Технологии разработки алгоритмов решения инженерных задач

лекция №2

Преподаватель: Дреев Александр Николаевич

2. Правила оформления программы:

2.1. Структурная, функциональная, блок схемы. Принцип написания от комментариев к программе.

2.2. Оформление программы для удобства чтения. Выравнивание текста. Олицетворение функций, условий, циклов.

2.3. Принцип универсальности использования. Потоки ввода-вывода, стандартизация.

Правила оформления программы

Причины введения правил оформления

Программы пишутся на ГОДА - основная часть в работе программиста является не создание программного продукта, а его совершенствования и сопровождения. Через месяц идентификатор приравнивается к чужому, но ошибки исправлять нужно, нужно дополнять функциональность, и тому подобное.

Правила оформления программы

Структурная, функциональная, блок схемы.

Постановка задачи:

1. Входные данные и способы их получения
2. Исходные данные и способы их выведения
3. Требования к времени выполнения
4. Требования к использованию памяти
5. Требования к размеру программы

6. Сроки выполнения
7. Методика проверки функциональности
8. Доведение правильности

Правила оформления программы

структурная схема

На примере сортировки выборке

С чего состоит Для программы выделяют:

1. Блок получения массива
2. Блок обращения к элементам массива
3. Блок получения наибольшего элемента в указанном диапазоне
4. Блок формирования под диапазонов
5. Блок перестановки элементов
6. Блок вывода результата

Правила оформления программы

функциональная схема

Строится функциональное взаимодействие:

КАК РАБОТАЕТ

1. Цикл подготовки массива
2. Цикл определения подмассив, выдает первый номер
3. Цикл поиска номера максимального элемента
4. Изменение элементов местами
5. Цикл вывода результата

Введение в предмет

Схема потоков информации

ВЫДЕЛЕНИЕ ПОТОКОВ

1. Поток ввода
2. Поток индексов начала поиска максимального
3. Поток значений элементов для поиска максимального значения
4. Поток индексов максимальных элементов
5. Потоки перемены местами элементов
6. Поток вывода

Определение интенсивных потоков

Правила оформления программы

анализ алгоритма

По структуре: максимализация отделения блоков управления информацией и контейнерами информации

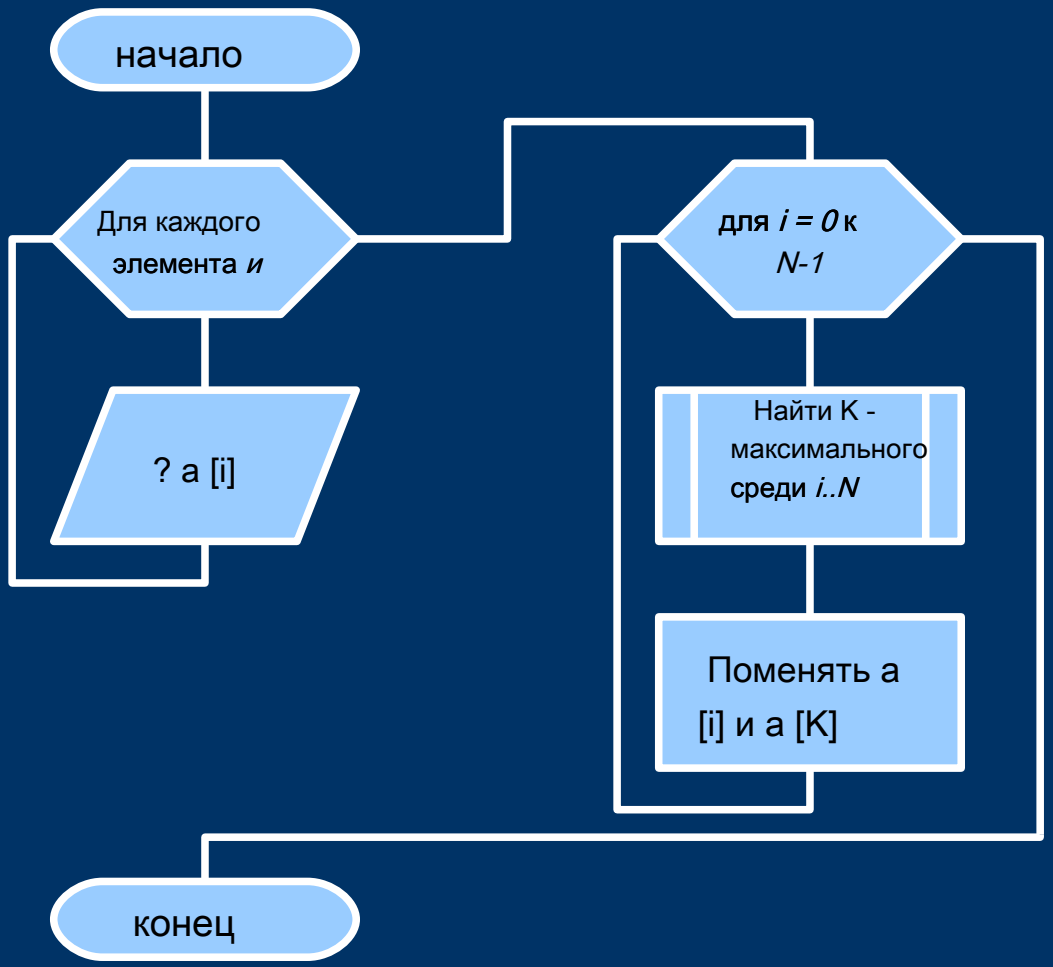
По функциональности: минимизация количества взаимодействующих связей модулей программы без учета движения информации

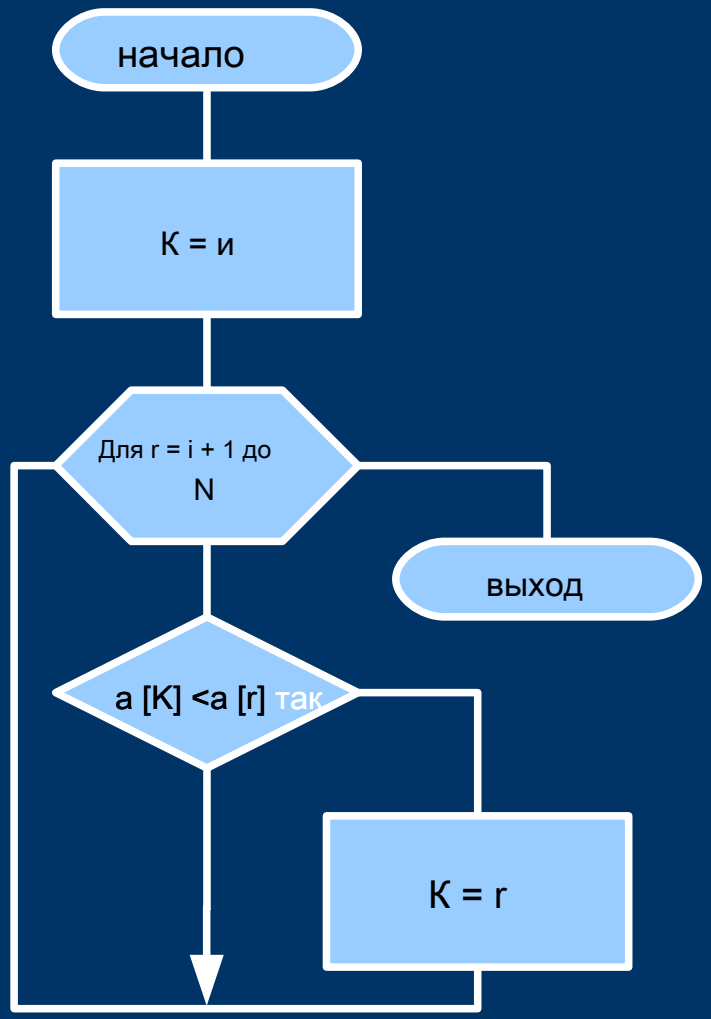
По потоками информации: минимилизация движения информации

Коррекция схем

Правила оформления программы

Создание блок-схемы алгоритма





Правила оформления программы

Принцип написания от комментариев к коду

```
# include <stdio.h>
/ * Функция для переданного массива A начиная с номера и
 *  ищет максимальный элемент и возвращает его номер * /
int findMaxFroml ( int * A, int start, int length)

int main ( int N, char * Param []) { / * Описание массива и
переменная для его длины * /
    int Massiv [ 256 ], K;
    int i, Tmp, M; / * Счетчик, временная, номер max * / / * Цикл ввода
    массива до конца файла или до k = 256 * /
    while (! Feof (stdin) && k < 256 ) { scanf ( "% i", &

    Massiv [k]);
        k ++; }
```

Правила оформления программы

Принцип универсальности использования, потоки

```
/ * Для каждого элемента введенного массива выполнить * /
    for (I = 0; i <k; i ++) { / * Определим номер
    максимального * /
        M = findMaxFromI (Massiv, i)
        / * Поменяем максимальный с настоящим элементом * /
        Tmp = Massiv [M];
        Massiv [M] = Massiv [i]; Massiv [i] = Tmp; } / * Массив
      отсортировано, приступим к выводу * /

    for (I = 0; i <k; i ++) { printf ( "% i \ n", Massiv
[i]); }}
```

Правила оформления программы

Оформление программы для удобства

```
int findMaxFromI ( int * A, int start, int length) {  int i, M; M = start; / * Первый
  элемент считаем самым * /

    for (I = start; i <length; i ++) { / * Для каждого из дальнейших
    элементов проверим, он
        * больше * /
      if (A [i]> A [M]) { / * Если действительно больше, то изменим номер
      крупнейшего * /
          M = i; }} return M; / * Возвращаем значение наибольшего элемента *
     /

}
```

Основные принципы проектирования алгоритмов

лекция №3

3. Основные принципы проектирования алгоритмов.

3.1. Принцип выполнения алгоритма на процессоре. Пример.

3.2. Метод пошаговой детализации.

3.3. Запись алгоритма.

3.4. Числа, ссылки, массивы, структуры.

Основные принципы проектирования алгоритмов

Принцип выполнения алгоритма на процессоре

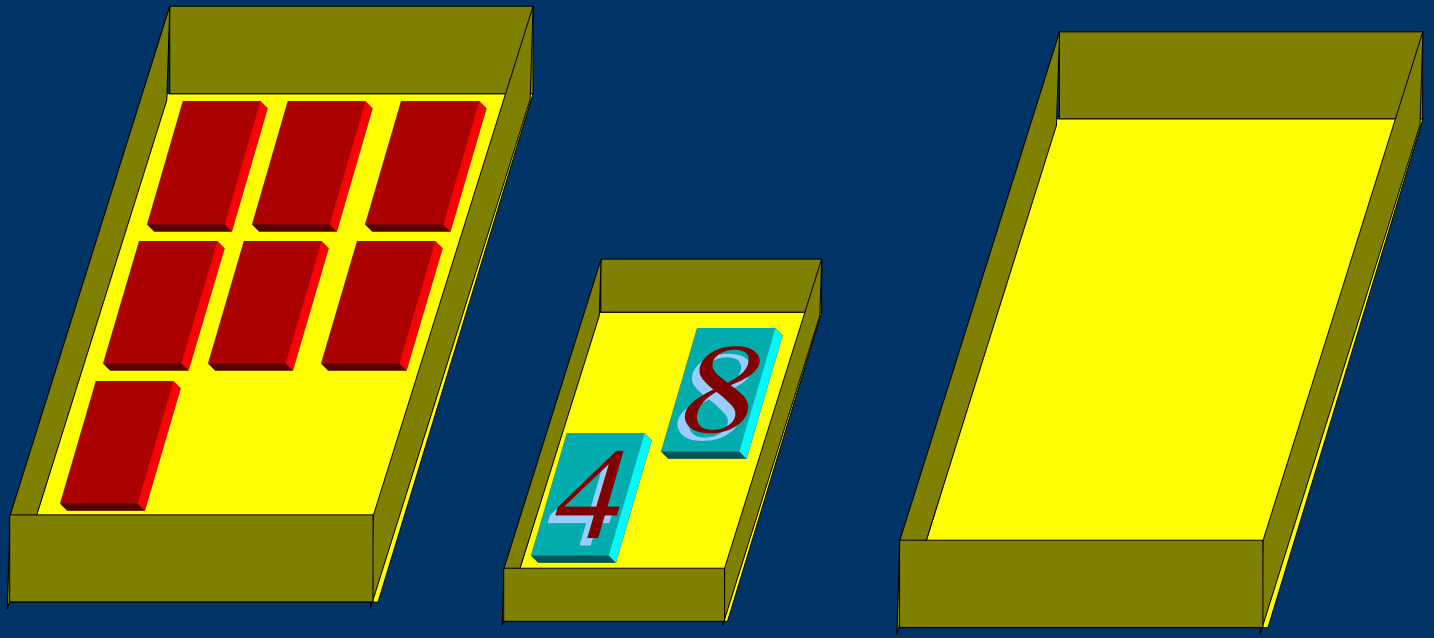
Пример. Поиск меньших.

1. Выходные данные: коробка с перевернутыми картами, на которых написано число.
2. Исходные данные: Карточки с числом меньше число на первой карточке
3. Условия выполнения: одной рукой исполнитель может получить или вернуть карточку в коробку, исполнитель может увидеть число на карточке только на килимку- "устройства исполнения" где помещается только две карточки.

Основные принципы проектирования алгоритмов

Принцип выполнения алгоритма на процессоре

Пример. Поиск меньших.



ОЗУ ЦБ ПЗУ

Память Регистры + память

логический пр.

Основные принципы проектирования алгоритмов

пошаговая детализация

Пошаговая детализация:

1. Обзор задачи в целом
2. Определение, "что я должен уметь, чтобы решить эту задачу".
3. Построение алгоритма для решения основной задачи
4. Повторить решение для поставленных подзадач в пункте 2.

ПРИМЕР: Сортировка по убыванию.

Основные принципы проектирования алгоритмов

пошаговая детализация

Задача: Отсортировать массив.

1. Пусть в массиве элементы 0..N-1 уже отсортированы от большего к меньшему. Как добавить к такому массива еще один элемент?
2. Есть общий массив. Первый элемент является упорядоченным массивом. К нему можно добавить следующий, но так, чтобы массив с новым элементом сохранил сортировки. Действие повторять и сортированный массив начнет увеличиваться.

Основные принципы проектирования алгоритмов

пошаговая детализация

3. Алгоритм:

а) Первый элемент считаем начальным отсортированным подмассив

б) Следующий элемент добавляем к подмассив с сохранением сортованости

г) Пока является не сортированные элементы, переходим к пункту б)

подзадача: последний элемент отсортированного массива не сортированный, передвинуть его на свое место

Основные принципы проектирования алгоритмов

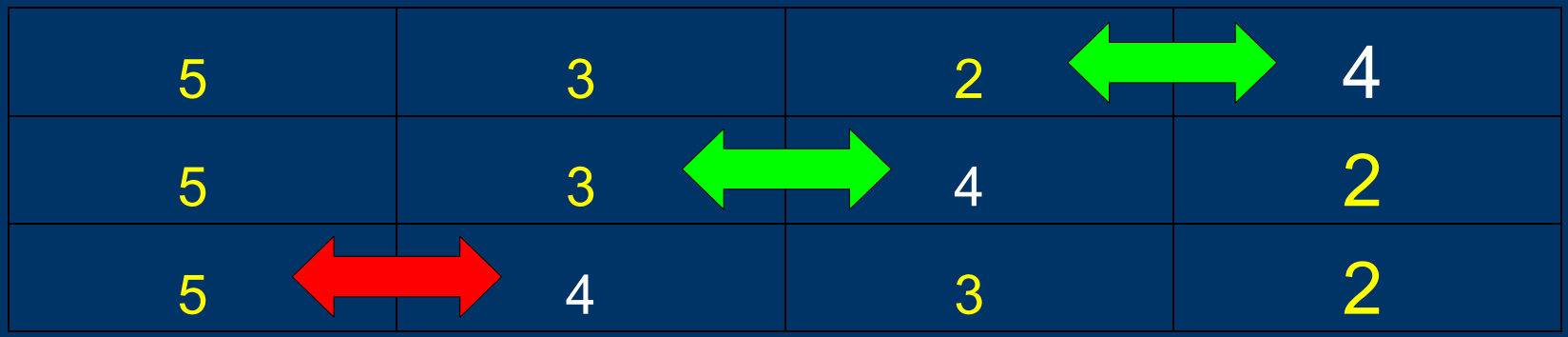
пошаговая детализация

4. Подзадача:

Последний элемент отсортированного массива не сортированный, передвинуть его на свое место.

а) Если предыдущий элемент от не сортирован меньше, то поменять их местами.

б) Повторить пункт а) пока не достигнем начала массива, или не было обмена местами



Основные принципы проектирования алгоритмов

запись алгоритма

```
# include <stdio.h> void addElement ( int * A, int length)


void main () {  int M [256], lengthM, i;


    . . .    / * Прочитать массив * /
    for (I = 1; i <lengthM; i ++) { / * Увеличение
    отсортированного части * /
        addElement (M, i)}}
```

Основные принципы проектирования алгоритмов

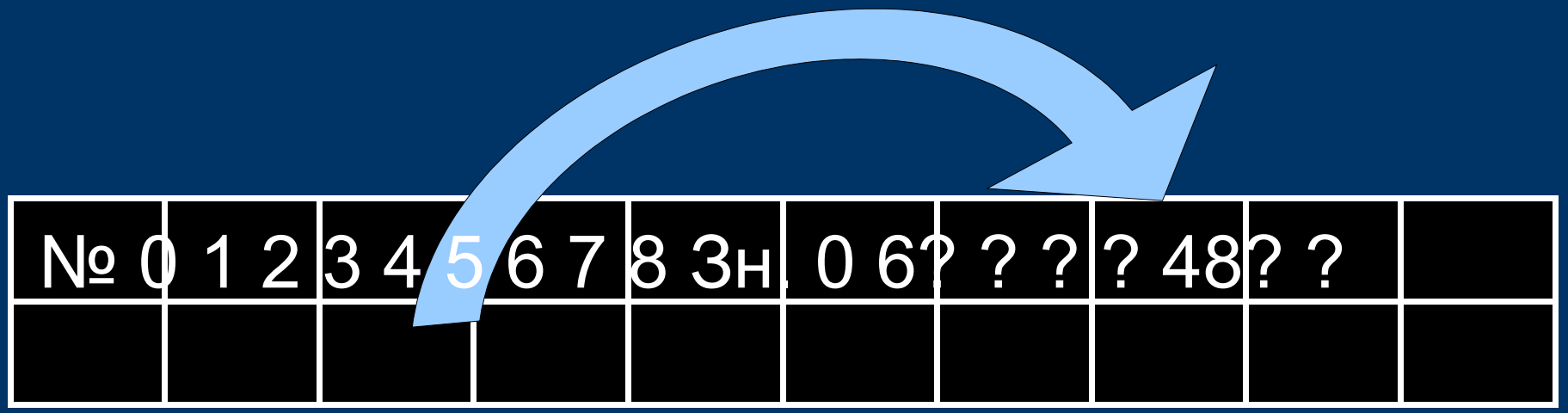
запись алгоритма

```
/ * Увеличение отсортированного части * /
void addElement ( int * A, int length) {  int k, tmp;


    for (K = length; k> 0; k--) { / * С конца подмассив до начала * /

      if (A [k]> A [k-1]) / * Если наш элемент больше * /
      { / * Изменение элементов местами * /
        tmp = A [k]; A [k] = A [k-1]; A [k-1] = tmp; } else

      { / * Элемент уже на месте, возвращаемся * /
        return ; }}}
```

Основные принципы проектирования алгоритмов

Числа, ссылки, массивы, структуры

- число - значение которое хранится в памяти компьютера.
- ссылка - значение адреса памяти по которой записаны данные (число).



Основные принципы проектирования алгоритмов

Числа, ссылки, массивы, структуры

- массив - значение хранящихся в памяти в последовательных адресах.
- структура - несколько значений записанных последовательно в определенном порядке.