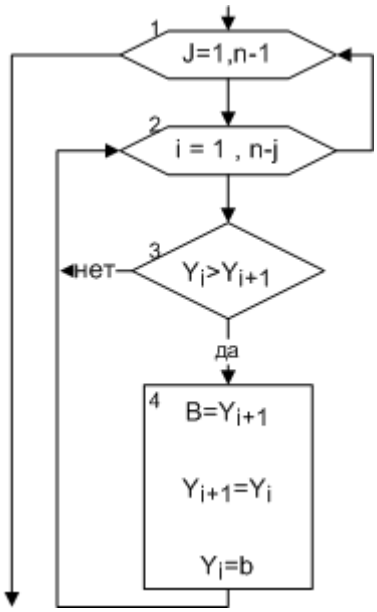


Рис. 6.8. Сортировка массива пузырьковым методом

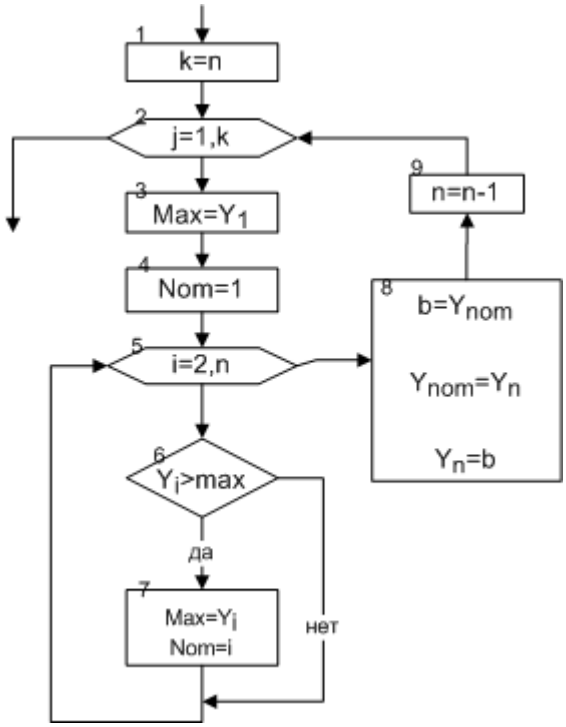


Рис. 6.9. Сортировка массива выбором наибольшего элемента

6.5.2. Сортировка выбором

Алгоритм *сортировки выбором* приведен в виде блок-схемы на рис. 6.9. Найдем в массиве самый большой элемент (блоки 3–7) и поменяем его местами с последним элементом (блок 8). Повторим алгоритм поиска максимального элемента, уменьшив количество просматриваемых элементов на единицу (блок 9), и поменяем его местами с предпоследним элементом (блоки 3–7). Описанную выше операцию поиска проводим до полного упорядочивания элементов в массиве. Так как в блоке 9 происходит изменение переменной ѣ, то в начале алгоритма ее значение необходимо сохранить (блок 1).

Совет. *Для упорядочивания массива по убыванию необходимо перемещать минимальный элемент*.

6.5.3. Сортировка вставкой

Сортировка вставкой заключается в том, что сначала упорядочиваются два элемента массива. Затем делается вставка третьего элемента в соответствующее место по отношению к первым двум элементам. Четвертый элемент помещают в список из уже упорядоченных трех элементов. Этот процесс повторяется до тех пор, пока все элементы не будут упорядочены.

Прежде чем приступить к составлению блок–схемы рассмотрим следующий пример. Пусть известно, что в массиве из восьми элементов первые шесть уже упорядочены, а седьмой элемент нужно вставить между вторым и четвертым. Сохраним седьмой элемент во вспомогательной переменной, так как показано на рисунке 6.10, а на его место запишем шестой. Далее пятый переместим на место шестого, четвертый на место пятого, а третий на место четвертого, тем самым, выполнив сдвиг элементов массива на одну позицию вправо. Записав содержимое вспомогательной переменной в третью позицию, достигнем нужного результата.

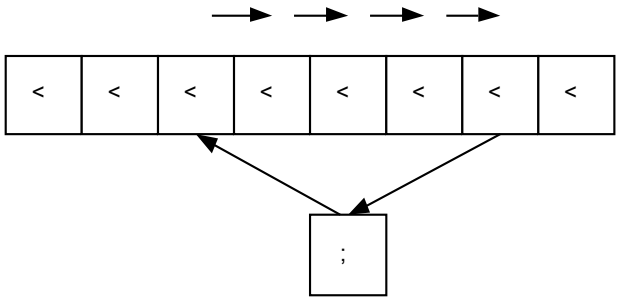


Рис. 6.10. Процесс вставки элемента в массив

Составим блок–схему алгоритма (рис. 6.11), учитывая, что возможно описанные выше действия придется выполнить неоднократно.

Организуем цикл для просмотра всех элементов массива, начиная со второго (блок 4). Сохраним значение текущего `i`–го элемента во вспомогательной переменной `X`, так как оно может быть потеряно при сдвиге элементов (блок 5) и присвоим переменной `j` значение индекса предыдущего (`i-1`)–го элемента массива (блок 6). Далее движемся по массиву влево в поисках элемента меньшего, чем текущий и пока он не найден сдвигаем элементы вправо на одну позицию. Для этого организуем цикл (блок 7), который прекратиться, как только будет найден элемент меньше текущего. Если такого элемента в массиве не найдется и переменная `j` станет равной нулю, то это будет означать, что достигнута левая

граница массива, и текущий элемент необходимо установить в первую позицию. Смещение элементов массива вправо на одну позицию выполняется в блоке 8, а изменение счетчика `j` в блоке 9. Блок 10 выполняет вставку текущего элемента в соответствующую позицию.

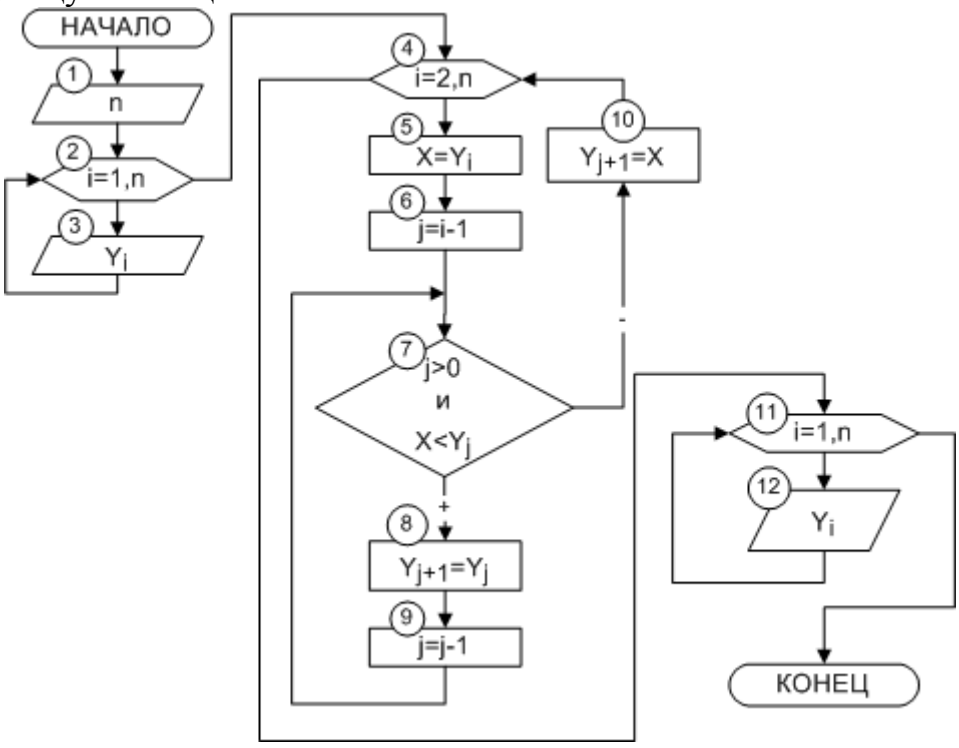


Рис. 6.11. Сортировка массива вставкой

6.6. Удаление элемента из массива

Необходимо удалить из массива `X`, состоящего из ~~n~~ элементов, `m`–й по номеру элемент. Для этого достаточно записать элемент (`m+1`) на место элемента `m`, (`m+2`)– на место (`m+1`) и т.д., `n`– на место (`n−1`) и при дальнейшей работе с этим массивом использовать `n−1` элемент (рис. 6.12).

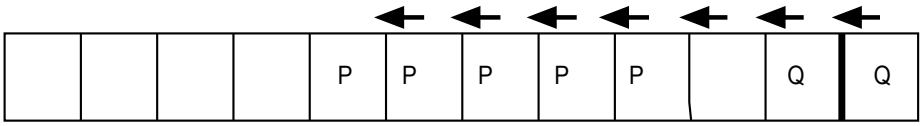


Рис. 6.12. Процесс удаления элемента из массива

Алгоритм удаления из массива `X` размерностью `n` элемента с номером `m` приведен на рис. 6.13.

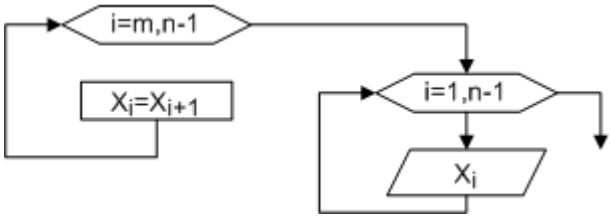


Рис. 6.13 Алгоритм удаления элемента из массива

6.7. Примеры алгоритмов обработки массивов

ПРИМЕР 6.1. Дан массив A состоящий из k целых положительных чисел. Записать все четные по значению элементы массива A в массив B.

Решение задачи заключается в следующем. Последовательно перебираются элементы массива A. Если среди них находятся четные, то они записываются в массив B. На рисунке 6.14 видно, что первый четный элемент хранится в массиве A под номером три, второй и третий под номерами пять и шесть соответственно, а четвертый под номером восемь. В массиве B этим элементам присваиваются совершенно иные номера. Поэтому для их формирования необходимо определить дополнительную переменную. В блок–схеме приведенной на рисунке 6.15 роль такой переменной выполняет переменная m. Операция, выполняемая в блоке 2, означает, что в массиве может не быть искомых элементов. Если же условие в блоке 5 выполняется, то переменная m увеличивается на единицу, а значение элемента массива A записывается в массив B под номером m (блок 6). Условный блок 7 необходим для того, чтобы проверить выполнилось ли хотя бы раз условие поиска (блок 5).

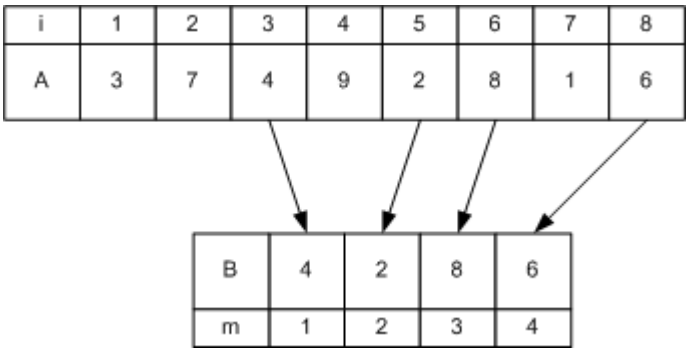


Рис. 6.14 Процесс формирование массива B из элементов массива A

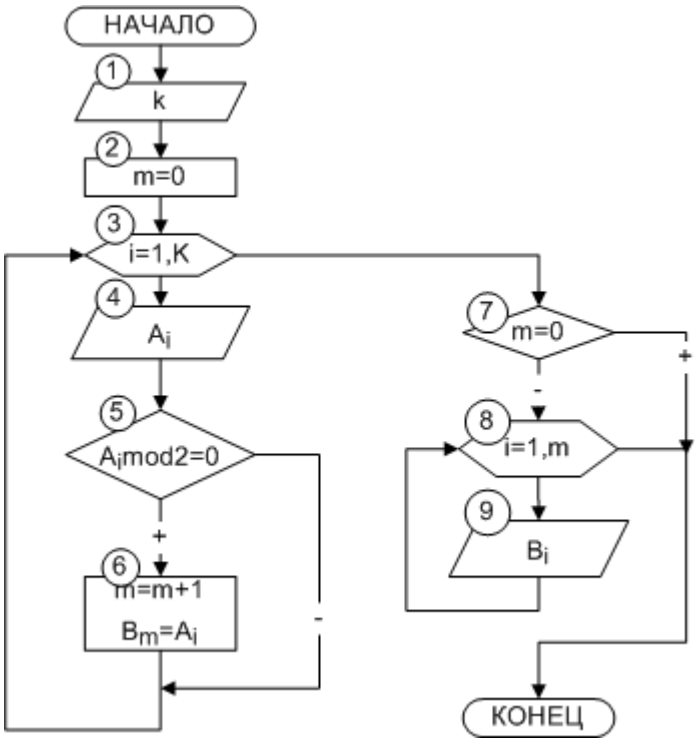


Рис. 6.15 Формирование массива B из соответствующих элементов массива A

ПРИМЕР 6.2. Задан массив `y` из `n` целых чисел. Сформировать массив `z` таким образом, чтобы в начале шли отрицательные элементы массива `y`, затем положительные и, наконец, нулевые. Блок-схема представлена на рис. 6.16.

ПРИМЕР 6.3. Переписать элементы массива `x` в обратном порядке. Блок-схема представлена на рис. 6.17. Алгоритм состоит в следующем: меняем местами `1`-й и `n`-й элементы, затем `2`-й и `n-1`-й элементы, и т.д. до середины массива (элемент с номером `i` следует обменять с элементом `n+1-i`).

ПРИМЕР 6.4. Задан массив из `n` элементов. Сформировать массивы номеров положительных и отрицательных элементов. Блок-схема представлена на рис. 6.18.

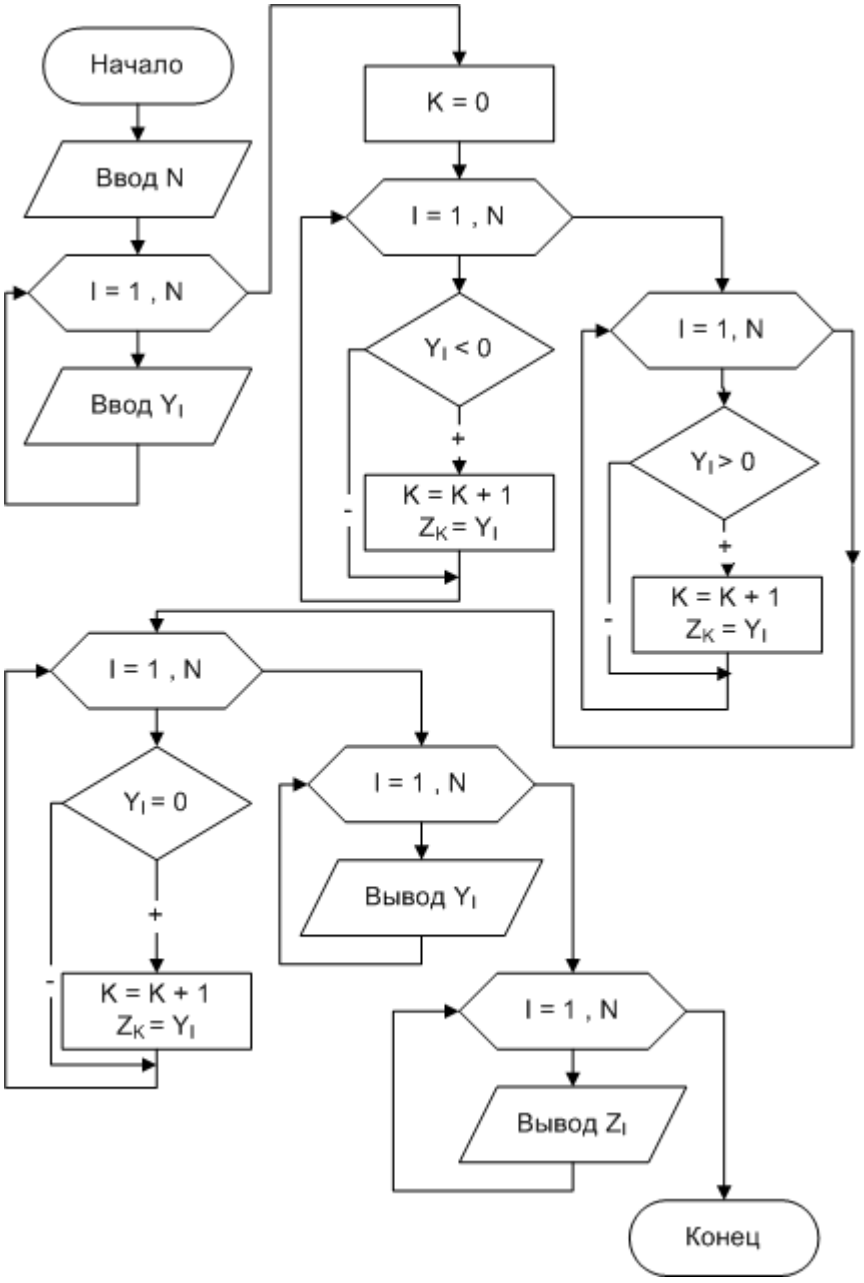


Рис. 6.16. Алгоритм примера 6.2

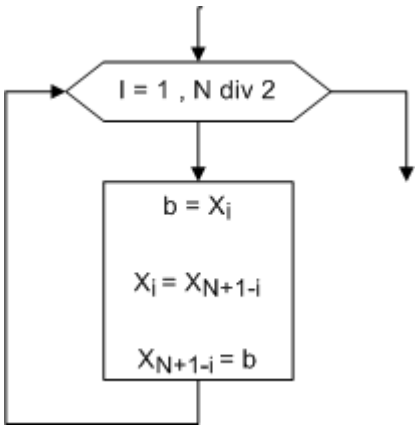


Рис. 6.17. Фрагмент блок–схемы к примеру 6.3

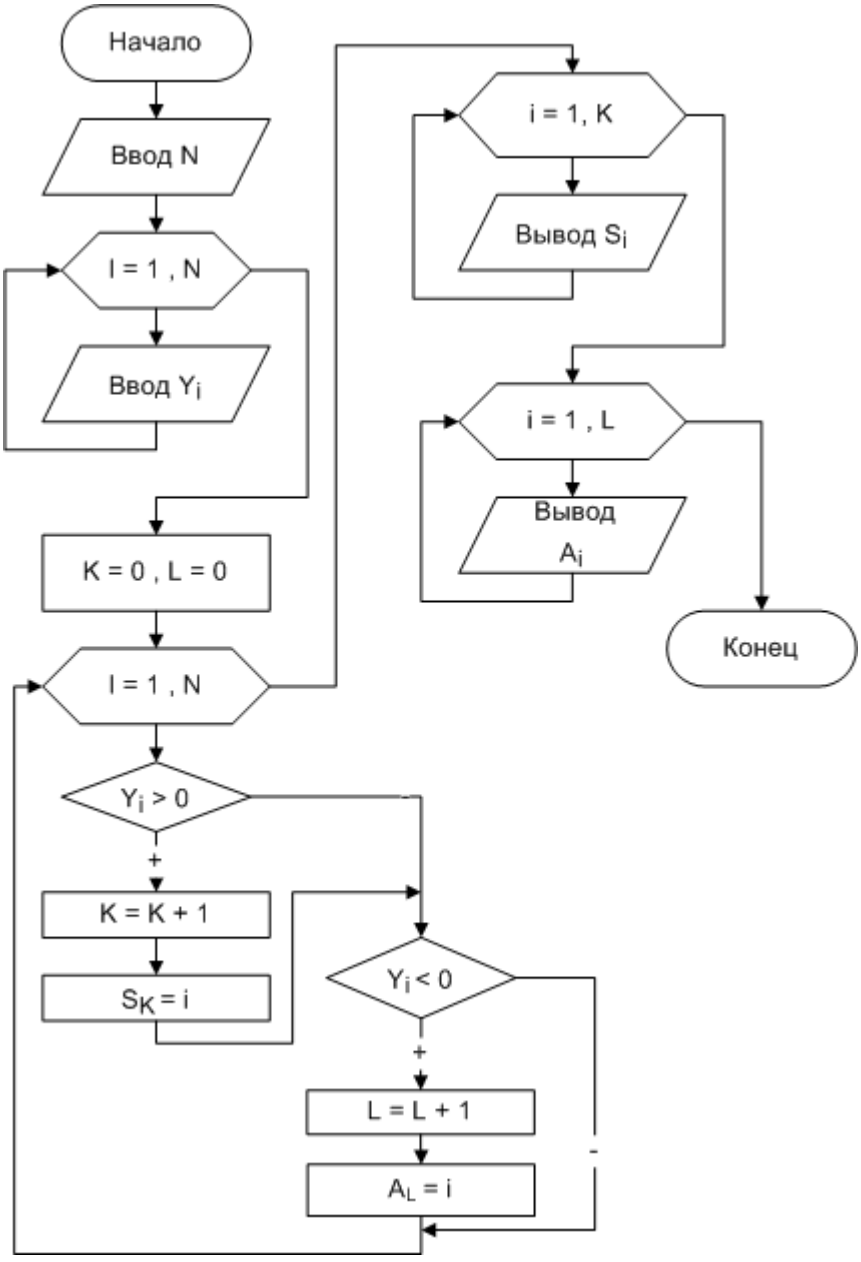


Рис. 6.18. Алгоритм примера 6.4

ПРИМЕР 6.5. Удалить из массива `X`, состоящего из ~~n~~ элементов, первые четыре нулевых элемента.

Вначале количество кулевых элементов равно нулю (`k=0`). Последовательно перебираем все элементы массива. Если встречается нулевой элемент, то

количество нулевых элементов увеличиваем на 1 (`k=k+1`). Если количество нулевых элементов меньше или равно 4, но удаляем очередной нулевой элемент, иначе аварийно выходим из цикла (встретился пятый нулевой элемент и дальнейшая обработка массива бесполезна). Блок-схема представлена на рис. 6.19.

ПРИМЕР 6.6. Массив целых чисел `C` состоит из `N` элементов, найти сумму простых чисел, входящих в него.

Идея алгоритма состоит в следующем. Сначала сумма равна 0. Последовательно перебираем все элементы, если очередной элемент простой, то добавляем его к сумме. Блок–схема алгоритма изображена на рис.6.20.

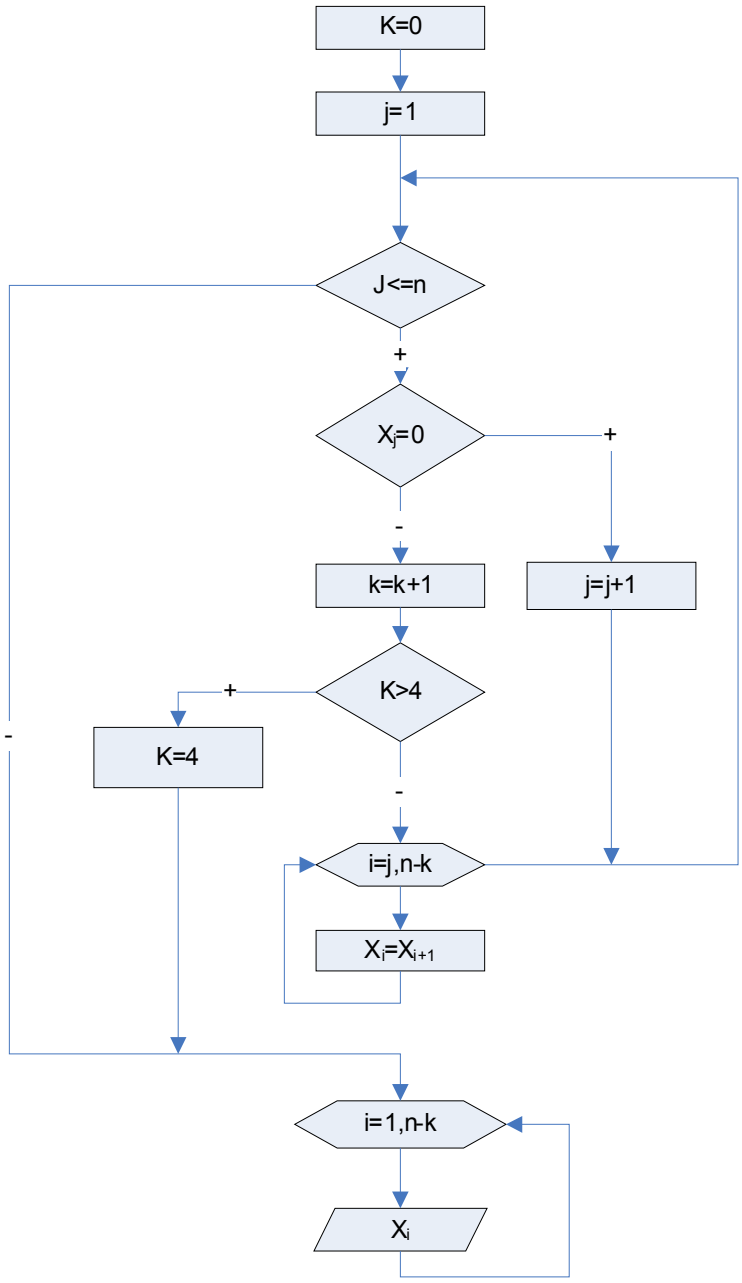


Рис. 6.19. Алгоритм примера 6.5

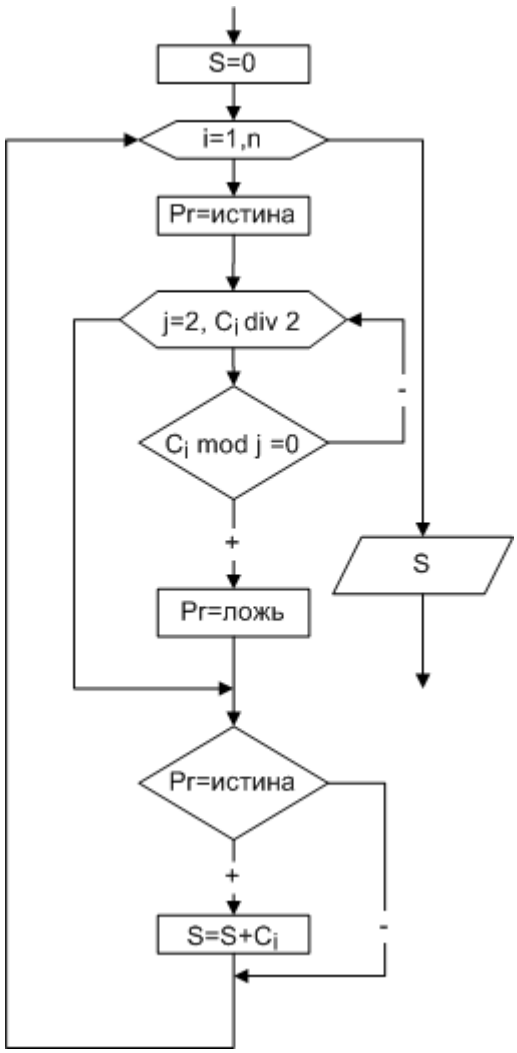


Рис. 6.20. Алгоритм примера 6.6

ПРИМЕР 6.7. Определить есть ли в заданном массиве серии элементов, состоящих из знакочередующихся чисел (рис. 6.21). Если есть, то вывести на экран количество таких серий.

Рис. 6.21. Рисунок к примеру 6.7

На рис. 6.22 изображена блок-схема решения поставленной задачи.

Здесь переменная `k` – количество элементов, попадающих в серию, `kol` – количество знакочередующихся серий в массиве.

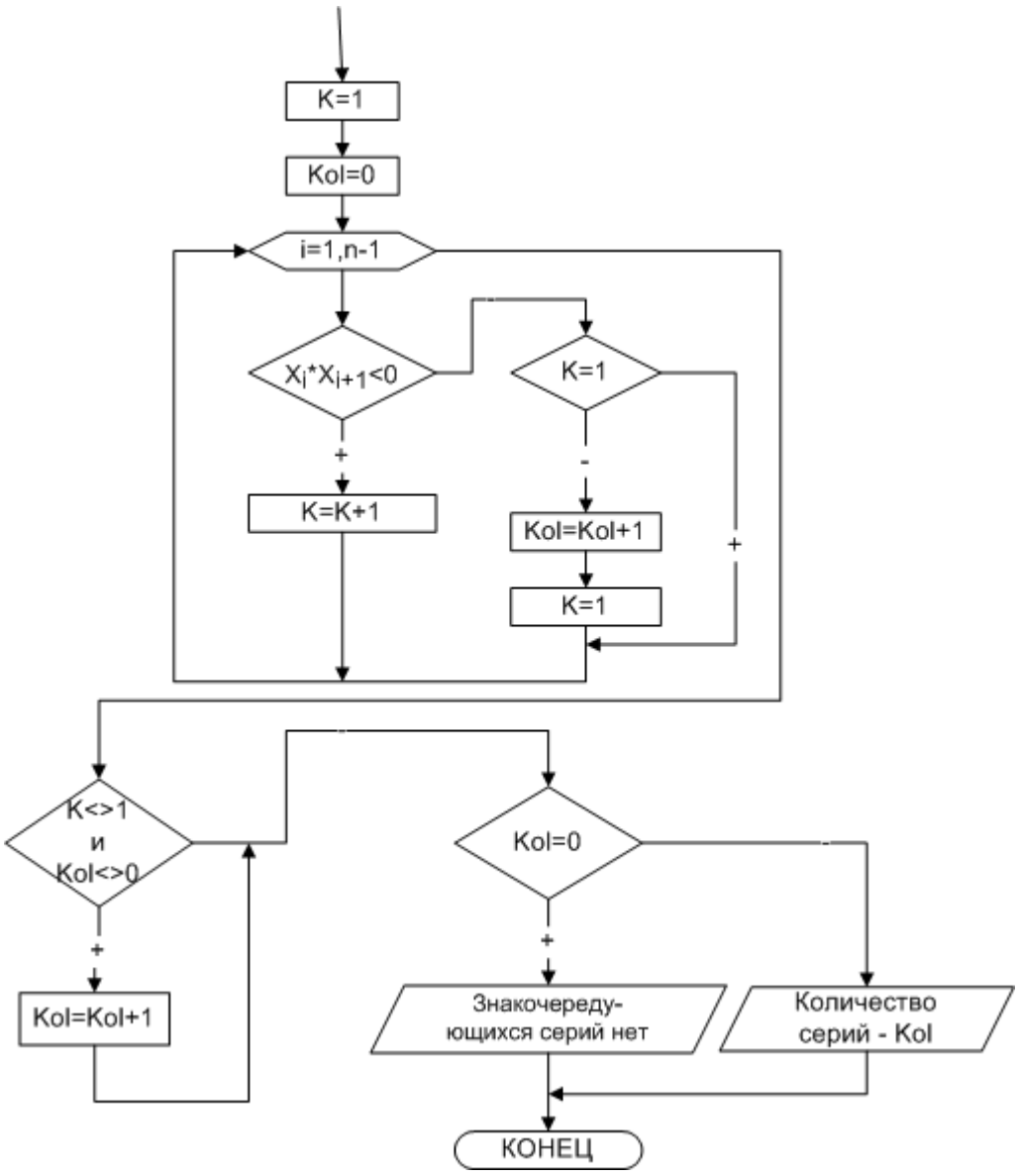


Рис. 6.22. Блок–схема решения задачи примера 6.7