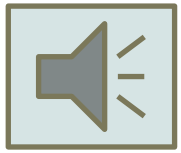
СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ



Янченко (Курапова)
Елена Викторовна

Кафедра ПМиК: 430а (гл.корпус),
406 (нов.корпус)

Формы освоения материала

Лекции

Домашние задания

Лабораторные работы

Курсовой проект

Формы контроля знаний

Контроль посещения лекций (проверка!)

Проверка домашних заданий (оценка)

Защита лабораторных работ (оценка)

Защита курсового проекта (оценка)

Экзамен (оценка)

Конспект лекций (конкурс!)

Рейтинг (баллы)

Лекции: посещение (+5), ответы (+), пропуск (-5)

Лабораторные работы: оценки 3, 4, 5, 5+

баллы 3, 5, 7, 10

пропуск (-5)

Домашние задания: оценки 3, 4, 5, 5+

баллы 1, 3, 5, 7

Автомат: ≥80% от max, все лабораторные ≥4, ДЗ ≥4, контрольные сроки ≥ 1

Собеседование (полуавтомат):

от 60% до 80% от max, все лабораторные ≥3, ДЗ ≥3, контрольные сроки ≥ 0

Экзамен по билетам: <60% от max с предварительной отработкой лабораторных работ

Материалы в электронном виде

CYBER2008 \ TXT \ KURAPOVA

posobie.doc - *теория*

Задания на лабы

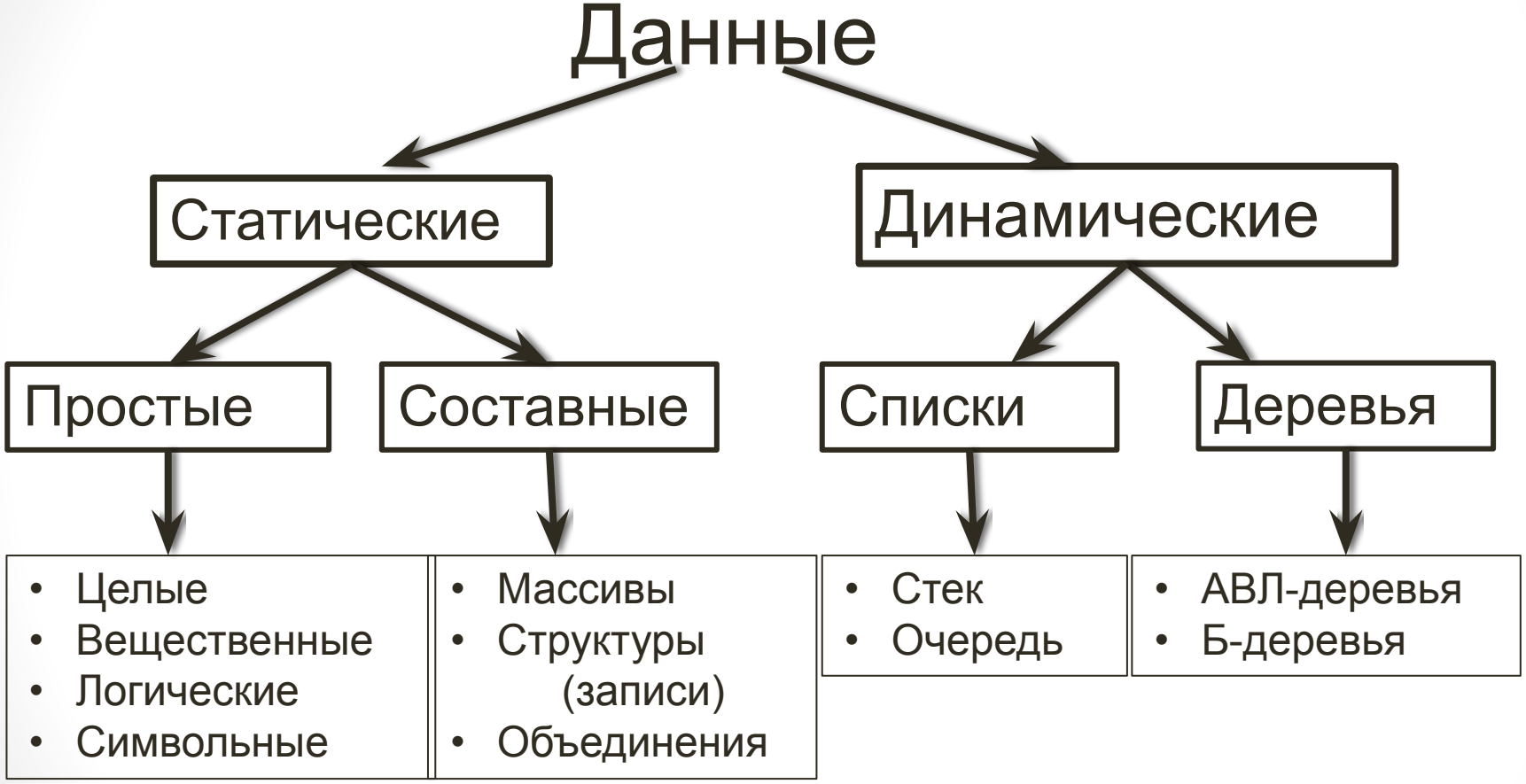
Сортировки

Lab1.doc

Lab2.doc

...

Основные структуры данных



- Статические данные имеют фиксированную структуру, поэтому размер выделенной для них памяти постоянен.
- Динамические данные изменяют свою структуру в процессе работы программы,
при этом изменяется объём выделенной памяти.

Простые типы данных

- Тип характеризует множество значений, которые может принимать переменная.
- Целые типы различаются количеством байт, отведённых в памяти и наличием знака. (int, short, long)
- Вещественные типы характеризуются точностью представления числа. (double, float, long double)

- Символьный тип определяет полный набор ASCII кода.
- Перечислимый тип - перечисление всех возможных значений.
- Логический тип - частный случай перечислимого типа с двумя возможными значениями ИСТИНА и ЛОЖЬ.

Составные типы

Структурированные (составные) типы всегда определяют набор компонентов *одинакового* или *разного* типа.

Массивы – структура данных, которая представляет собой фиксированное количество элементов одного типа.

Тип *элементов* массива - любой,

тип *индексов* массива –

только скалярный.

Массив – это структура данных со случайным (прямым) доступом.

Способы доступа к памяти

1. Прямой доступ (случайный) – в любой момент времени доступен любой элемент.
2. Последовательный доступ – (***к+1***)-й элемент может быть получен только путем просмотра предыдущих ***к*** элементов.

Записи (структуры)

Запись состоит из фиксированного числа компонент называемых **полями**, которые могут быть разных типов.

Пример. Struct data { char day;
char month;
int year; }
zap[n]; - массив записей.

Поле записи может являться записью, тогда образуются вложенные записи.

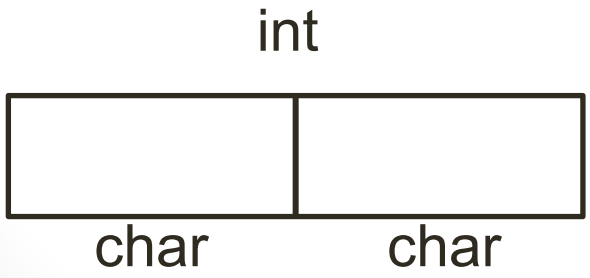
Объединения

Используются для размещения в одной и той же области памяти данных различного типа.

Но в каждый конкретный момент времени используются данные только одного типа.

Пример. Union w

```
{ int a;
  char b [2]; }
```




Simula
Visual C++
Visual Basic
APL
HTML
Cobol
PL/1
Algol
Eiffel
Fortran
Basic
awk
Prolog
Lisp
Ada
Icon
Pascal
C++
sed
C
Smalltalk
Modula-2
Perl
Scheme
Delphi
JAVA

Псевдокод
(*некоторые соглашения по записи алгоритмов*)

Алгоритм на псевдокоде записывается в свободной форме на естественном языке

с использованием двух формализованных конструкций: ветвления и повтора.

Конструкция_ветвления

- **IF** (условие)
  <действие>
  FI

- **IF** (условие)
  <действие1>
  ELSE
  <действие2>
  FI

- **IF** (условие1)
  <действие1>
  ELSE IF (условие2)
  <действие2>
  FI
  FI

Конструкция повтора

1. **Цикл с предусловием**

DO (условие)
    <действия>
OD

2. **Цикл с постусловием**

DO
    <действия>
OD (условие)

3. **Цикл с параметром**

DO (i := 1, 2, …, n)
    <действия>
OD

4. **Бесконечный цикл**

DO
    <действия>
    …
    IF (условие) **OD FI**
    …
OD

Присваивание **:=**
Обмен значений ↔

Сортировка

Причины, по которым мы обращаемся к задаче сортировки в нашем курсе

1) Сортировка – фундаментальная деятельность, без которой не обходится ни одна обработка реальных данных.

2) На примере сортировки удобно рассматривать множество алгоритмов, сравнивать и анализировать их.

Постановка задачи сортировки

Пусть дан массив $A = \{ a_1, a_2, \ldots, a_n \}$.

Для всех его элементов определены ***операции отношения***: меньше, больше, равно (**<, >, =**).

Необходимо **отсортировать массив**, т.е. переставить элементы массива A таким образом, что выполняется одно из следующих неравенств:

$$a_1 \le a_2 \le a_3 \le \ldots \le a_n \quad (1)$$
$$a_1 \ge a_2 \ge a_3 \ge \ldots \ge a_n \quad (2)$$

Если выполняется неравенство (1), то массив отсортирован по возрастанию или в прямом порядке.

Если выполняется неравенство (2), то массив отсортирован по убыванию или в обратном порядке.

Цель сортировки ???

– облегчить (ускорить) последующий поиск элементов в отсортированном массиве.

Проверку правильности сортировки осуществляем с помощью:

- подсчета контрольной суммы

- подсчета количества серий.

Определение. Контрольная сумма – это сумма всех элементов массива.

$$КС = \sum_{i=1}^{n} a_i$$

Вычисляется до и после сортировки.

Определение. **Серия** – это неубывающая последовательность максимальной длины.

Пример:

5 неубывающих послед-тей => 5 серий

Массив, отсортированный по возрастанию, состоит из *одной серии*,
а в массиве, отсортированном по убыванию, количество серий равно *количеству элементов* в массиве (если все элементы различны).

Сортировка элементов со сложной структурой

Пример: Телефонный справочник – сложная структура, состоит из:

Фамилии; Имени; Телефона; Адреса.

Чтобы его отсортировать, нужно определить *отношения порядка* (**<,>,=**).

Например, пусть при сравнении двух элементов меньше тот, чья фамилия идет раньше по алфавиту.

Если *фамилии совпадают*, то меньше элемент, у которого имя раньше по алфавиту.

Если *фамилии и имена совпадают*, то будем считать, что элементы равны.

Таким образом, мы выбрали поля телефонного справочника для определения отношения порядка (**<,>,=**).

Определение. Та часть информации, которая учитывается при определении отношения порядка, называется *ключом сортировки.*

В данном примере ключ сортировки: фамилия и имя - составной ключ из двух полей (фамилия – ***старшая часть*** ключа, имя – ***младшая часть*** ключа).

Ключ поиска обычно

-равен ключу сортировки ***или***

-состоит из его старшей части.

Сортировка рассматривается с точки зрения двух свойств:

1. Устойчивость сортировки.

2. Зависимость от исходной упорядоченности массива.

Устойчивость сортировки

Определение Сортировка называется *устойчивой*, если после её проведения в массиве не меняется относительный порядок элементов с одинаковыми ключами.

Пример:

1 2 **5’** 6 3 4 **5”**

Итог сортировки:

1 2 3 4 **5’** **5”** 6 - устойчивая сортировка

1 2 3 4 **5”** **5’** 6 - неустойчивая сортировка

В общем случае устойчивая сортировка лучше, чем неустойчивая. В частности, когда данные уже были ранее упорядочены по другим ключам.

Зависимость от исходной упорядоченности массива

Эта характеристика показывает, изменяется ли трудоемкость сортировки в зависимости от исходных данных.

Но все же…

Главным качественным показателем сортировки является быстрота работы (трудоемкость).

Для сравнения методов сортировки необходимо оценивать их **эффективность по времени.**

Трудоемкость сортировки

Операции, влияющие на трудоемкость сортировки:

1. **Сравнение** элементов - **C** (“Compare”);
2. **Перестановки** элементов – **M** (“Move”).

Мы будем оценивать трудоемкость количеством операций сравнения и перестановки:

$$C + M = T$$

C и **M** зависят от количества элементов в массиве:

$$C(n) + M(n) = T(n),$$

где C(n), M(n), T(n) – функции от количества элементов массива.

При подсчете трудоемкости учитываются *только те операции*, в которых участвуют *элементы массива*.

Предварительно обнулить счетчики **C=0; M=0;**

При подсчете количества сравнений счетчик желательно ставить **перед** условным оператором.

Пример: **C++; if (a_i < x) …**

При подсчете количества пересылок счетчик ставится **до или после** оператора присваивания:

Пример: **a_i=x; M++;**

Рассматриваемые алгоритмы написаны на естественном языке, в котором массив нумеруется от 1 до n: $\mathbf{A = \{ a_1 , a_2 , \ldots , a_n \}}$.

Для удобства реализации алгоритмов на языке C++, массив возможно описывать следующим образом:

int A [1+n];

Тогда при заполнении, обработке и выводе массива элемент **A[0]** не используется.