Практическое занятие №6

Густов Владимир Владимирович
gutstuf@gmail.com

Цитата дня: Скажи НЕТ НАРКОТИКАМ, и, может быть, ты не закончишь так же, как разработчики Hurd. - (c) Линус Торвальдс

Ассоциативный массив

Абстрактная структура данных основанная на взаимоотношениях «ключ-значение».
Ассоциативный массив (читай - массив ассоциаций) содержит в себе пары (ключ, значение), где:
-ключ – уникальная идентификационная информация, с помощью которой можно «описать»/распознать значение;
- значение – какая-то информация, данные;
Так же считается, что ассоциативный массив не может хранить в себе две пары с одинаковыми ключами.

Примеры из жизни:
- таблицы в базах данных;
- журнал успеваемости/посещаемости студентов (ключом является ФИО студента, а значением – его оценки/Н-ки);
- турнирный зачёт (ключом будет считаться имя команды, а значением – её баллы/медали);
- телефонный справочник (ключом является номер телефона, а значением имя компании, т.к. одна и та же компания может иметь несколько разных номеров).

В общем, практически в любой области, где данные можно ассоциировать с каким-то (уникальным/не повторяющимся) значением, можно реализовать ассоциативный массив.

Словарь (таблица)

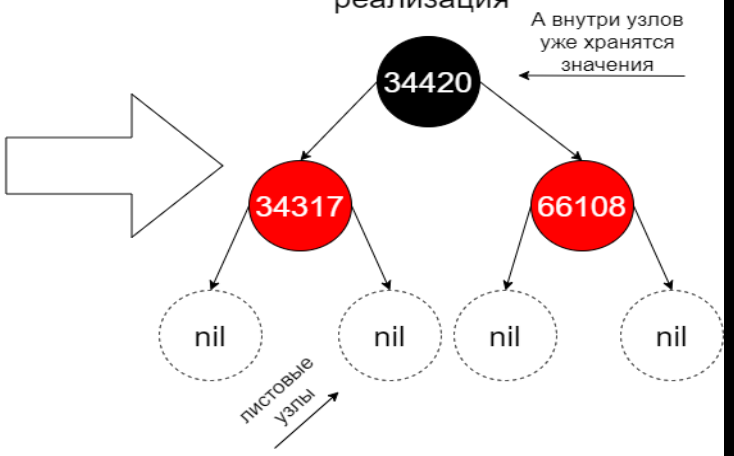
Абстрактная структура данных, является ассоциативным массивом реализующимся с помощью сбалансированного бинарного дерева поиска.

Свойства (как в сбалансированных БДП, далее СБДП):
- вставка/удаление элемента — O(log n);
- поиск элемента — O(log n);
- все пары (далее – элементы) отсортированы по ключу;
- не имеет пар с одинаковыми ключами.

Что видит пользователь структуры

Ключ	Значение
3-43-17	Дугин О.Б.
3-44-20	Воробьёв В.В.
6-61-08	Грудинин П.Н.

Что из себя представляет реализация



Ключ	Значение
Васин И.О.*	180
Дугина А. В.	180
Петров Р.Д.	175

Словарь

Предметная область: журнал учёта роста

* - для краткости, имя и отчество сокращается до инициалов

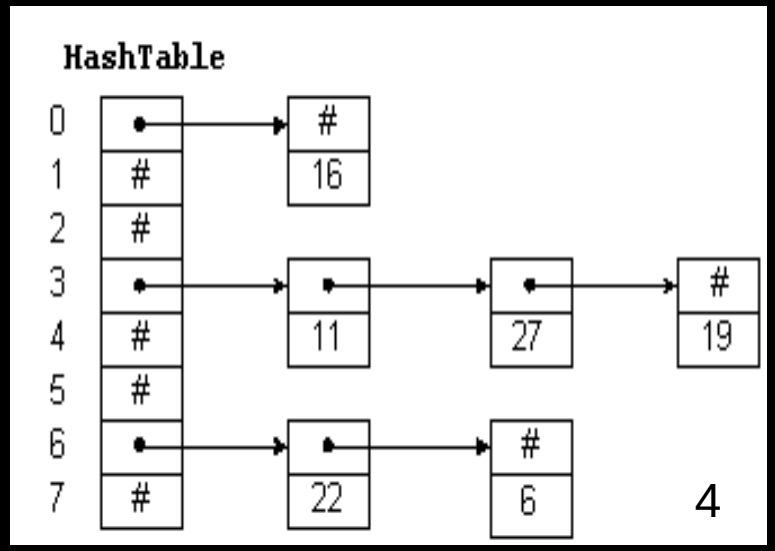
Хеш-таблица

Ассоциативный массив, чьи ключи преобразуются некоторой математической функцией (хеш-функция).

Свойства:

- вставка/удаление элемента — O(1) (в худшем – O(n));
- поиск элемента — O(1) (в худшем – O(n));
- все пары (далее – элементы) хранятся **не отсортированными**;
- может содержать элементы с одинаковым хешем (коллизия).

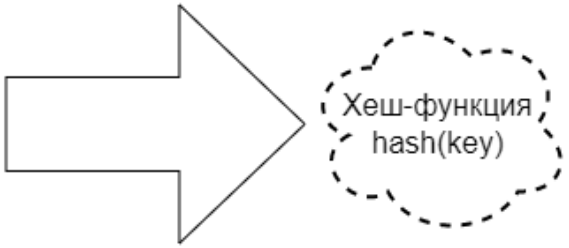
Хеш-функция – это функция преобразующая ключ ***key*** в некоторый индекс ***i*** равный ***h(key)***, где ***h(key)*** – хеш (хеш-сумма) ***key***.
Хеширование – процесс преобразования ключей в индексы хеш-таблицы.
Хеш (хеш-сумма) – результат обработки данных (ключей) хеш-функцией.



Хеш-таблица

Что видит пользователь структуры

Ключ	Значение
3-43-17	Дугин О.Б.
3-44-20	Воробьёв В.В.
6-61-08	Грудинин П.Н.



Что из себя представляет реализация

Индекс	Содержимое
0	указатель*
1	указатель
2	указатель

Ключ	Значение
3-43-17	Дугин О.Б.

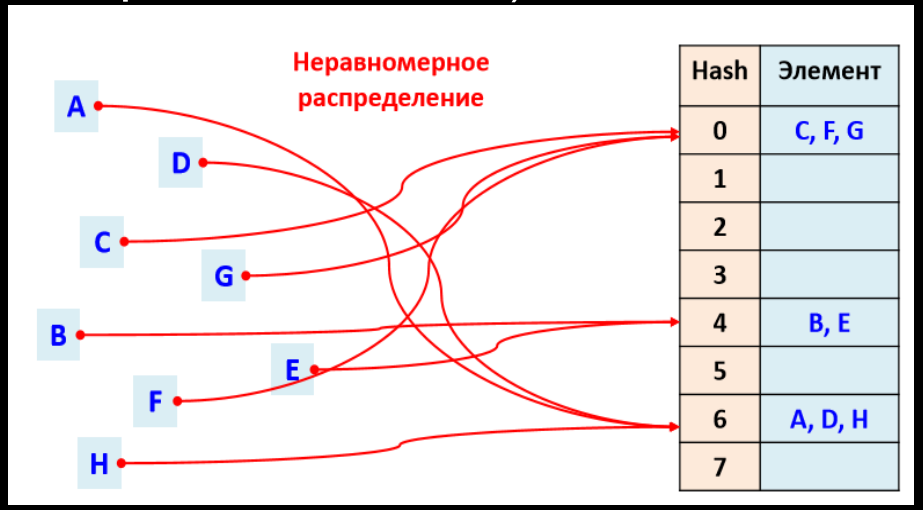
Ключ	Значение
3-44-20	Воробьёв В.В.

Ключ	Значение
6-61-08	Грудинин П.Н.

* - либо вместо указателя лежит сам объект-пара, содержащий в себе и ключ и значение

Пример простейшей (но плохой, с точки зрения коллизий) хеш-функции:

```
int calc_hash(string key) {
    int hash = 0;
    for(auto el: key)
        hash += el ^ hash;
    return hash;
}
```



Свойства (хорошей) хеш-функции:
- должна быстро вычисляться – дабы выполнение хеширования не стоило дороже, чем худшая операция за O(n);
- является равномерной – равномерно заполняет ячейки массива;
- не должна быть непрерывной – для близких значений должны получаться сильно различающиеся результаты;
- значения функции не должны образовывать кластеры – множество близко стоящих точек (значений);

Примеры хеш-функций: со схемой Горнера, алгоритм Седжвика

Так или иначе, практически любая хеш-функция для двух (и более) ключей может выдать один и тот же хеш – такая ситуация называется ***коллизией***.

Проверь себя

Дана следующая хеш-функция: нажми сюда
Проведите заполнение хеш-таблицы с учётом решения коллизий методом цепочек.
Набор ключей для заполнения:

1525, 1515, 123, 42, 5151, 23, 6, 321, 9, 0, 11, 101, 24.

В данной работе можете воспринимать вышеуказанные значения и как данные и как ключ.

Проверить себя можете здесь (не работает с отрицательными числами):
https://www.cs.usfca.edu/~galles/visualization/OpenHash.html

Язык

Любой язык можно представить как некоторое множество предложений или формул – строк символов – с корректно определённой структурой и со значением.

Правила, определяющие допустимые конструкции языка, составляют его синтаксис: **синтаксис языка описывает его форму**. Например, когда мы говорим, что «*x + 2*» является выражением, а «*x2 +*» таковым не является, мы **оперируем синтаксисом** алгебры.

Интерпретация символов и формул **есть семантика** языка.
Когда мы говорим, что «*x + 2*» есть сумма значения *x* и *2*, или что «*2 * x == x + x*» - есть истина, мы **апеллируем к семантике** алгебры.

Символы и **слова являются лексикой** языка. Лексика оперирует лексемами (словами). Лексема же состоит из элементов алфавита языка (токенов).
Например, «x + 2» содержит в себе 3 лексемы: x – идентификатор/переменная, + - оператор сложения, 2 – целочисленная константа.

Грамматика

Терминал – символ алфавита (например: буквы русского алфавита – а, б, в, и т.д.). Обычно обозначаются строчными буквами.

Нетерминал – множество нетерминальных и (группа) терминальных символов используемых в продукции языка (например: слова и предложения). Обычно обозначаются заглавными буквами.

Продукция – является правилом преобразования строки, имеющим левую часть (являющуюся образцом для распознавания преобразуемой подстроки) и правую часть (содержащую замену для части строки, соответствующей образцу).

Грамматики

Грамматика – это описание способа построения предложений языка. Грамматика описывается правилами порождения строк языка, т.е. **грамматика** – это генератор цепочек языка.
Формально грамматику можно описать с помощью системы правил (или продукций).

Два самых популярных способа записи правил грамматики: форма Бэкуса-Наура (БНФ) и расширенная форма Бэкуса-Наура (РБНФ).

Правило – это упорядоченная пара строк (α, β). В правилах важно соблюдать порядок следования строк.

Варианты записи правил:
α → β (БНФ)
<α> ::= <β> (БНФ)
α = β (РБНФ)

Читается как: α порождает β.

Формальные грамматики

L(G) – язык, заданный грамматикой G.
Формальная грамматика G определяется четырьмя параметрами:

`G(T, N, P, S)`, где:
T – множество (алфавит) терминальных символов;
N – множество нетерминальных символов;
P – множество правил (продукций) вида: «левая часть» → «правая часть», где:
- «левая часть» - **непустая последовательность** терминалов и нетерминалов, содержащая хотя бы один нетерминал;
- «правая часть» – **любая последовательность** терминалов и нетерминалов.
S – стартовый (начальный) символ грамматики из набора нетерминалов.

Параметр S в грамматике G означает, с какого нетерминала N начнётся описание правил в параметре P.

БНФ

Метаязык, в котором одни продукции последовательно описываются через другие.
Используется для описания контекстно-свободных грамматик.
С помощью БНФ описывается синтаксис языков программирования, протоколов, и т.д.

- нетерминальные символы обрамляются с помощью: «<» и «>»;
- терминальные символы обрамляются кавычками;
- комментарии указываются после символа «;»;
- выбор между продукциями указывается с помощью символа «|»;
- порождаемость (или – следование) указывается с помощью символов «→» либо «::=»;
- использование символа 0 или более раз указывается с помощью символа «*»;
- использование символа 1 или более раз указывается с помощью символа «+»;
- группирование символов производится с помощью обычных скобок: «(» «)»;

Пример БНФ

Описание идентификатора с помощью БНФ:

<буква> ::= “a” | “b” | “c” | “d” | “e” | “g”

<цифра> ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

<идентификатор> ::= <буква> | <идентификатор> <буква> | <идентификатор> <цифра>

Исходя из правил, можно сказать, что:

1) идентификатор состоит как минимум из 1 буквы;
2) идентификатор всегда начинается с буквы;
3) идентификатор может содержать числа;
4) идентификатор содержит буквы латинского алфавита от a до g и только;

Соответственно, для такой грамматики:

- abcd – валидный идентификатор;
- ec9 – валидный идентификатор;
- hje0 – невалидный идентификатор, т.к. в описании отсутствуют h и j;
- 2ab – невалидный идентификатор, т.к. исходя из описания, не может начинаться с цифры.

РБНФ

Расширенная версия БНФ. Также используется для описания контекстно-свободных грамматик.
Отличается от БНФ большей выразительностью и ёмкостью.

- нетерминальные символы не обрамляются «<» и «>»
- терминальные символы выделяются кавычками или апострофами (“ или ‘);
- порождаемость (следование) указывается с помощью «=» или «::=»;
- символ «,» (запятая) используется в качестве конкатенации;
- выражение внутри {} повторяется 0 или более раз;
- выражение внутри [] является необязательным, т.е. выражение может присутствовать, а может и отсутствовать;
- для группировки выражений используются круглые скобки «(» и «)»;
- выбор организуется так же, как и в БНФ;

https://www.bottlecaps.de/rr/ui#_Choice – рисует диаграммы по описанной РБНФ.

Пример РБНФ

Описание идентификатора с помощью РБНФ:

```
буква = “a” | “b” | “c” | “d” | “e” | “g”
цифра = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”
идентификатор = буква {буква | цифра}
```

По итогу, выводы из данных правил можно сделать те же самые, что и в случае описания с БНФ.

Соответственно, для такой грамматики:
- abcd – валидный идентификатор;
- ec9 – валидный идентификатор;
- hje0 – невалидный идентификатор, т.к. в описании отсутствуют h и j;
- 2ab – невалидный идентификатор, т.к. исходя из описания, не может начинаться с цифры;
- a – валидный идентификатор.

Формальные грамматики

Пример: язык простых арифметических операций над целыми числами без знака.

G({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -}, {<формула>, <знак>, <число>, <цифра>}), P, <формула>)

P (набор правил, в формате РБНФ):

формула = формула знак формула | число
знак = “+” | “-”
число = цифра {цифра}
цифра = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

Соответственно по такой грамматике можно описать выражения по типу: 2 + 2 – 3, где мы бы рекурсивно спускались по правилам.

P.S. Заметьте, что здесь «формула», «знак», «число», «цифра» – являются **нетерминальными символами** (словами)**,** а знаки +/- и цифры от 0 до 9 – **терминальными символами** (по сути, алфавитом языка)

2 + 2 - 3

```
формула = формула знак формула | число
знак  = + | -
число = цифра {цифра}
цифра = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”
```

tl;dr: 2 + 2 - 3

Начинаем разбор с начала предложения: *2 + 2 - 3*
И видим, что нам явно подходит первая продукция "ф з ф"

1) формула = формула знак формула | число

Теперь мы рассматриваем *[2 + 2] - 3* и выбираем соответствующую продукцию

2) формула = формула знак формула | число

Теперь мы рассматриваем *[2] + 2 - 3* рекурсивно спускаясь по правилам

3) формула = формула знак формула | число

число = цифра {цифра}

цифра = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

Вернувшись обратно к **(2)**, мы рассматриваем *2 [+] 2 - 3*

4) формула = формула знак формула | число

знак = + | -

Вернувшись обратно к **(2)**, мы рассматриваем *2 + [2 - 3]*

5) формула = формула знак формула | число

Которая для нас предстаёт как продукция "ф з ф"

формула = формула знак формула | число

Соответственно, по аналогии с **(2)**, разбираем по отдельности *2*, *-* и *3*

6) формула = формула знак формула | число

число = цифра {цифра}

цифра = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

формула = формула знак формула | число

знак = + | -

пояснения опущены, для краткости

формула = формула знак формула | число

формула = формула знак формула | число

число = цифра {цифра}

цифра = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

Разберём выражение «2 + 2 – 3» в соответствии с имеющейся у нас грамматикой (описана в формате РБНФ).

Если вы обратите внимание, то заметите, что к примеру выражение «a + b * c» для такой грамматики будут некорректными, т.к.:

1) в грамматике не описаны такие терминалы как a, b, c и *;
2) в грамматике отсутствуют продукции (правила) с такими символами.

«ф з ф» – означает: формула знак формула.

P.S. на самом деле наша грамматика страдает неоднозначностью и левой рекурсией, но об этом в след. семестре.

Зачем всё это нужно? Для чего?

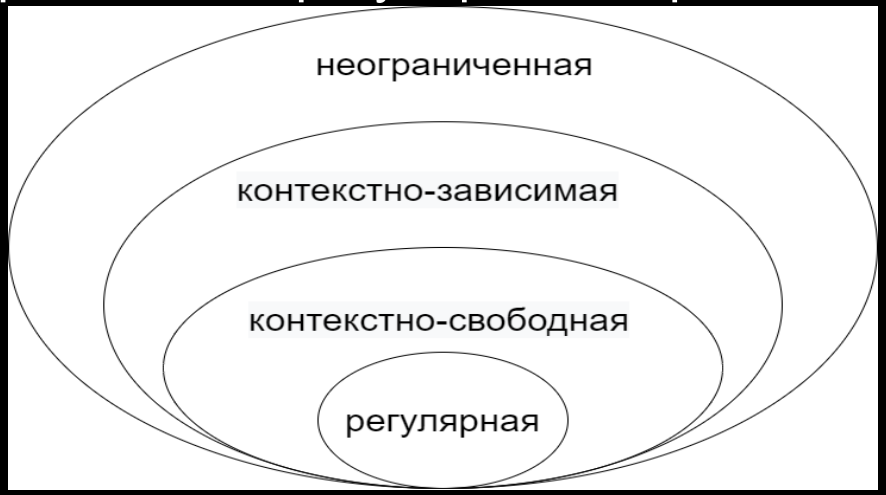
На основе грамматики конструируются языки программирования и компиляторы для них.

Имея качественно сформированную грамматику можно облегчить себе написание синтаксического анализатора и генератора кода.

Также, с помощью грамматики можно проверить принадлежность входной строки к некоторому языку/правилу/свойству. Например: проверять на «сложность» пароли пользователей или формировать таблицы с данными.

Типы грамматик

Ниже представлен перечень типов грамматик по Хомскому:
0) неограниченные – содержит в себе перечень остальных грамматик. Генерирует языки, распознаваемые машиной Тьюринга;
1) **контекстно-зависимые** – для выявления продукции необходимо учитывать **контекст** ($\sigma\alpha\tau \rightarrow \sigma\beta\tau$, где $\sigma\tau$ – контекст, какие-то нетерминалы) и **длину** продукции (количество символов правой части >= кол-ва символов в левой);
2) **контекстно-свободные** – **не зависит** от **контекста** (левая часть содержит всего 1 нетерминал);
3) регулярные – ограничивает число терминальных и нетерминальных символов на каждом шаге вывода. Пример такой грамматики: регулярные выражения (regex), например: [aA-zZ].



P.S. в основном в написании ЯП/компиляторов используется 2 тип грамматик (КС), поэтому тут и далее будет неявно подразумеваться она.

Трансляторы

Транслятор – это программа, которая **переводит программу на входном** языке **в** эквивалентную ей программу на **выходном** языке.

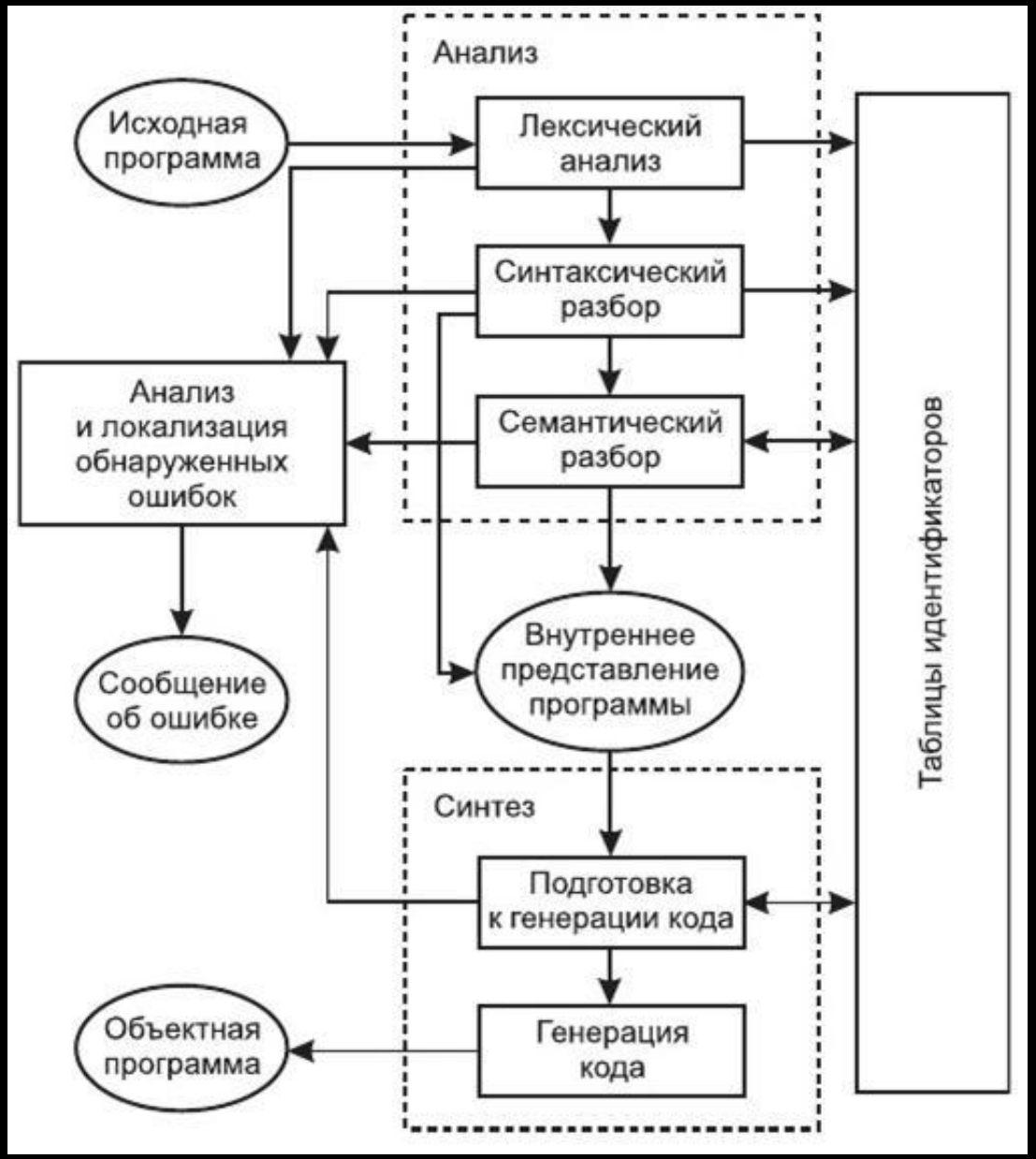
Компилятор – это транслятор, который **выполняет перевод** исходной **программы в** эквивалентную результирующую программу на языке **машинных команд или языке ассемблера**.

Интерпретатор – это программа, которая **воспринимает** исходную программу на входном языке и **выполняет** её.

Результат работы **компилятора** – **объектная программа (объектный код),** которая записана в объектный файл.
Объектная программа **не может выполняться** на компьютере! Это может делать исполняемая программа!

Перевод объектной программы в исполняемую выполняет – **линкер**.

Этапы компиляции



Более подробно каждый из этапов мы рассмотрим с вами в следующем семестре.

Лексический анализ (ЛА)

Лексема (лексическая единица языка) – это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своём составе другие структурные единицы языка.

Лексический анализатор (сканер) – это часть компилятора, которая читает исходную программу и выделяет в её тексте лексемы входного языка.

Основные функции сканера:
1) **исключение из текста** исходной программы комментариев, незначащих пробелов, символов табуляции и перевода строки;
2) **выделение лексем**: идентификаторов, констант (символьных, строковых, числовых), ключевых слов входного языка, знаков операций, разделителей.

Результат работы сканера:
1) **таблица лексем** – содержит все лексемы встреченные во входной программе;
2) **таблица идентификаторов** – содержит информацию об идентификаторах (переменных).

Алгоритм лексического анализатора

Общий алгоритм работы сканера:

1) выбрать очередной символ из входного потока и запустить нужный частный сканер;
2) запущенный сканер из входного потока выделяет символы, входящие в лексему, до обнаружения ограничивающего или ошибочного символа;
3) при успешном распознавании информация о лексеме заносится в таблицу лексем или таблицу идентификаторов; возврат к п.3;
4) при неуспешном распознавании – выдача ошибки, а дальше – зависит от реализации сканера: останов или попытка распознать следующую лексему.

Пример разбора арифметического выражения лексическим анализатором (сканером) представлен на следующем слайде.

P.S. под курсором подразумевается положение в считываемом файле/строке.

Имея следующую грамматику (часть токенов пропущена для сокращения):

```
формула = формула знак формула | число | идентификатор
знак  = + | -
число = цифра {цифра}
идентификатор = буква {буква | цифра}
буква = "a" | "b" | "c" | ... | "Z"
цифра = "0" | "1" | "2" | "3" | ... | "9"
```

Просканируем данное выражение: abc + 23 - a1

Сканер считывает первый символ (a) и сдвигает курсор вперёд

1) abc + 23 - a1

Слово ещё не закончилось (не было пробела), продолжаем сдвигать курсор

2) abc + 23 - a1

Сканер считывает символ (c) и сдвигает курсор вперёд

3) abc + 23 - a1

Сканер считывает символ (пробел), понимает, что слово закончилось, формирует лексему (идентификатор) и добавляет её в таблицы

4) abc + 23 - a1

Таблица идентификаторов (ТИ)

Имя	Номер строки	Значение
abc	1	0

Таблица лексем (ТЛ)

Имя	Номер строки	Токен	Ссылка на ТИ
abc	1	идентификатор	1

* - реализация таблиц может отличаться от действительности и содержать дополнительные столбцы/значения.

Сканер считывает символ (плюс) и сдвигает курсор далее. Т.к. следующий символ является пробелом (при след. считывании) то сканер формирует следующую лексему и добавляет её в таблицу лексем

5) abc + 23 - a1

Таблица идентификаторов (ТИ)

Имя	Номер строки	Значение
abc	1	0

Таблица лексем (ТЛ)

Имя	Номер строки	Токен	Ссылка на ТИ
abc	1	идентификатор	1
+	2	знак	--

Сканер считывает символ (2) и сдвигает курсор далее.

6) abc + 23 - a1

* - я опущу посимвольное считывание лексемы 23 для сокращения места

Сканер считывает символ (пробел), т.к. слово закончилось, формируется лексема (число) и добавляется в таблицу лексем

7) abc + 23 - a1

Таблица идентификаторов (ТИ)

Имя	Номер строки	Значение
abc	1	0
def	1	0

Таблица лексем (ТЛ)

Имя	Номер строки	Токен	Ссылка на ТИ
abc	1	идентификатор	1
+	1	знак	--
23	1	число	--

** - я опущу шаги связанные со знаком -, они работают по аналогии с +

Сканер считывает символ (1), т.к. следующим шагом сканер заканчивает проход по строке (т.к. она закончилась), формируем лексему (идентификатор) и добавляем её в таблицы

7) abc + 23 - a1

Таблица идентификаторов (ТИ)

Имя	Номер строки	Значение
abc	1	0
a1	1	0

Таблица лексем (ТЛ)

Имя	Номер строки	Токен	Ссылка на ТИ
abc	1	идентификатор	1
+	1	знак	--
23	1	число	--
-	1	знак	--
a1	1	идентификатор	2

Пример автомата для лексического анализа (ЛА)

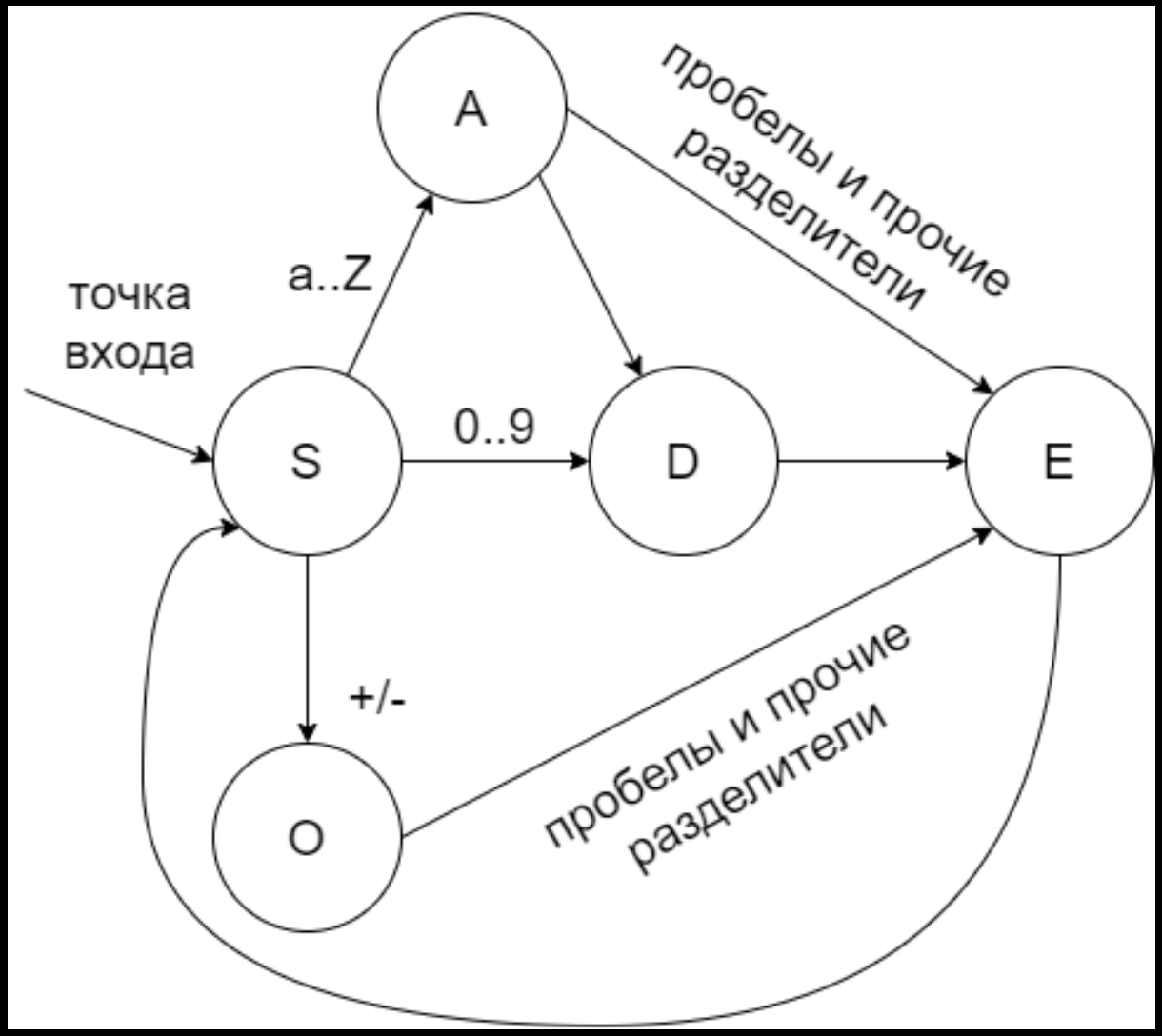


Таблица лексем

Содержит **все последовательно** «отсканированные» лексемы из входного файла/строки. Используется в последующих фазах компиляции.
Может быть реализована как в виде таблицы лексем, так и в виде списка.

Лексема должна обладать информацией об: имени считанной лексемы, её месторасположении (например: номер строки), токен (тип лексемы) и т.д.

В зависимости от реализации компилятора, информацию об идентификаторах либо выносят в отдельную таблицу (таблицу идентификаторов), либо же всё хранят в одной таблице и все остальные фазы компиляции работают только с ней.

В случаях, если формируют отдельную таблицу идентификаторов, то при сканировании лексемы-идентификатор, в ТИ заносят всю известную информацию об идентификаторе, а в ТЛ ссылку на ТИ.

Таблица идентификаторов

Хранит в себе только идентификаторы и всю информацию о них: имя переменной, месторасположение (имя блока, номер строки), значение (если неизвестно – либо зануляем, либо указываем явно, например: «?»/undefined/null), и т.д.

Обычно, на этапе ЛА нам не известно значение переменной. Мы узнаём о нём уже на этапе синтаксического или семантического анализа.
Соответственно, в отличии от ТЛ, ТИ может изменяться на всём этапе компиляции.

Зачастую реализуется непосредственно как таблица (или словарь).

Ссылки

- рисует диаграммы по заданной РБНФ;
- Вики о формальной грамматике;
- Немного теории про грамматики по Хомскому;
- Вики про БНФ;
- Вики про РБНФ;
- Вики про ДБНФ;
- ещё несколько примеров РБНФ;
- немного Вики про трансляторы, компиляторы и интерпретаторы;
- (осторожно) немного про неоднозначность грамматики;
- (осторожно) ещё более "теоретично" про неоднозначные грамматики;
- валидатор регулярных выражений (regexp);
- немного про регулярные выражения с практической точки зрения;
- и ещё немного про регулярочки;