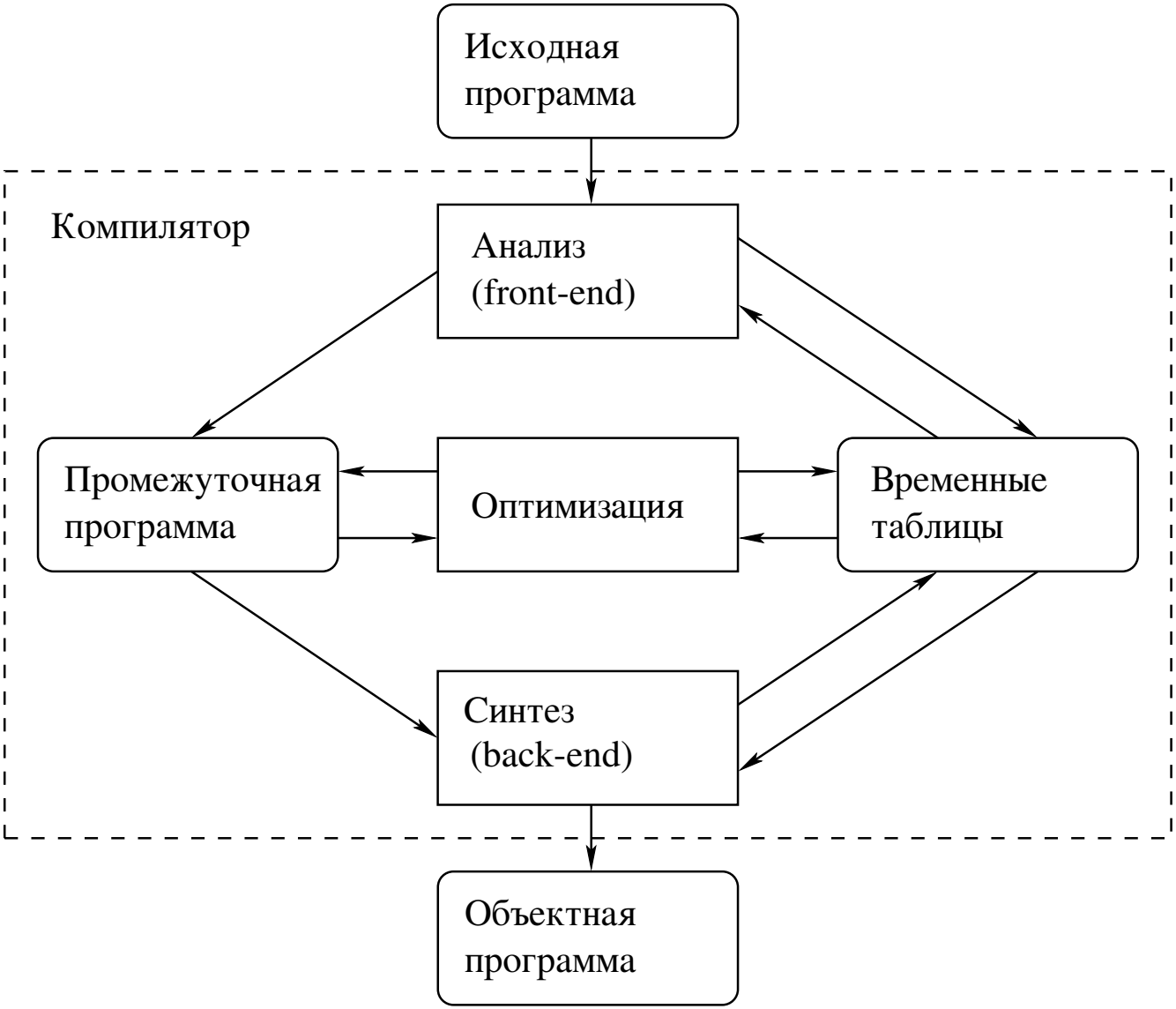
Лекция 2

Фазы компиляции

§5. Общая схема компиляции



Стадии компиляции:
– анализ;
– оптимизация;
– синтез.

Стадия анализа не зависит от целевого языка. Во время этой стадии могут порождаться сообщения об ошибках.

Если компиляция программы продолжается после обнаружения в ней ошибки, говорят, что компилятор выполняет *восстановление при ошибках* (error recovery).

Стадия оптимизации зависит только от языка промежуточного представления программы.

Стадия синтеза почти не зависит от исходного языка.

Стадии анализа, оптимизации и синтеза – это сложные транслирующие преобразования, поэтому их рассматривают как композиции более простых преобразований – фаз компиляции (compilation phases).

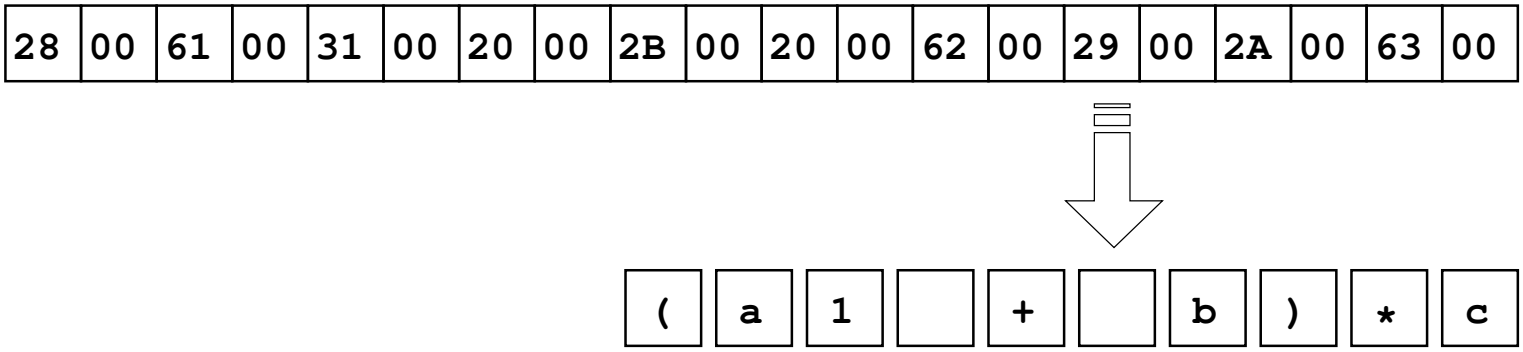
Фазы анализа:
– чтение входного потока;
– лексический анализ (линейный анализ, сканирование);
– синтаксический анализ (иерархический анализ, разбор);
– семантический анализ;
– генерация промежуточного представления.

Фазы оптимизации сильно зависят от промежуточного представления.

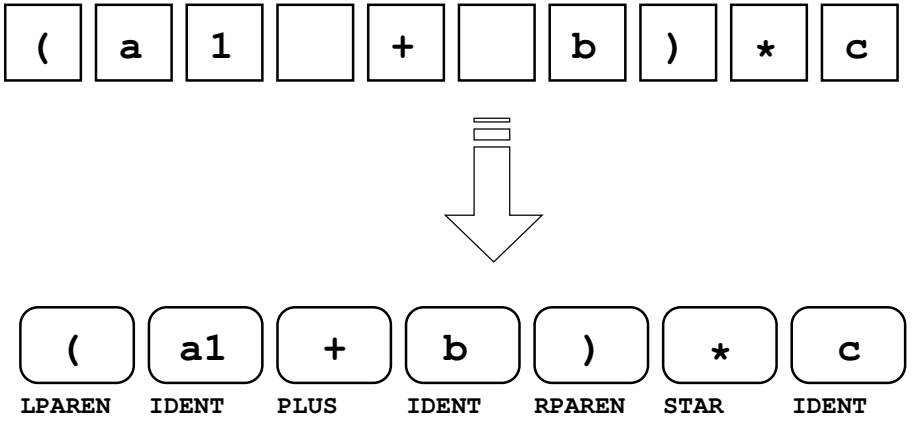
Фазы синтеза:
– распределение памяти;
– генерация кода на целевом языке;
– оптимизация, зависящая от целевого языка (постобработка).

§6. Фазы анализа

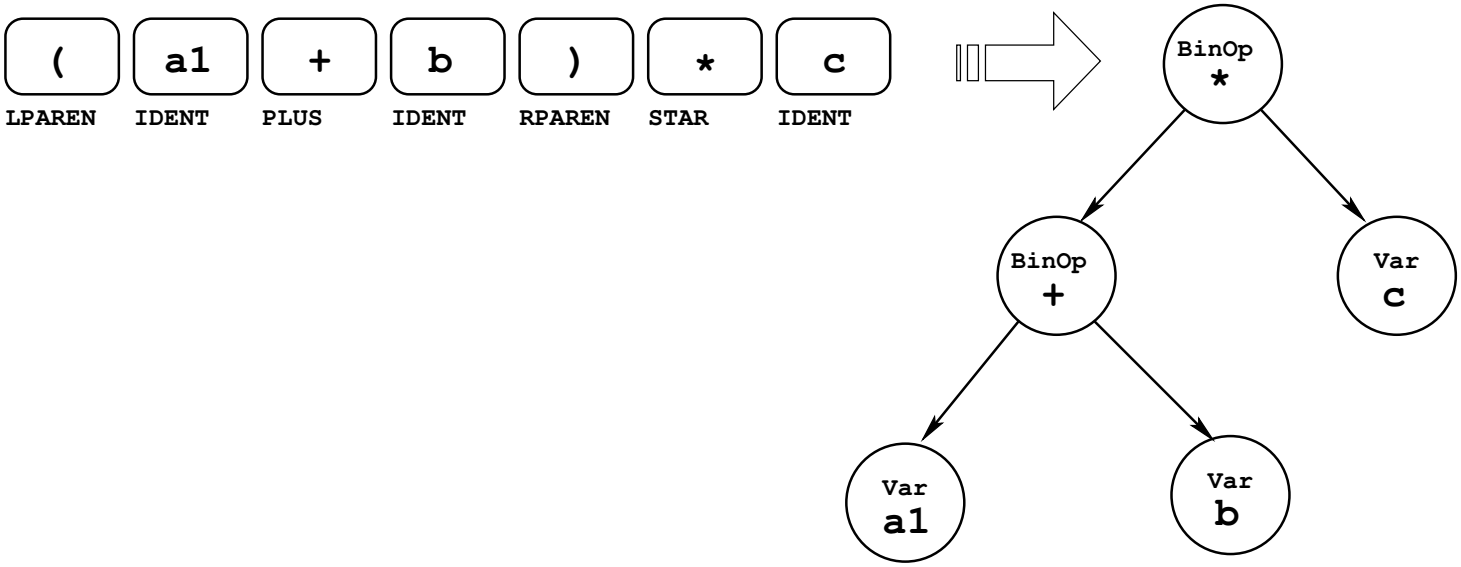
Определение. *Чтение входного потока* (input stream reading, character handling) – это фаза компиляции, осуществляющая преобразование образа текста в последовательность кодовых точек во внутреннем для компилятора стандарте кодирования текста.



Определение. *Лексический анализ* (lexical analysis, scanning) – это фаза компиляции, объединяющая последовательно идущие во входном потоке кодовые точки в группы, называемые *лексемами* исходного языка (lexems).



Определение. *Синтаксический анализ* (syntax analysis, parsing) – это фаза компиляции, группирующая лексемы, порождаемые на фазе лексического анализа, в синтаксические структуры.



Широко используются генераторы лексических и синтаксических анализаторов. Например, lex и yacc.

Определение. *Абстрактный синтаксис* (abstract syntax) – упрощённая грамматика языка, в которой отсутствует информация, гарантирующая построение уникальных деревьев вывода.

Пример. Абстрактный синтаксис арифметических выражений.

```
Expr    ::= Expr BinOp Expr
          | UnOp Expr
          | Var
          | Number.
BinOp   ::= "+" | "-" | "*" | "/".
UnOp    ::= "+" | "-".
```

Абстрактный синтаксис – противоположность конкретного синтаксиса.

Пример. Конкретный синтаксис арифметических выражений.

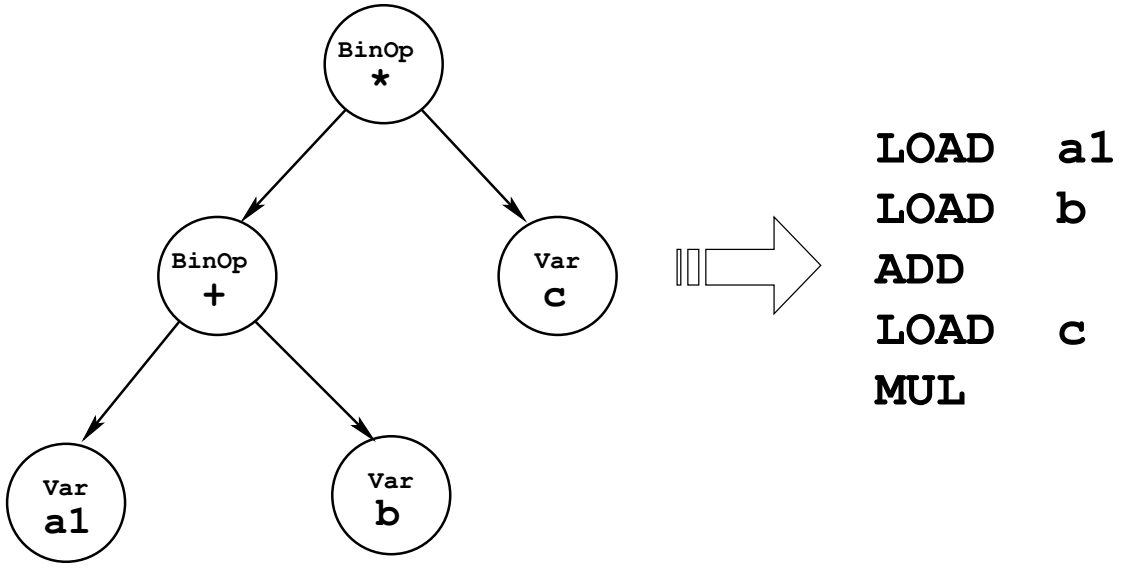
```
Expr    ::= Expr AddOp Term
          | AddOp Term
          | Term.
Term    ::= Term MulOp Factor
          | Factor.
Factor  ::= "(" Expr ")"
          | Var
          | Number.
MulOp   ::= "*" | "/".
AddOp   ::= "+" | "-".
```

Определение. *Семантический анализ* (semantic analysis) – это фаза компиляции, выполняющая проверку синтаксического дерева на соответствие его компонентов контекстным ограничениям.

Под контекстными ограничениями мы будем понимать такие вещи, как правила видимости идентификаторов, проверку типов выражений и т.п.

При семантическом анализе накапливается информация о типах для генерации кода.

Определение. *Генерация промежуточного представления* (intermediate code generation) – это фаза компиляции, выполняющая перевод синтаксического дерева в форму, удобную для последующей оптимизации и генерации кода.



§7. Фазы синтеза

Определение. *Распределение памяти* – это фаза компиляции, связанная с переносом структур данных, определённых в промежуточном представлении программы, в модель данных целевого языка.

Определение. *Генерация кода* (code generation) – это фаза компиляции, выполняющая перевод программы из промежуточного представления в целевой язык.

Определение. *Постобработка* (postprocessing) – это фаза компиляции, связанная с оптимизацией объектной программы, полученной в результате генерации кода.

§8. Группировка фаз компиляции

Фазы компиляции могут работать последовательно или параллельно. При этом возможны сложные сочетания последовательного и параллельного выполнения.

Определение. *Проход* (pass) – это выполнение группы параллельно работающих фаз.

Проход может быть реализован в виде отдельной программы.

Параллелизм фаз в рамках прохода означает, как правило, что эти фазы работают в режиме обменивающихся данными сопрограмм (то есть реального параллелизма нет, и фазы выполняются поочерёдно).

Если при компиляции происходит последовательное выполнение n проходов, то говорят, что компиляция – n-проходная.

Преимущества однопроходной компиляции:
– отсутствие больших промежуточных структур данных в памяти (временных файлов на диске);
– высокая скорость;
– сообщения об ошибках порождаются в правильном порядке.

Недостатки однопроходной компиляции:
– трудность организации (псевдо)параллельного выполнения фаз (невыразимость (псевдо)параллельного выполнения при использовании для разработки компилятора функциональных языков, необходимость обратных поправок при генерации инструкций перехода (back-patching), и т.д.);
– негативное влияние на исходный язык
(опережающие объявления и т.д.);
– невозможность глобальной оптимизации программы.