



ГУАП

Государственный университет
аэрокосмического приборостроения

www.guap.ru

Информатика

**Рождественская Ксения
Николаевна**

Кафедра 14

ksu.khramenkova@gmail.com



Структурирование

- **Структурирование** – совокупность технологий программирования, приемов и закономерностей, используемых при создании программного продукта

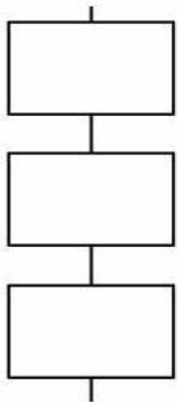
Как написать безошибочную программу?
Как написать программу в хорошем стиле?

Структурирование

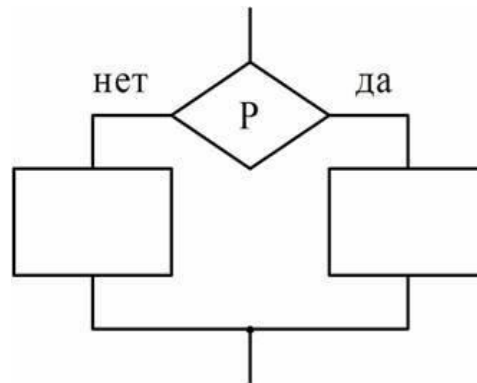
Теорема о структурировании

Как бы сложна ни была задача, схема соответствующей программы всегда может быть представлена с использованием ограниченного числа элементарных управляющих структур

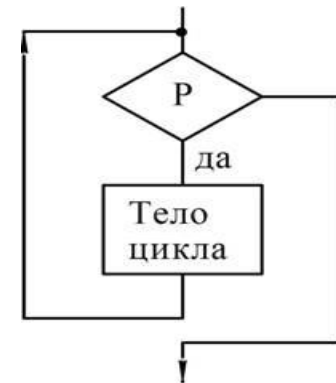
Базовыми элементарными структурами являются



Линейны
й



Ветвящийс
я



Циклически
й

Структурирование

Какова бы ни была степень и глубина «вложенности», важно, что любая конструкция в конечном итоге имеет **один вход** и **один выход**



Лемма о базовых структурах

Любую сложную структуру можно рассматривать как «черный ящик» с одним входом и одним выходом



цикл for не может быть базовой структурой, однако, его можно реализовать через базовые циклы ДО и ПОКА; чтобы case был базовой структурой, надо отображать его в виде множества if

В структурированной программе выше надежность, проще сопровождение программы, проще делать модификацию программы

Структурирование

- Существует три основные технологии структурирования
 - Нисходящая
 - Восходящая
 - Комбинированная

Нисходящая

- Задачи разбиваются на подзадачи, которые можно проектировать отдельно
- Подзадача вновь разбивается, и так до тех пор, пока это разбиение имеет смысл
- Эта технология полезна, когда возможно укрупненный алгоритм разбить на подзадачи и легко установить связь между ними

Структурирование

Восходящая

- Общий алгоритм неясен, но известно решение отдельных алгоритмов и задач, то можно начинать проектирование с низшего уровня, решая мелкие подзадачи, устанавливая связи между ними

Комбинированная

- Комбинация нисходящей и восходящей технологий

Структурирование

Последовательность действий при структурировании:

- Попытка решения задачи за один шаг. Если на этом шаге появляется хоть одно укрупненное предписание, оно разукрупняется с использованием только одной базовой структуры на каждом шаге
- Встретив укрупненное предписание далее, мы опять разукрупняем его, и так до получения только базовых структур. Все это – **детализация**
- После проведения полной детализации осуществляется рекурсивное вложение структурных формул друг в друга. Полученное в результате описание – и есть **структурная программа**

Структурирование

- Структурирование позволяет пропустить ни одного шага.
 - ❖ Сделать большой шаг, пропустив часть предписаний.
 - ❖ Предотвращается появление сложных связей типа goto
- Структурирование **применяется** в случаях, когда потребуется модификация программы в процессе ее эксплуатации
- Структурирование **не применяется**, если задача решается в режиме реального времени и имеются строгие временные ограничения

Структурирование

Пример

Задача: дана последовательность, содержащая от 2 до m слов ($m \geq 2$), в каждом из которых от 1 до n ($n \geq 1$) строчных латинских букв; между соседними словами – не менее одного пробела, за последним словом – точка.

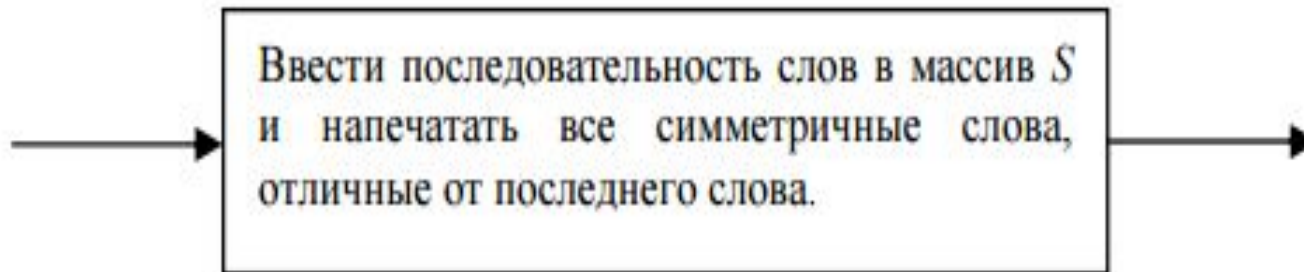
Решение

Для представления слова в программе будем использовать строку длины n . Поскольку каждое вводимое слово требуется сравнивать с последним словом, все вводимые слова требуется сначала запомнить. Для этого воспользуемся массивом строк S .

Структурирование

Пример.
Решение

Шаг
1



Структурирование

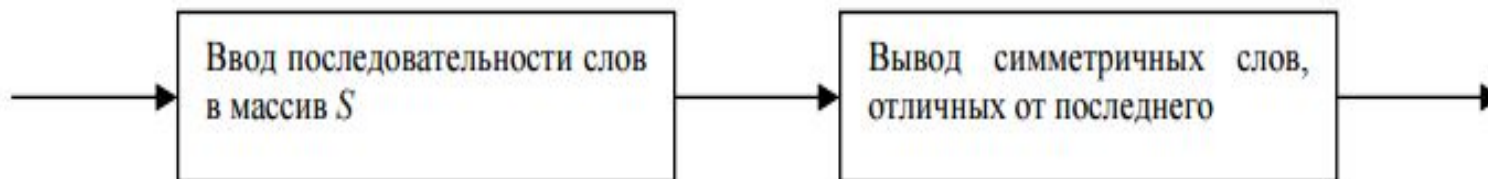
Пример.

Решение

Шаг

Решение задачи разбивается на два последовательных блока:

- ввести данные в массив;
- просматривая массив от начала до конца, выполнить поиск слов по доп. условию (не рассматривается в данном примере)

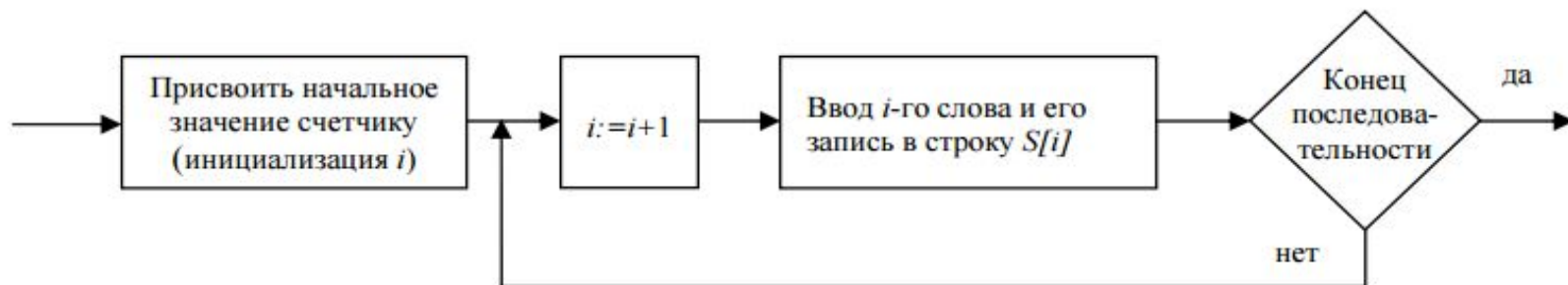


Структурирование

Пример.
Решение

Шаг

- Детализируем ввод последовательности слов: очередное слово записывается в строку $S[i]$, где i – переменная счетчик слов; счет начинается с 1, по окончании ввода значение i будет равно количеству введенных слов.
- Поскольку последовательность не пуста (есть хотя бы одно слово), организуем ввод, используя циклическую конструкцию с постусловием

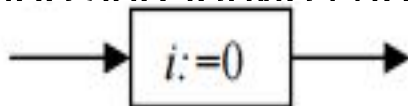


Структурирование

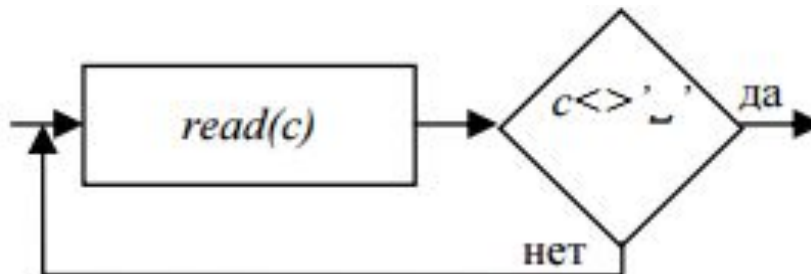
Пример.
Решение

Шаг

- Уточним блоки, входящие в конструкцию, полученную на Шаге 3.
- Для того, чтобы первое слово записывалось в первую строку, то, очевидно, начальное значение i должно быть нулевым



- Представим теперь ввод слова как посимвольный с пропуском лишних пробелов

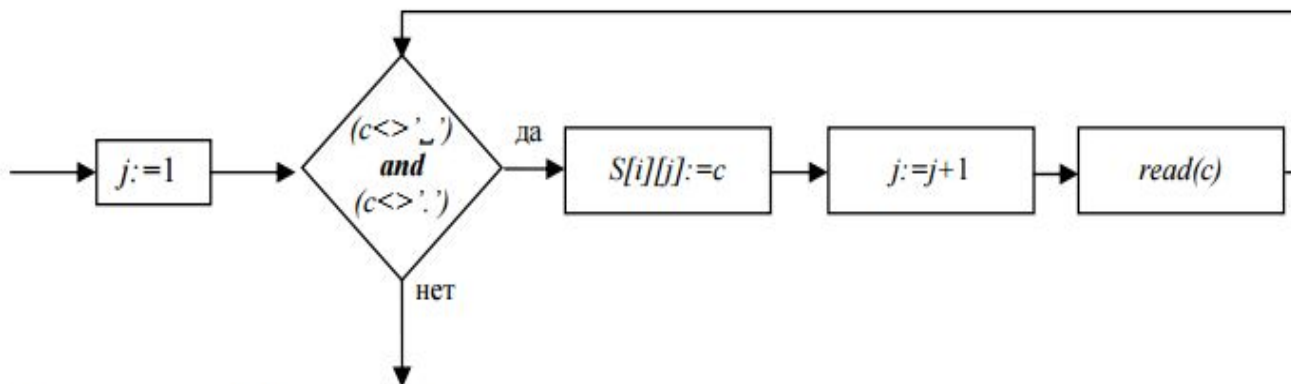


Структурирование

Пример.
Решение

Шаг

- После пропуска пробела вводим символы слова (по условию в слове есть хотя бы один символ).
- Признак конца слово – очередной символ оказался пробелом или точкой.
- Пока не конец слова, записываем очередной символ в массив S и считываем в переменную c следующий символ.



Методы построения алгоритмов

Этапы построения алгоритмов

Задача коммивояжера

Коммивояжеру для совершения торговых сделок требуется объехать n городов и вернуться в свой родной город, при этом в каждом городе он может побывать только один раз. Среди множества возможных маршрутов требуется найти тот, который позволит минимизировать затраты на поездку.

Методы построения алгоритмов

Этапы построения алгоритмов

Дополнительные условия и допущения:

- Используемое понятие стоимости должно быть формализовано
 - Для каждой пары городов a и b задана функция стоимости $C: (a,b) \rightarrow c, c \geq 0$, т.е. каждой паре городов ставится в соответствие неотрицательное число (стоимость) c
- Функция стоимости учитывает направление движения, т.е. в общем случае $C(a,b)$ не обязательно должно быть равно $C(b,a)$
- Будем считать $C(a,a) = \infty$, т.е. запретим переезд из города в этот же город
- Спрос на товары коммивояжера во всех городах одинаков, и единственное, чем определяются его предпочтения при выборе маршрута – это функция стоимости C .

Методы построения алгоритмов

Этапы построения алгоритмов

Зададим стоимости переездов между городами

	A	B	C	D
A	∞	10	1	3
B	10	∞	5	6
C	1	5	∞	7
D	3	6	7	∞

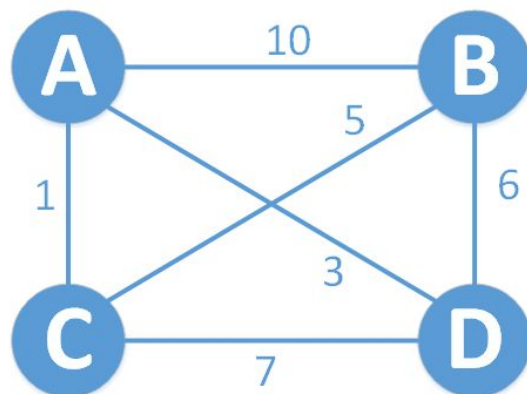
Методы построения алгоритмов

Этапы построения алгоритмов

Построение математической

модели

Опишем сеть из n городов как полносвязный взвешенный ориентированный граф (орграф), т.е. множество из n вершин, каждая из которых связана с любой другой ребром, при этом каждому ребру указано направление движения и стоимость. Такой граф можно представить $(n \times n)$ -матрицей стоимостей C



← Граф стоимостей

Методы построения алгоритмов

Этапы построения алгоритмов

Построение математической

модели

- Пронумеровав города от 1 до n , можно задать объезд как последовательность чисел, соответствующую порядку посещаемых городов, при этом каждое число в этой последовательности должно присутствовать только один раз.

$$\pi = \{j_1, \dots, j_n\}, j_i \in \{1, \dots, n\}$$

- Каждому объезду π можно поставить в соответствие функцию стоимости

$$C(\pi) = c_{j_1 j_2} + c_{j_2 j_3} + \dots + c_{j_{n-1} j_n} + c_{j_n j_1}$$

- Тогда искомое решение

$$\pi^* : C(\pi^*) = \min_{\pi} \{C(\pi)\}$$

Методы построения алгоритмов

Этапы построения алгоритмов

Выбор или построение

- Наиболее очевидный и простой метод – метод полного перебора всех возможных маршрутов.
- Т.к. любой объезд – это замкнутый цикл, то существует всегда $(n-1)!$ Различных маршрутов.
 - Для 4х городов существует $3!=6$ объездов (предполагаем, что начинается объезд из города А и заканчивается там же).

№	π	C(π)
1	A BCD A	25
2	A BDC A	24
3	A CBD A	15
4	A CDB A	24
5	A DBC A	15
6	A DCB A	25

Методы построения алгоритмов

Этапы построения алгоритмов

Проверка корректности алгоритма

- Для любого предложенного алгоритма должно быть доказано, что при любых входных данных алгоритм задачи завершится, и полученный ответ будет соответствовать требованиям в условии задачи
- Т.к. предложен был полный перебор всех вариантов, и число этих вариантов конечно, то это может служить гарантией того, что задача всегда будет решена и решена корректно.

Методы построения алгоритмов

Этапы построения алгоритмов

Анализ сложности алгоритма

- В нашей задаче всего существует $(n-1)!$ Маршрутов, их можно генерировать динамически или хранить в памяти, в любом случае сложность нахождения полного перебора задается функцией $O(n!)$.
- Нужно хранить матрицу стоимостей из n^2 ячеек.
- Таким образом, предложенный алгоритм решения задачи приемлем только для малых значений n .

Методы построения алгоритмов

Этапы построения алгоритмов

Реализация алгоритма

- Как правило алгоритм может быть реализован различными способами.
- Конкретная реализация может быть предназначена для конкретного требования или ограничения по памяти или времени выполнения.

Методы построения алгоритмов

Этапы построения алгоритмов

Проверка корректности программы

- Или тестирование, является самостоятельной большой задачей.
- На практике почти никогда нельзя гарантировать корректную работу сложных программ – речь идет лишь о корректной работе в определенных условиях и в заданном диапазоне входных данных.
- Существуют методики и подходы к тестированию, к выбору и построению систем тестов.

Методы построения алгоритмов

Этапы построения алгоритмов

Оценка сложности программы

- Время выполнения программы, занимаемая программой память не всегда прямо зависят от сложности алгоритма.
- Большое влияние оказывает реализация, аппаратная платформа.
- Иногда вопросы сложности реализации выходят на первый план. Бывает более выгодно реализовать «жадный» переборный алгоритм, чем более тонкий метод.

Методы построения алгоритмов

Этапы построения алгоритмов

Документирование программы

- Если с программой должны работать сторонние пользователи, наличие четкого и ясного руководства по работе становится обязательным.