Указатели и массивы

Понятие указателя

Указатель – это переменная, в которой хранится адрес другой переменной или участка памяти. Назначение — хранение (указание/получение) адресов.

Указатель-константа – адрес участка памяти, который не изменяется в процессе работы программы.

```
int a[MAX_SIZE]
```

`a` — указатель-константа, т. к. место в памяти резервируется при описании массива и больше не изменяется.

Понятие указателя

Указатели-переменные явно объявляются в списке переменных, но перед их именем ставится знак *. Указатель всегда указывает на переменную того типа, для которого он был объявлен.

```
double F;
double *pF;

F = 1.23;
pF = &F;
printf("Address: %p, value: %lf\n", pF, *pF);
```

Пример lect-03-01.c

pF — адрес памяти, по которому записана переменная F (& - получение адреса).
pF — значение, записанное по адресу pF (в коде «» - «разыменование указателя», «разадресация», косвенная адресация).

Смысл «*» при описании переменной и в коде — разный!

Операции с указателями (адресная арифметика)

Указатель — целое число в шестнадцатеричной системе (адрес памяти).

Что можно делать?

- Инициализация (можно при описании: `int a=5, p=&a;` **но не нужно!**)
- Присваивание
- Сравнение
- Добавление числа
- Вычитание
- Инкремент и декремент

(Примеры lect-03-02.c, lect-03-03.c. Для указателей типа double все значения измеряются в `sizeof(double)`)

Сравнение корректно только для указателей одного типа.

NULL — указатель, который никуда не указывает (пустой).

Операции с указателями (адресная арифметика)

Что нельзя делать (недопустимые операции):

- Сложение двух указателей
- Вычитание двух указателей на ***различные*** объекты
- Сложение указателей с вещественными числами
- Вычитание вещественных чисел из указателей
- Умножение указателей
- Деление указателей
- Поразрядные (битовые) операции

Операции с указателями (адресная арифметика)

Инициализация указателей
Пусть есть `int *p`;

1. Можно дать p значение корректного адреса памяти

```
p=(int*)0x7ffd653b09ac;
```

Для этого нужно знать, какие адреса могут быть корректными.

2. Если описаны переменные, то для них уже память выделена. Тогда указателю можно присвоить адрес переменной или другой указатель

```
float f, *pF, *pG;
pF=&f;
pG=pF; /* pF и pG содержат один и тот же адрес */
```

3. Запросить выделение памяти для указателя

```
float *pF;
pF=malloc(sizeof(float));
```

Операции с указателями (адресная арифметика)

Косвенная адресация
Используется, чтобы получить значение, хранящееся по адресу, на который ссылается указатель, или записать значение в конкретный адрес.
См. пример lect-03-02a.c

Преобразование типа.
Указатель на объект одного типа может быть преобразован в указатель на другой тип. При этом следует учитывать, что объект, адресуемый преобразованным указателем, будет интерпретироваться по-другому.
См. пример lect-03-03a.c

Преобразование типа применяется при использовании функций выделения памяти (*будут разбираться далее*), которые имеют тип **`void`** (неопределенный, пустой).

Операции с указателями (адресная арифметика)

Сложение и вычитание числа

В операции могут участвовать указатель и величина типа **int**. При этом результатом операции будет указатель на исходный тип, а его значение будет на указанное число элементов больше или меньше исходного.

```
int *ptr1, *ptr2, a[10];
int i=2;
ptr1=a+(i+4);   /*  равно адресу элемента a[6] */
ptr2=ptr1-i;    /*  равно адресу элемента a[4] */
```

Вычитание указателей

В операции могут участвовать два указателя на один и тот же тип. Результат такой операции имеет тип **int** и равен числу элементов исходного типа между уменьшаемым и вычитаемым, причем если первый адрес младше, то результат имеет отрицательное значение.

```
int *ptr1, *ptr2, a[10];
int i;
ptr1=a+4;
ptr2=a+9;
i=ptr1-ptr2;  /*   равно -5 */
i=ptr2-ptr1;  /*   равно 5  */
```

Операции с указателями (адресная арифметика)

Инкремент и декремент (**++** и **--**)

Значение указателя увеличивается или уменьшается на длину типа, на который ссылается используемый указатель.

```
int *ptr, a[10];
ptr=&a[5];
ptr++;      /* равно адресу элемента a[6] */
ptr--;      /* равно адресу элемента a[5] */
```

Сравнение указателей

К значениям двух указателей на одинаковые типы применимы операции отношения ==, !=, <, <=, >, >= при этом значения указателей рассматриваются просто как целые числа, а результат сравнения равен 0 (ложь) или 1 (истина).

```
int *ptr1, *ptr2, a[10];
ptr1=a+5;
ptr2=a+7;
if (ptr1>ptr2) a[3]=4;
```

Значение ptr1 меньше значения ptr2 и поэтому оператор a[3]=4 не будет выполнен.

Операции с указателями (адресная арифметика)

Определение размера
Для определения размера указателя можно использовать операцию размер в виде **`sizeof(<указатель>)`** – пример lect-03-04.c.

Индексация
Указатель может индексироваться применением к нему операции индексации, обозначаемой в Си квадратными скобками [ ]. Индексация указателя имеет вид **`<указатель>[<индекс>]`**, где <индекс> записывается целочисленным выражением.

Возвращаемым значением операции индексации является данное, находящееся по адресу, смещенному в бóльшую или меньшую сторону относительно адреса, содержащегося в указателе в момент применения операции. Этот адрес определяется так: **`(адрес в указателе) + (значение <индекс>) * sizeof(<тип>)`**, где <тип> – это тип указателя.

Из этого адреса извлекается или в этот адрес посылается значение, тип которого интерпретируется в соответствии с типом указателя (пример lect-03-05.c).

Понятие массива

Массив – это переменная, которая является совокупностью компонентов одного типа. Доступ к компонентам (элементам) осуществляется по номерам (индексам).

Элементы массива объединены общим именем (именем массива). Если требуется обратиться к определенному элементу массива, то достаточно указать имя массива и индекс элемента (например, `a[3]`).

В математике есть примеры массивов – векторы и матрицы.

В Си массив как переменная не существует!

Виды массивов

Одномерные
Многомерные

- Массивы чисел
- Массивы символов
- *Массивы указателей*

Одномерный массив

m1[0]	m1[1]	…	m1[n-2]	m1[n-1]

n элементов

Двумерный массив

n x k элементов

m2[0][0]	m2[0][1]	…	m2[0][n-2]	m2[0][n-1]
m2[1][0]	m2[1][1]	…	m2[1][n-2]	m2[1][n-1]
…	…	…	…	…
m2[k-2][0]	m2[k-2][1]	…	m2[k-2][n-2]	m2[k-2][n-1]
m2[k-1][0]	m2[k-1][1]	…	m2[k-1][n-2]	m2[k-1][n-1]

Размерность массива — количество индексов, которые необходимо задать одновременно для доступа к элементу массива.

Одномерные массивы в Си

int a[8]; – объявление массива

int a[8] = { 1, 2, 3, 4, 5, 6, 7, 8 }; – инициализация массива при объявлении

int a[] = { 1, 2, 3, 4, 5, 6, 7, 8 }; – размер можно не указывать

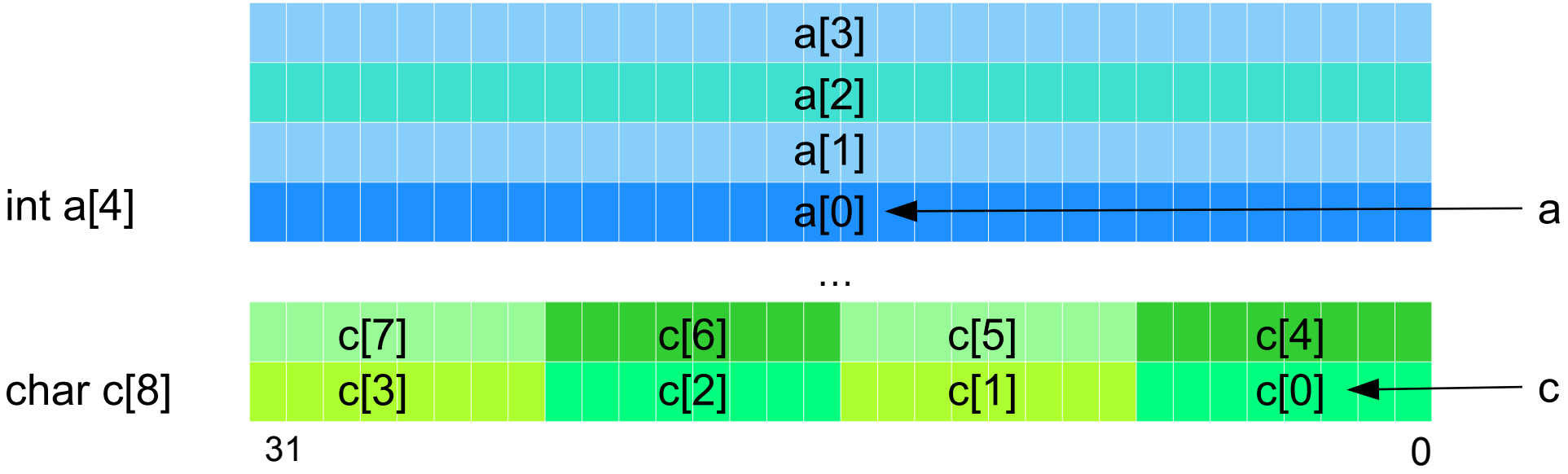
int a[8] = { 1, 2, 3, 4 }; – последние 4 элемента =0

Свойства одномерного массива:

- один и тот же тип данных
- все элементы в памяти располагаются друг за другом
- индекс первого элемента - 0
- имя массива определяет адрес начала массива (адрес первого элемента — постоянное значение и не меняется в процессе работы программы, ***имя массива является указателем-константой на первый элемент***)
- объем памяти для хранения массива = `n*sizeof(<type>)`
- ввод и вывод — в цикле.

См. примеры: lect-03-06.c, lect-03.07.c

Одномерные массивы в Си



В разных операционных системах направление заполнения — разное.

Одномерные массивы в Си

Размер массива — количество элементов

Размерность массива — количество индексов (для одномерного массива — 1 индекс).

Всегда нужно резервировать место для хранения элементов массива.

`int a[100];` /* резервируется память на 100 элементов */

`int n;` /* реальное количество элементов массива (с которыми работаем) */

`#define` позволяет задать переменную, содержащую максимально возможное количество элементов массива для резервирования памяти (пример lect-03-08.c).

При работе с одномерными массивами нужна служебная переменная — индекс элемента массива (традиционные имена: `i, j, k, l`).

Одномерные массивы и указатели

`int a[100];`

`&a[0]` → `a`

`&a[1]` → `a+1`

`&a[2]` → `a+2` и т. п.

Имя массива (указатель-константа) определяет ячейку памяти, в которой размещается первый элемент массива.

Добавление целого числа к имени массива приводит к сдвигу на указанное количество элементов.

Если `c` – массив `char`, то `(c+2)–c` → `2` байта (`2*sizeof(char)`)

Если `a` – массив `int`, то `(a+2)-a` → `8` байтов (`2*sizeof(int)`)

Двумерные массивы в Си

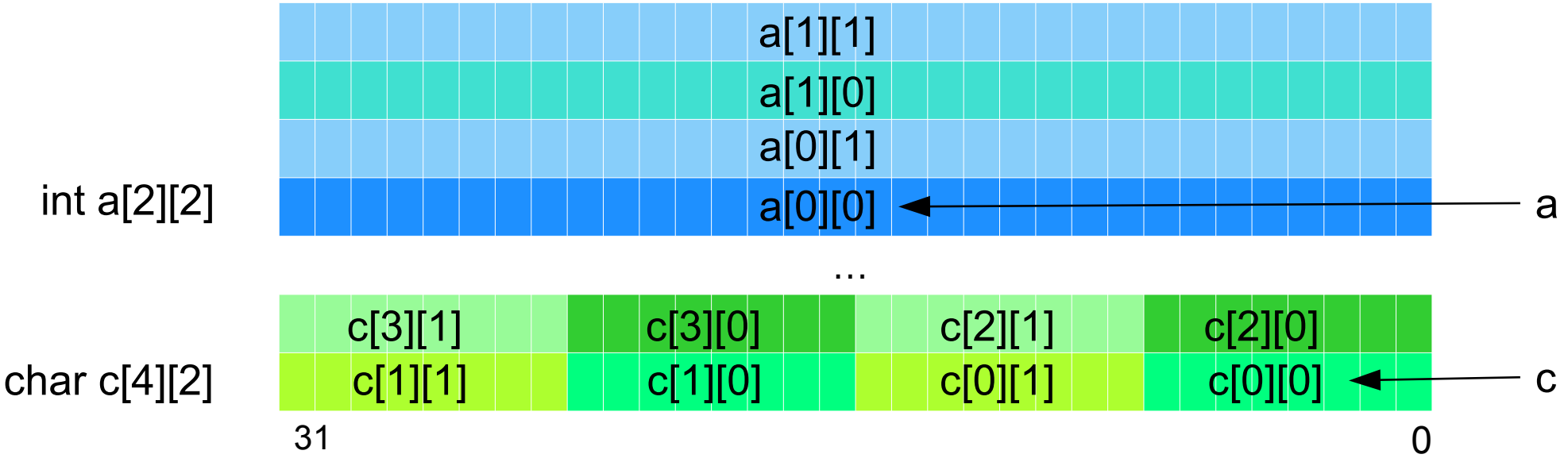
int b[8][6]; – объявление массива

Свойства двумерного массива:

- один и тот же тип данных
- все элементы в памяти располагаются друг за другом по строкам (сначала — все элементы первой строки, затем элементы 2-й строки и т. д.)
- первый индекс — номер строки, второй — номер столбца
- индекс первого элемента — [0][0]
- имя массива определяет адрес первого элемента первой строки (**имя двумерного массива является указателем-константой на массив указателей-констант**)
- объем памяти, требуемый для массива = `n*k*sizeof(<type>)` (`n` строк `k` столбцов)
- ввод и вывод — во вложенных циклах. Самый правый индекс меняется быстрее всего (самый «внутренний» цикл).

Пример: lect-03-09.c

Двумерные массивы в Си



В разных операционных системах направление заполнения — разное.

Двумерные массивы в Си

Инициализация при описании:

```
int a2[4][2]= {
              {1,2},
              {3,4},
              {5,6},
              {7,8}
              };

int a2[][2] = … /* вариант */
```

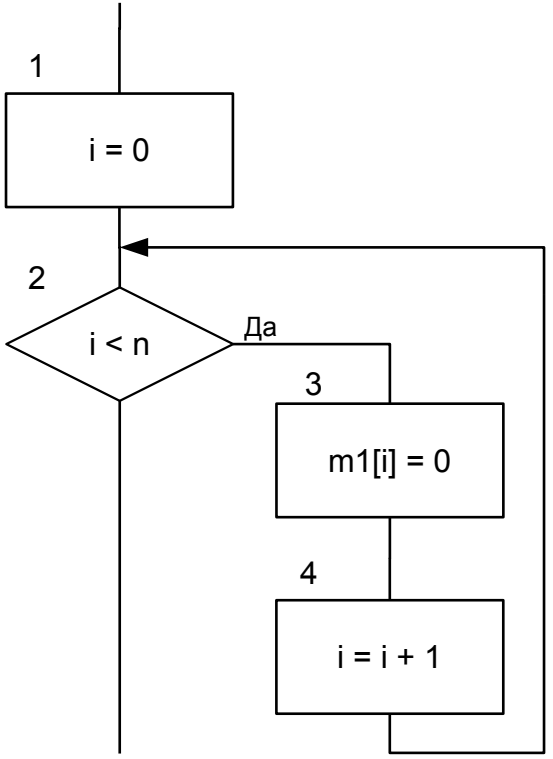
Типовые действия с массивами

- Инициализация (установка значений элементов)
- Перестановка элементов (изменение порядка)
- Вычислительные действия с элементами массива
- Поиск элементов массива (частный случай — поиск максимального и минимального элементов)
- Копирование элементов массива
- Сортировка массива

Типовые действия с массивами

Инициализация массива

Все элементы массива получают некоторые исходные значения (например, 0) — примеры lect-03-10.c, lect-03-11.c (*for()* и *while()*)



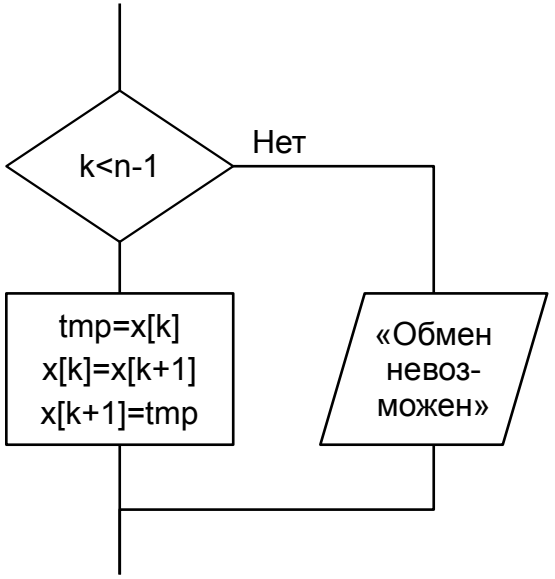
Типовые действия с массивами

Перестановка элементов массива:

Элемент массива с номером `k` можно поменять местами со следующим или предыдущим элементом.
Если заданный элемент массива `x[k]` является последним, то выполнить обмен со следующим элементом невозможно, поскольку последующий элемент отсутствует.

При обмене с предыдущим элементом обмен невозможен, если `k=0` (элемент — первый).

См. пример lect-03-12.c



Типовые действия с массивами

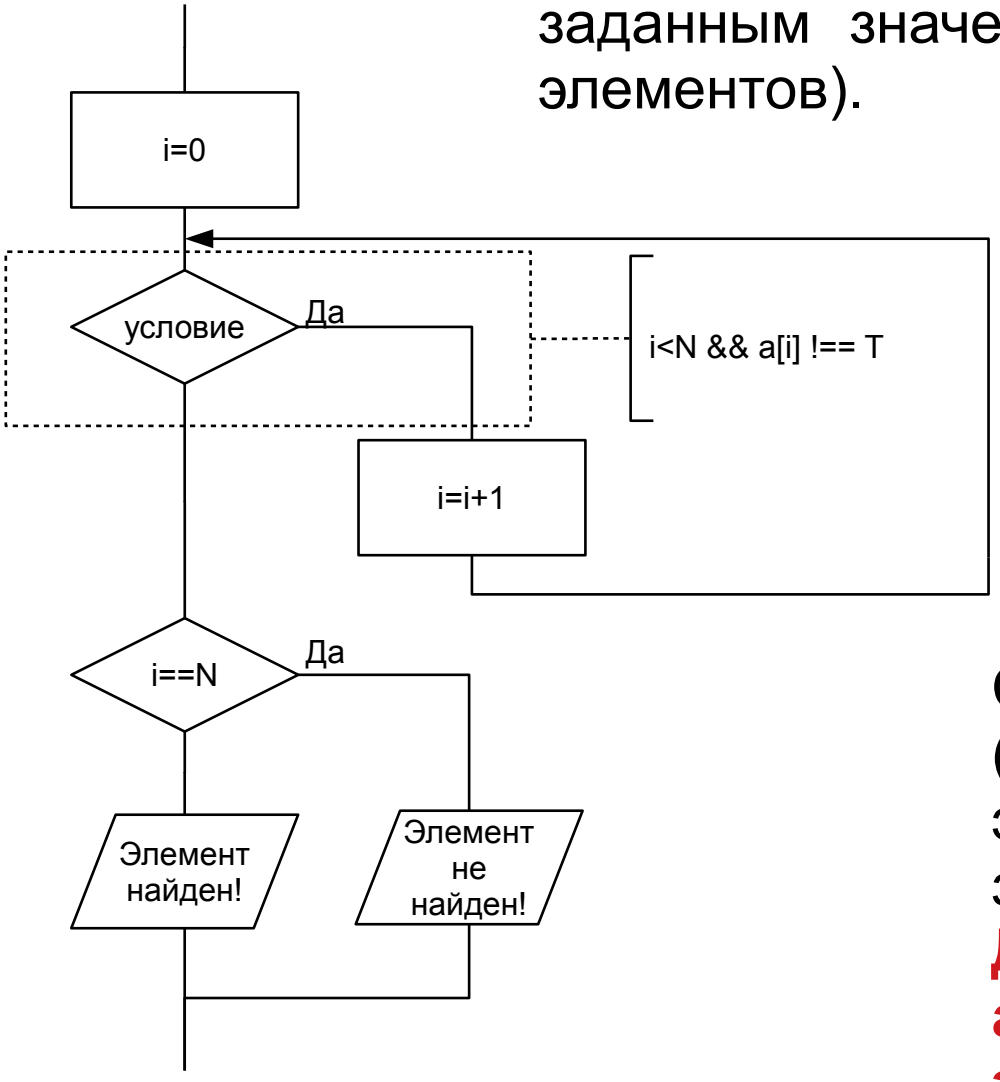
Вычислительные действия:

- Сумма элементов (возможно, выборочно)
- Произведение элементов (возможно, выборочно)
- Добавление значений к элементам
- Умножение элементов на число
- …

Пример lect-03-13.c (сумма элементов массива с четными номерами).

Типовые действия с массивами

Алгоритм проверки наличия элемента с заданным значением T (N — количество элементов).



См. также пример lect-03-14.c (количество и индексы элементов с заданным значением)

Д/З: Составить схему алгоритма решения этой задачи.

Типовые действия с массивами

Копирование элементов массива:

Создается новый массив из каких-то элементов исходного (подряд или с заданным шагом, начиная с начального i_n до какого-то i_k.

В зависимости от параметров i_n и i_k и шага, в целевой массив копируется подмножество элементов исходного массива.

Для копирования всех элементов исходного массива необходимо задать $i_n=0$, $i_k=n$ (n – количество элементов исходного массива).

При копировании части массива, например с 3 по 9, принимаем $i_n=2$ (поскольку нумерация элементов массива начинается с нуля) и $i_k=8$.

Пример lect-03-15.c — копирование элементов начиная с заданного номера с заданным шагом.

Д/З: Составить схему алгоритма решения этой задачи.

Типовые действия с массивами

Сортировка массива:

Сортировка – расстановка элементов массива в заданном порядке (по возрастанию или по убыванию).

Основные алгоритмы (методы) сортировки

- Сортировка выбором
- Сортировка обменом (метод «пузырька»)
- Сортировка вставками
- Сортировка Шелла (модификация сортировки вставками)
- Сортировка Хоара («быстрая сортировка», quicksort).

Типовые действия с массивами

Сортировка выбором
(по возрастанию — выбор максимального, по убыванию — выбор минимального).
Вычислительная сложность — N^2 операций.

При сортировке по возрастанию
1) Ищем максимальный элемент
2) Меняем местами максимальный элемент с последним элементом
3) Уменьшаем количество просматриваемых элементов на 1 («укорачиваем» массив с конца)
4) Повторяем операции 1-3 пока количество просматриваемых элементов не станет 1.

Схема алгоритма: ***alg-sort-sel.jpg***

При сортировке по убыванию ищем не максимальный, а минимальный элемент и переставляем его тоже в конец.

Типовые действия с массивами

Сортировка обменом (метод «пузырька»)
Вычислительная сложность — N^2 операций.

При сортировке по возрастанию
1) Начиная с первого и до предпоследнего элемента (N-1) сравниваем текущий элемент со следующим, если текущий > следующего, меняем их местами
2) Повторяем действия с 1-го элемента до N-2, затем с 1-го до N-3 и т.п

Схема алгоритма: ***alg-sort-bubble.jpg***

При сортировке по убыванию сравниваем не на >, а на <.

Типовые действия с массивами

Сортировка вставками
Вычислительная сложность — от N до N^2 операций.

Первый элемент массива считается отсортированным.
Каждый следующий прочитанный элемент массива размещается таким образом, чтобы предыдущий был меньше, а последующий — больше (при сортировке по возрастанию).

Схема алгоритма: ***alg-sort-insert.jpg***

При сортировке по убыванию — предыдущий элемент должен быть больше, последующий — меньше.