Классическая теория компиляторов

Лекция 1

Более чем полувековая история

- 1957 г., Бэкус. Первый компилятор языка Фортран.
- Начало формальной теории компиляторов - стимул развитию математической лингвистики и методам ИИ, связанных с естественными и искусственными языками.
- ТК – формальное направление ИТ (*Искусство* $\rightarrow$ *Наука* $\rightarrow$ *Дилетантизм (?)*)

Перспективные направление ТК

- Создание новых языков
- Автоматизация решения задач
- Параллельные вычисления

Содержание, цели и задачи

Цель курса – освоить основы построения компиляторов.

- Грамотное написание интерпретаторов

- Формальные грамматики и языки
- Формирование объектного кода
- Оптимизация
- Компоновка
- Язык ПРОЛОГ

Литература

- Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. В 2-х томах. Том 1. М.: Мир, 1978. –616с.
- Ахо А., Моника С. Лам М., Рави Сети Р., Ульман Дж. Компиляторы. Принципы, технологии и инструментарий. – Вильямс, 2003. – 768 с.
- Грис Д. Конструирование компиляторов для цифровых вычислительных машин /Пер.с англ. –М.: Мир, 1975.
- Донован Дж. Системное программирование /Пер.с англ. –М.: Мир, 1975. –540с.
- Ин Ц., Соломон Д. Использование Турбо-пролога. –М.: Мир, 1993.
- Керниган Б.В., Пайк Р. UNIX – универсальная среда программирования /Пер.с англ. Березко, Иващенко. Под ред. М.И.Белякова –М.: Финансы и статистика, 1992. –304 с.
- Марселлус Д. Программирование экспертных систем на Турбо-прологе. –М.: Финансы и статистика, 1994, –254с.
- Хантер Р. Проектирование и конструирование компиляторов. –М.: Финансы и статистика, 1984.

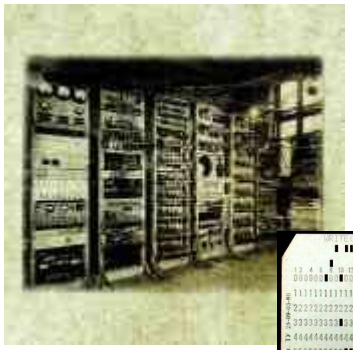
Карпов В.Э. Классическая теория компиляторов, Москва: , 2011, -91с.
http://rema44.ru/resurs/students/karpov/

Терминология

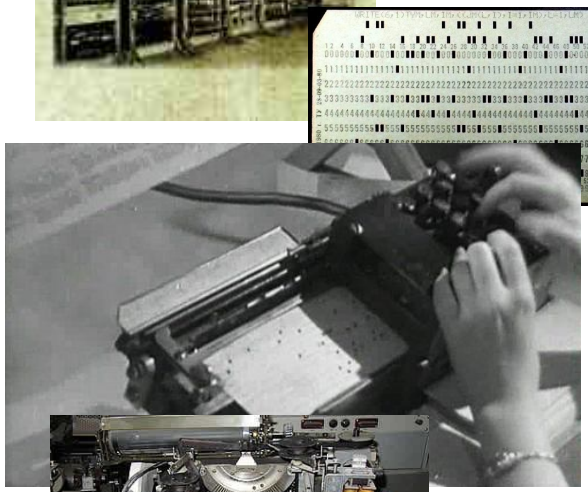
- *Транслятор.* Программа, которая переводит текст исходной программы в эквивалентную объектную программу. Если объектный язык представляет собой автокод или некоторый машинный язык, то транслятор называется *компилятором*.
- *Ассемблер* – это программа, которая переводит исходную программу, написанную на автокоде или на языке ассемблера (что, суть, одно и то же), в объектный (исполняемый) код.
- *Интерпретатор* принимает исходную программу как входную информацию и выполняет ее. Интерпретатор не порождает объектный код.

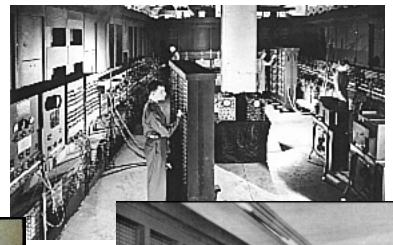
Языковые парадигмы. «Снизу-вверх». Шаг 1.

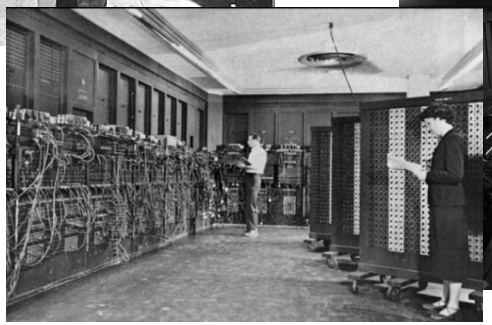
Последовательная автоматизация программирования



“Марк-1”

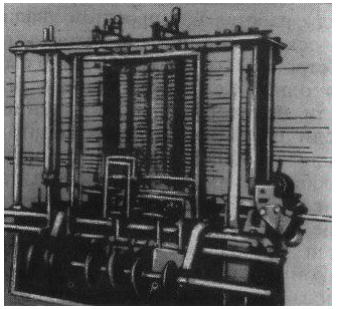


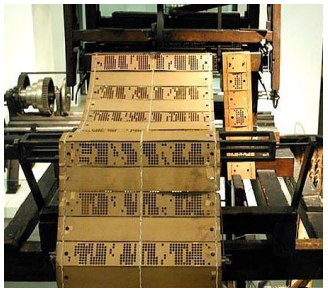




“ЭНИАК”

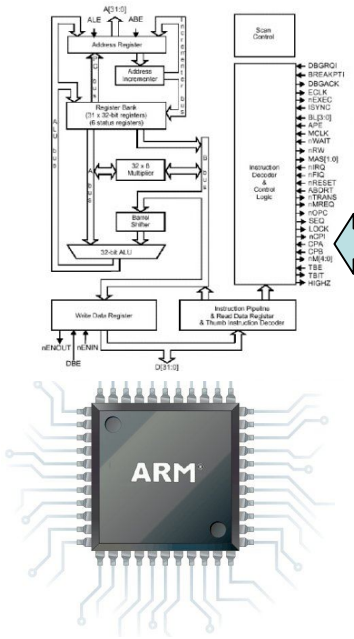
Аналитическая машина Чарльза Беббеджа





Языковые парадигмы. «Снизу-вверх». Шаг 2.

- Считывание команд, как исполнение программы.
- Ассемблер. Схемотехнический уровень.
- Библиотеки
- «Обертка» машинных инструкций.



Программатор

```
:0A0000002EC0FECFFDCFFCCFFBCFDA
:10000A00FACFF9CFF8CFF7CFF6CFF5CFF4CFF3CFBA
:10001A00F2CFF1CFF0CFEFCFEECFEDCFECCFEBCFEA
:10002A00EACFE9CFE8CFE7CFE6CF1027E8036400AD
:10003A000A000100001000011000010071776572CA
:10004A007479000001002D4E414E000001000002AB
:10005A004E000000F894EE27EFBBF1E0F5BFE5BFD4
:10006A00F8E1A895A4B7A77FA4BFF0936000E09336
:10007A0060008DE0A2E0BB27ED938A95E9F780E066
:10008A0094E0A0E0B1E0ED930197E9F7E6E5F0E04E
:10009A0085919591009761F0A591B5910590159007C
:1000AA00BF01F00105900D920197E1F7FB01F0CF36
```

HEX

```
CPI  R26,LOW(@0)
LDI  R30,HIGH(@0)
CPC  R27,R30
LDI  R30,BYTE3(@0)
CPC  R24,R30
LDI  R30,BYTE4(@0)
CPC  R25,R30
```

ASM

```
CPI  R26,LOW(@0)
LDI  R30,HIGH(@0)
CPC  R27,R30
LDI  R30,BYTE3(@0)
CPC  R24,R30
LDI  R30,BYTE4(@0)
CPC  R25,R30
```

MACRO

Библиотеки

```
void main(void)
{ int i;
for(i=0;i<10;i++)
{
  printf("qwerty");
  i=i+1;
  i++;
}}
```

C

Транслятор

Языковые парадигмы. «Сверху-вниз»

Сверхвысокоуровневые языки программирования (VHLL — very high-level programming language)

Языки программирования с высоким уровнем абстракции.

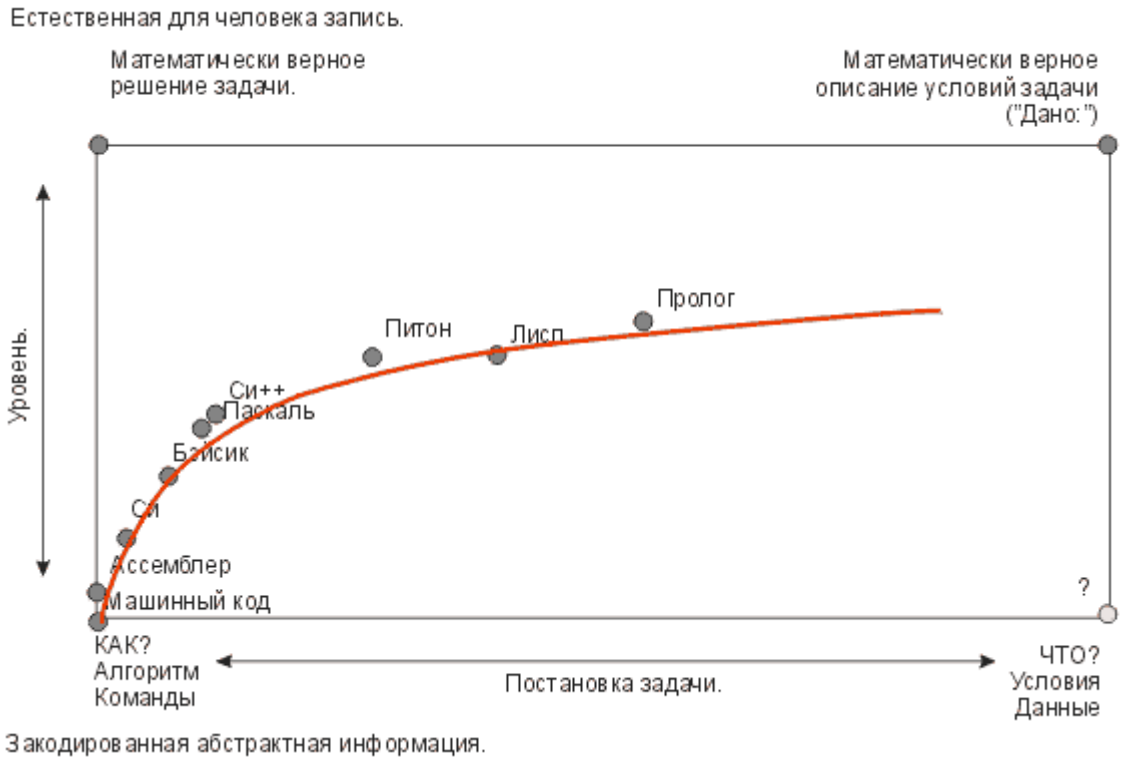
Основной принцип – **декларативность**, т.е. описание не того, «как нужно сделать», а «что нужно сделать».

- Современные скриптовые языки
- Декларативные
- Функциональные
- Предметно-ориентированные языки (для специфических приложений и задач) со своими особенностями синтаксиса.

Уровни языков программирования

Уровень языка программирования показывает, насколько язык близок к естественной для человека записи.

- **ООП.** Основа - процедурная модель программирования.
- **Функциональное программирование.** Программа состоит из совокупности функций. Лаконизм. Сложность реализации. Малое быстродействие.
- **Логическое программирование.** Основа – формальная логика. Декларативность.

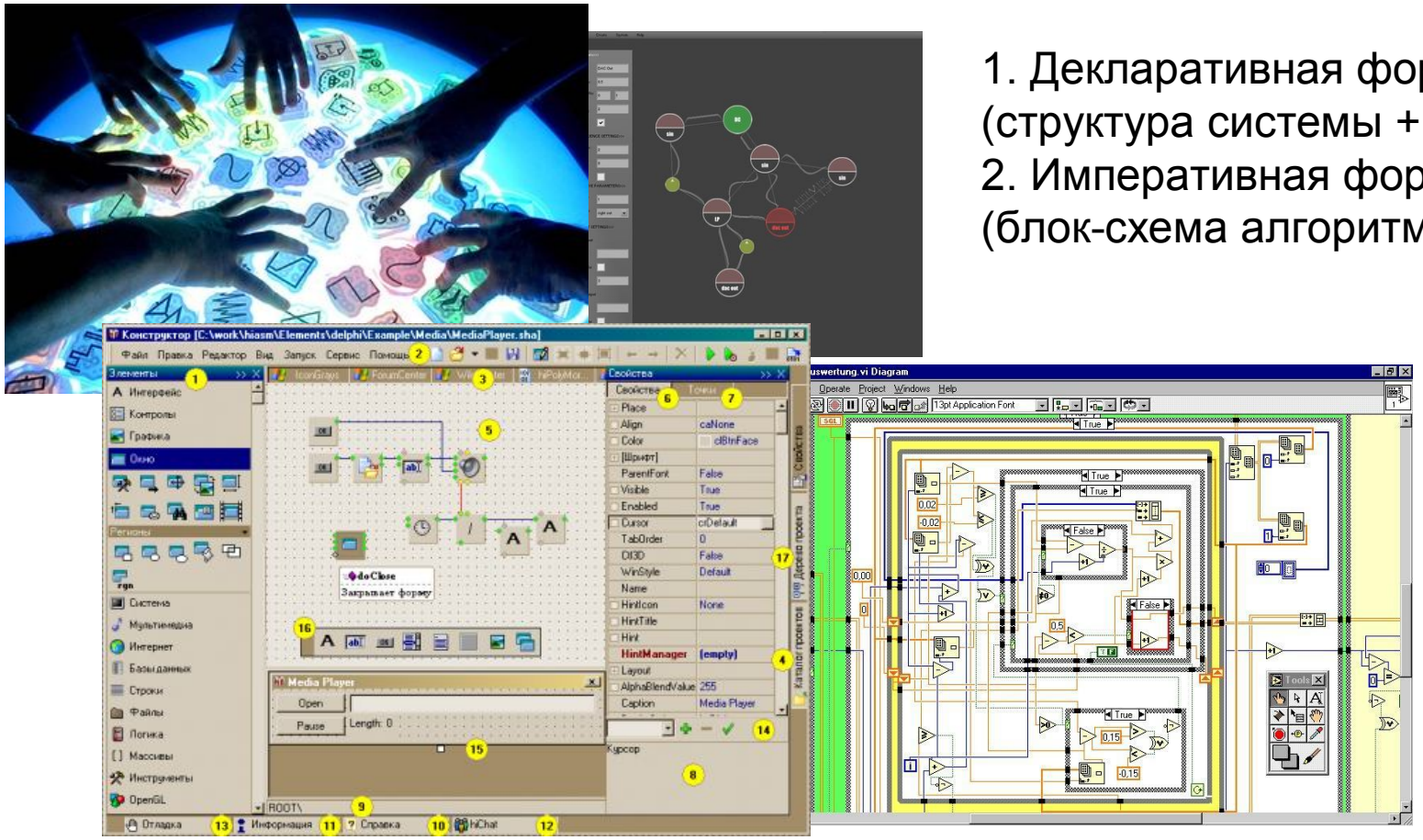


Основная парадигма развития языков – **максимальная автоматизация** процесса решения задачи.

Идеал – избавиться от программиста вообще.

«Визуальное» программирование

Еще одна форма автоматизации
Визуальная форма выражения способа решения задачи

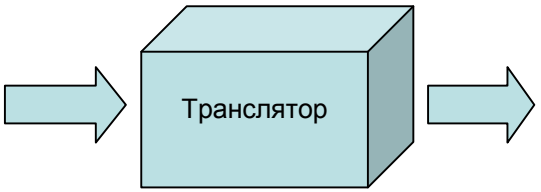


1. Декларативная форма (структура системы + ...)
2. Императивная форма (блок-схема алгоритма + ...)

Постановка задачи

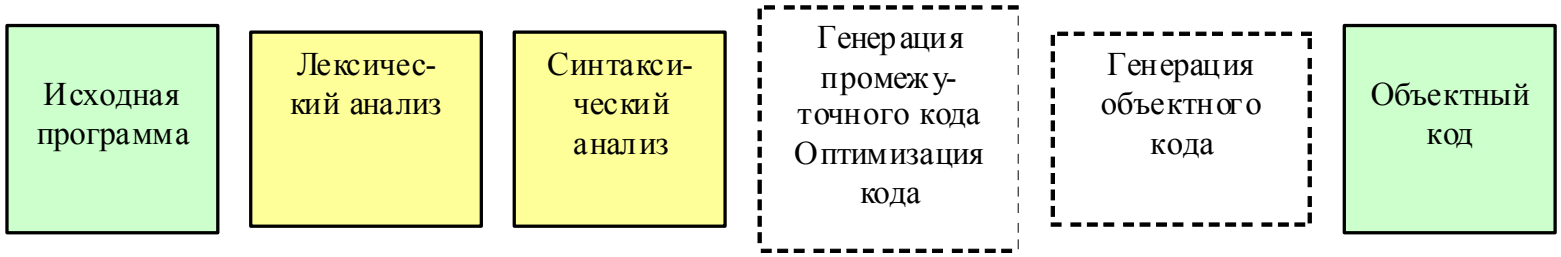
- Вход: исходный текст
- Выход: объектный код

```
// ...
void main(void)
{ int i;
for(i=0;i<10;i++)
{
  printf("qwerty");
  i=i+1;
  i++;
}}
```




```
:0A0000002EC0FECFFDCFFCCFFBCFDA
:10000A00FACFF9CFF8CFF7CFF6CFF5CFF4CFF3CFBA
:10001A00F2CFF1CFF0CFEFCFEECFEDCFECCFEBCFEA
:10002A00EACFE9CFE8CFE7CFE6CF1027E8036400AD
:10003A000A000100001000011000010071776572CA
:10004A007479000001002D4E414E000001000002AB
:10005A004E000000F894EE27EFBBF1E0F5BFE5BFD4
:10006A00F8E1A895A4B7A77FA4BFF0936000E09336
:10007A0060008DE0A2E0BB27ED938A95E9F780E066
:10008A0094E0A0E0B1E0ED930197E9F7E6E5F0E04E
:10009A0085919591009761F0A591B591059015907C
:1000AA00BF01F00105900D920197E1F7FB01F0CF36
```

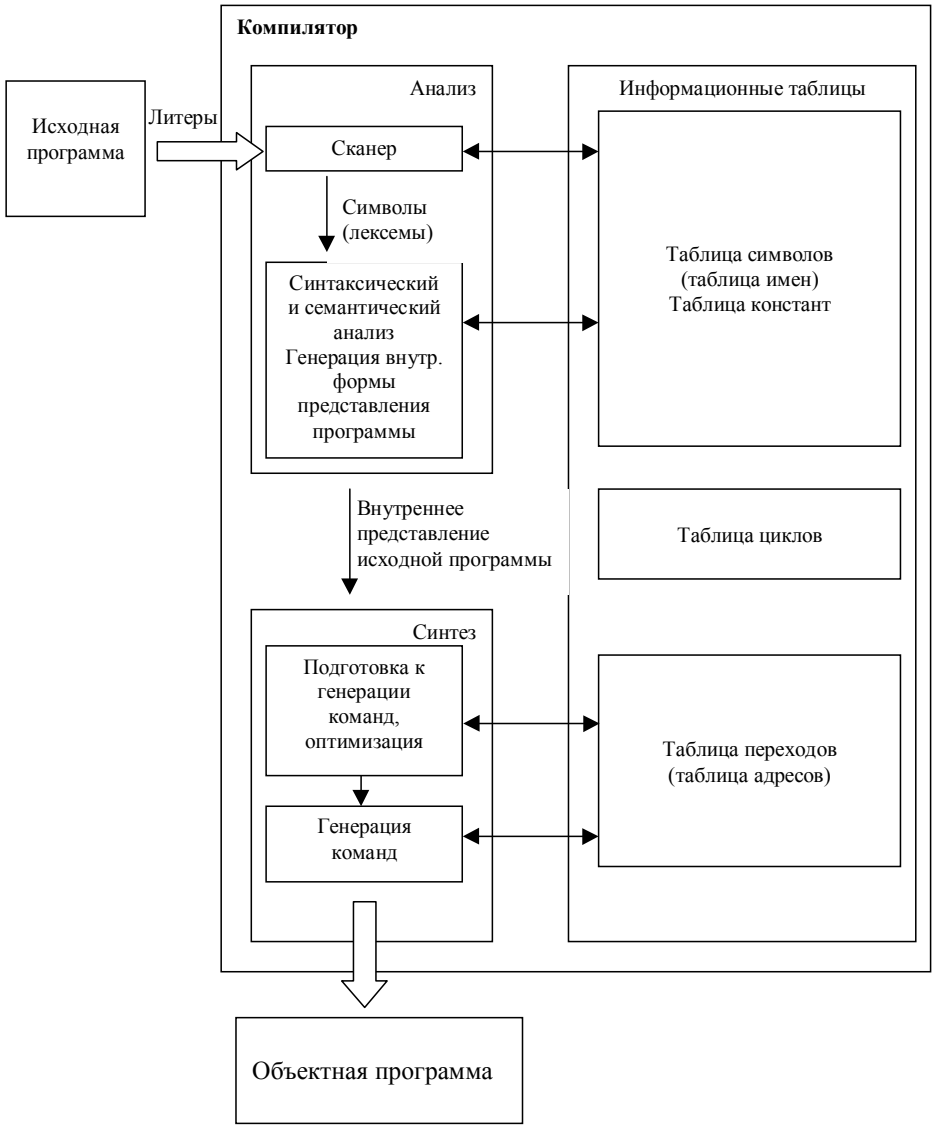
```
CPI  R26,LOW(@0)
LDI  R30,HIGH(@0)
CPC  R27,R30
LDI  R30,BYTE3(@0)
CPC  R24,R30
LDI  R30,BYTE4(@0)
CPC  R25,R30
```



Логическая структура компилятора

```
// ...
void main(void)
{ int i;
for(i=0;i<10;i++)
{
  printf("qwerty");
  i=i+1;
  i++;
}}
```

```
1.          ; 0000 0023 // Начало цикла
2.          ; 0000 0024 for(i=0;i<10;i++)
3.          ;       i -> R16,R17
4.                  __GETWRN 16,17,0
5.          _0x4:
6.                  __CPWRN 16,17,10
7.                  BRGE _0x5
8.          ; 0000 0025 {
9.          ; 0000 0026   printf("qwerty");
10.                 __POINTW1FN _0x0,0
11.                 RCALL SUBOPT_0x0
12.                 LDI  R24,0
13.                 RCALL _printf
14.                 ADIW R28,2
15.         ; 0000 0027   i=i+1;
16.                 __ADDWRN 16,17,1
17.         ; 0000 0028   i++;
18.                 __ADDWRN 16,17,1
19.         ; 0000 0029 }
20.                 __ADDWRN 16,17,1
21.                 RJMP _0x4
22.         _0x5:
23.         ; 0000 002A
24.         ; 0000 002B //Конец цикла
25.         ; 0000 002C
```



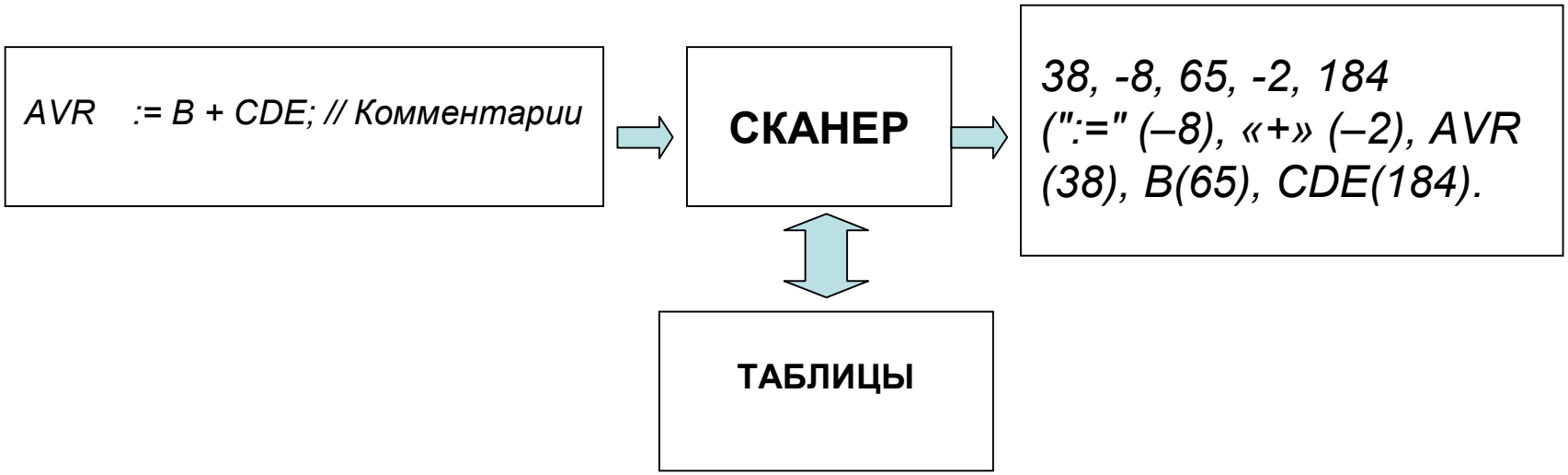
Лексический анализатор (сканер)

Вход: цепочка символов некоторого алфавита. При этом некоторые комбинации символов рассматриваются сканером как единые объекты. Например:

- один или более пробелов заменяются одним пробелом;
- ключевые слова (вроде BEGIN, END, INTEGER и др.);
- цепочка символов, представляющая константу;
- цепочка символов, представляющая идентификатор (имя);

ЛА группирует определенные терминальные символы (т.е. входные символы) в единые синтаксические объекты – *лексемы*.

Лексема: <тип_лексемы, значение>.



Сканер. Все на так просто.

Задача выделения лексем из входного потока зависит от структуры языка.

- "567AB" = "567AB" ?
- "567AB" = "567" + "AB" ?

Прямые (ПЛА) и *непрямые* (НЛА).

- **ПЛА** определяет лексему, расположенную непосредственно справа от текущего указателя, и сдвигает указатель вправо от части текста, образующей лексему (ПЛА определяет тип лексемы, которая образована символами справа от указателя).
- **НЛА** определяет, образуют ли знаки, расположенные непосредственно справа от указателя, лексему этого типа. Если да, то указатель передвигается вправо от части текста, образующей лексему. Для НЛА *заранее* задается тип лексемы, и он распознает символы справа от указателя и проверяет, удовлетворяют ли они заданному типу.

ПЛА сложнее НЛА.

ПЛА более распространен в современных ЯП (это м.б. видно по внешнему виду фраз языка).

Пример НЛА. В языке *Фортран* игнорируются пробелы.

Дано: "DO5I=1,10 …"

- "DO5I" = "DO5I " ?
- "DO5I" = "DO " + "5 " + "I" ?

Сканер. Таблица имен

Механизм работы с таблицами должен обеспечивать:

- быстрое добавление новых идентификаторов и сведений о них;
- быстрый поиск информации (*хеш-функции*).

№ элемента	Идентификатор	Дополнительная информация (тип, размер и т.п.)
1	ABC	идент., тип = "строка"
…	…	…
N	103	константа, число

```
/*   …   */
int A;
A = B + 3.1415;
…
```

№ элемента	Имя	Дополнительная информация (тип, размер и т.п.)
1	A	идент., тип = ?; значение = ?
2	B	идент., тип = ?; значение = ?
3	3.1415	константа; тип = float; число; значение = 3.1415
…		

Синтаксический и семантический анализ

- Синтаксический анализ – это процесс, в котором исследуется цепочка лексем и устанавливается, удовлетворяет ли она структурным правилам языка (синтаксису). Это – самая сложная часть компилятора.
- *Синтаксический анализатор:*
  - расчленяет исходную программу на составные части, формирует ее внутреннее представление,
  - заносит информацию в таблицу символов и другие таблицы.
  - производит полный синтаксический и, по возможности, семантический контроль программы.

Внутренние формы представления программы

- *Инфиксная* форма

A = B-C*(D+E^3)

Привычно, но неудобно

- *Польская форма* записи
  - *постфиксная*
  - *префиксная*.
- *Дерево*
- *Тетрады*

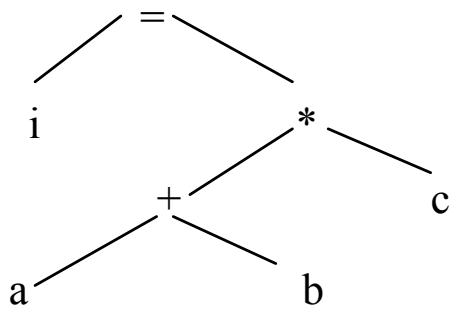
ABCDE3^+*-=

Непривычно, но удобно

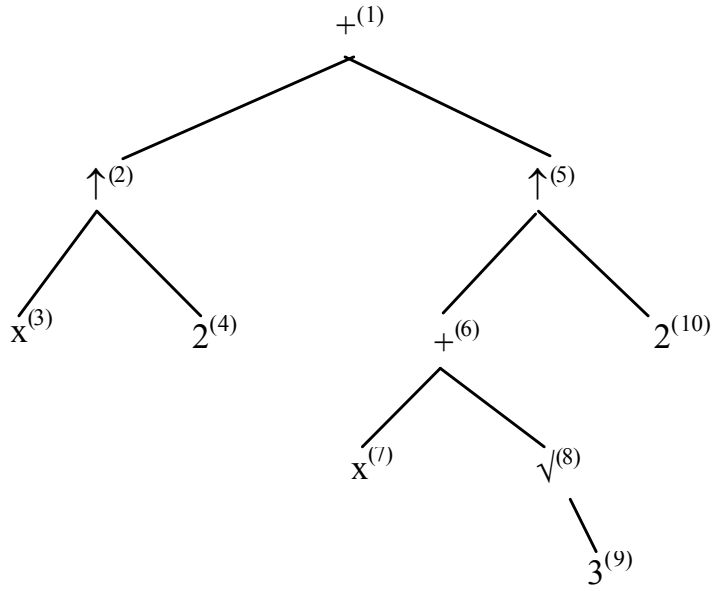
Дерево

- У каждого элемента дерева может быть только один “предок”. Дерево “читается” снизу вверх и слева направо.
- Дерево – это удобная математическая абстракция.
- На практике дерево можно реализовать в виде списковой структуры.

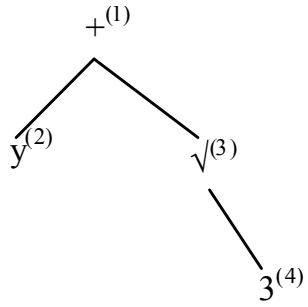
$E_{инф}$: i=(a+b)*c



$E_{инф}$: $x^2+(x+\sqrt{3})^2$



$E_{инф}$: $y+\sqrt{3}$

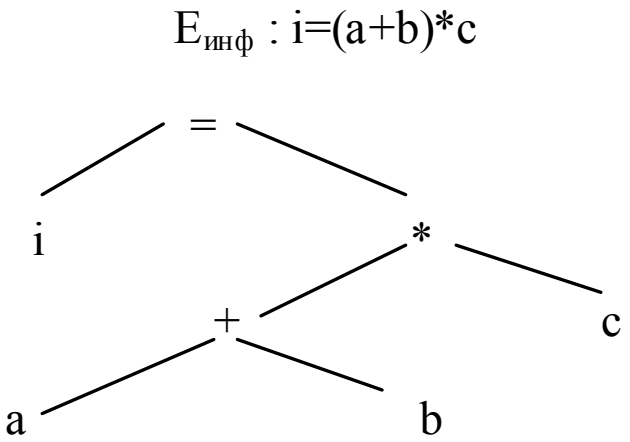


Польская форма записи

Инфиксная форма: оператор расположен *между* операндами: "a+b"

Польская форма

- постфиксная форма: оператор расположен *после* операндов (то же выражение выглядит как "a b + ");
- префиксная форма: оператор расположен *перед* операндами ("+ a b").



$E_{инф}$: "i=(a+b)*c"

$E_{постф}$: "iab+c*=".

Удобно расписывается по дереву

Тетрады

- Четверка: код операции, приемник и два операнда

<OP, RES, A1, A2>

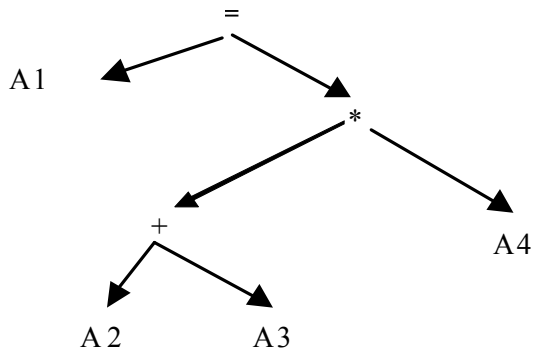
- Если требуется не два, а менее операторов, то в этом случае тетрада называется *вырожденной*

<OP, RES, A, _>

Исх. выражение	Код	Приемник	Операнд 1	Операнд 2
a+b→T1	+	T1	a	b
T1+c→T2	*	T2	T1	c
i=T2	=	I	T2	(вырожденная тетрада)

Примеры

"A1=(A2+A3)*A4"



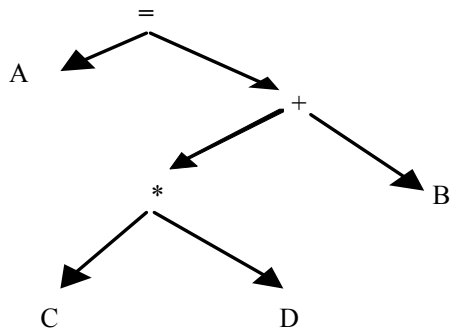
+, A2, A3, T1

*, T1, A4, T2

=, T2, A1

"A1 A2 A3 + A4 * ="

"A = B+C*D"



*, C, D, T1

+, B, T1, T2

=, T2, A

"A B C D * + ="

Вычисление польской формы

Просматриваем последовательно символы входной цепочки.
Если символ - операнд, то
    помещаем его в стек и читаем дальше.
Кесли
Если символ является оператором, то
  извлекаем из стека нужное кол-во операндов,
  производим операцию
  помещаем результат обратно в стек.
Кесли

```
while TRUE do
begin
case gettype(P[n]) of
            operand:
                    Push(P[n]);
            operator:
                    arg1 := Pop();
                    arg2 := Pop();
                    Push(arg1 @ arg2);
            else: error();
    endcase
    n := n+1
end
```

Стек

$E_{инф}$: (A+B)*C

$E_{пост}$: AB+C*