

Введение в компиляцию

Kirill Yukhin
19.09.2018

Компилятор и целевая программа

Исходная программа



Компилятор



Целевая программа

Входные данные

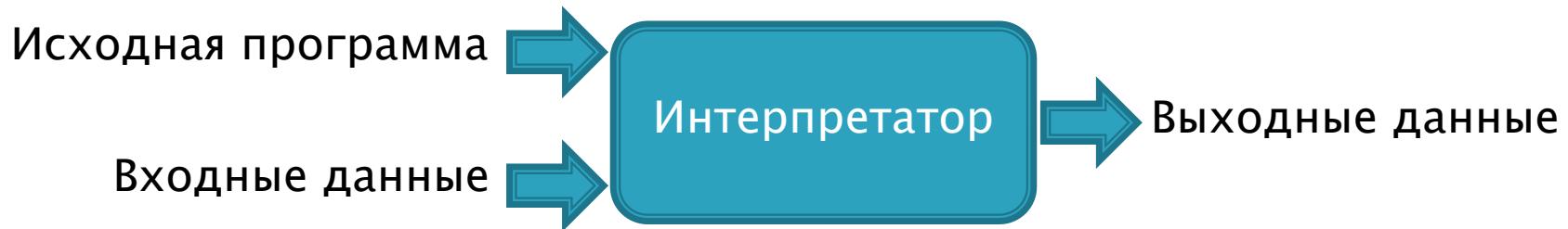


Целевая
программа



Выходные данные

Интерпретатор



В сравнении с компилятором:



Больше способностей к диагностике ошибок



Работает медленнее

Гибридный компилятор

Исходная программа



Транслятор

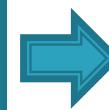


Промежуточная программа

Входные данные

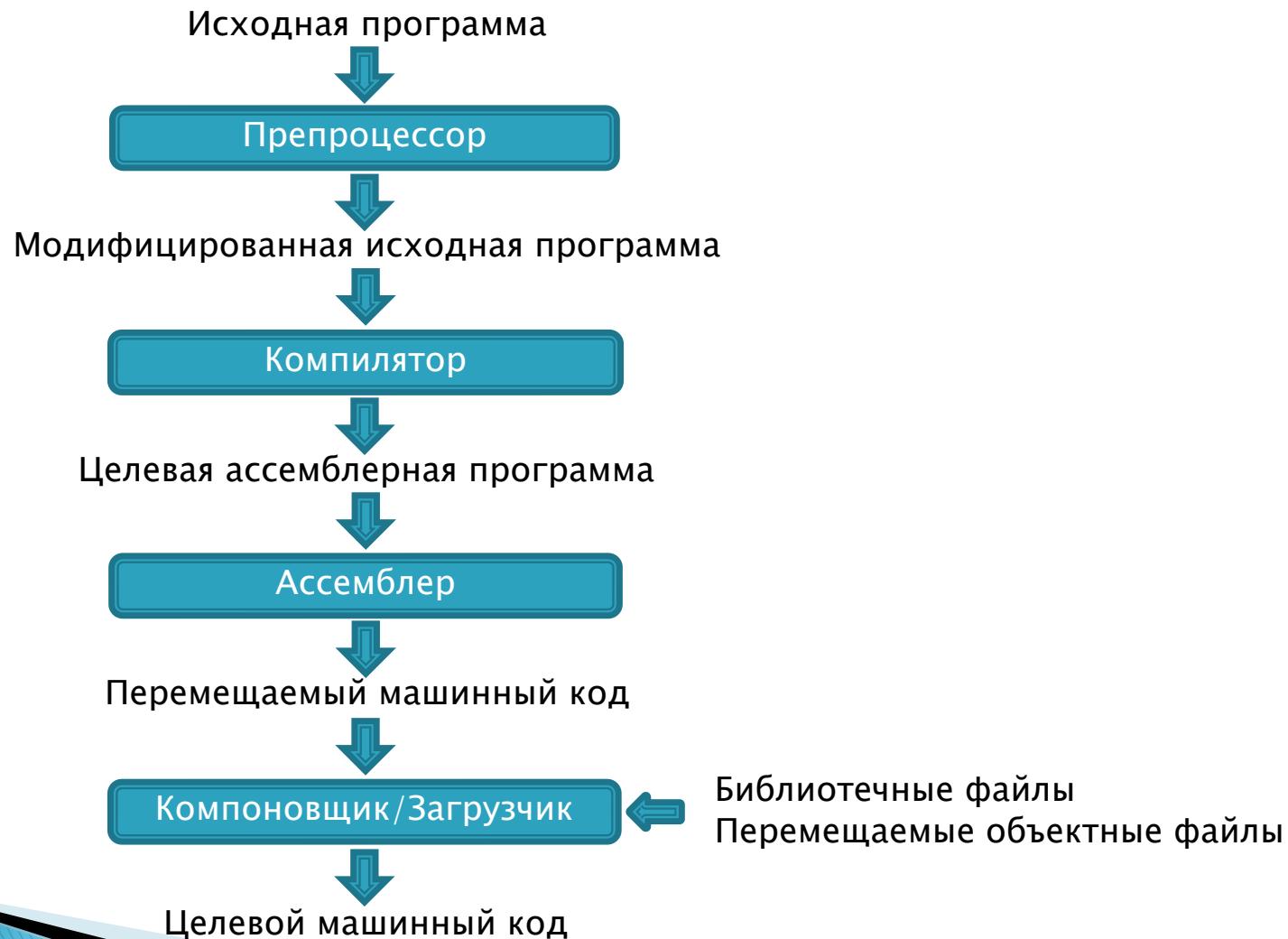


Виртуальная
машина



Выходные данные

Система обработки языка



High-level структура компилятора

Анализ
(front end)

- Чтение программы
- Построение промежуточного представления
- Поиск ошибок
- Сбор информации об исходной программе в таблицу символов

Синтез
(back end)

- Построение целевой программы на основе промежуточного представления и таблицы символов

Пофазная структура компилятора

Таблица
СИМВОЛОВ



Лексический анализ

Группировка символов в лексемы

Построение выходного токена для
каждой лексемы:
<имя_токена, значение_атрибута>

Таблица символов

1	position	...
2	initial	...
3	rate	...
...		

$$\text{position} = \text{initial} + \text{rate} * 60$$

Лексема	Токен
position	<i><id, 1></i>
=	<i><=></i>
initial	<i><id, 2></i>
+	<i><+></i>
rate	<i><id, 3></i>
*	<i><*></i>
60	<i><60></i>

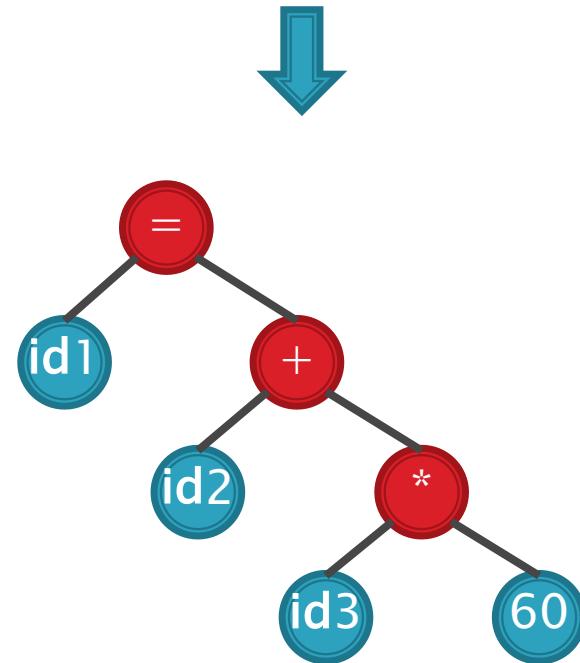


<id,1> <=> <id,2> <+> <id,3> <> <60>*

Синтаксический анализ

```
<id,1> <=> <id,2> <+> <id,3> <*> <60>
```

Создание синтаксического дерева.
Внутренний узел – операция
Дочерние узлы – аргументы



Семантический анализ

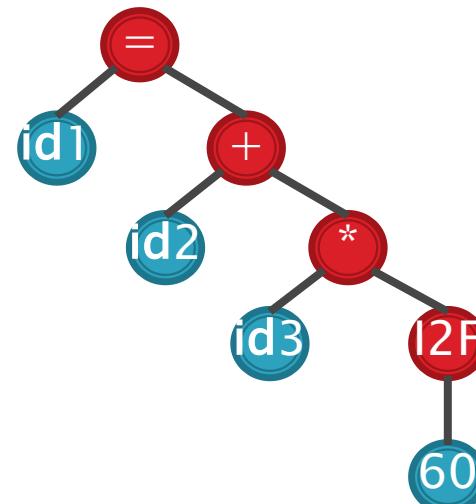
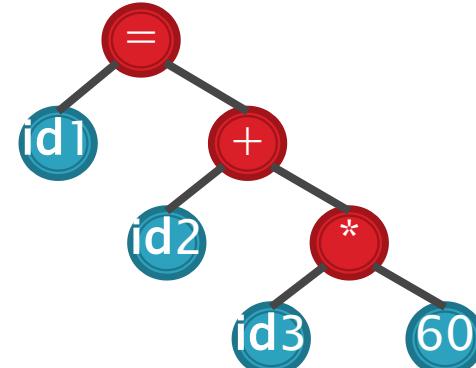
Проверка исходной программы на семантическую согласованность.

Сбор информации о типах переменных

Проверка типов

Таблица символов

1	position	float	...
2	initial	float	...
3	rate	float	...
...			



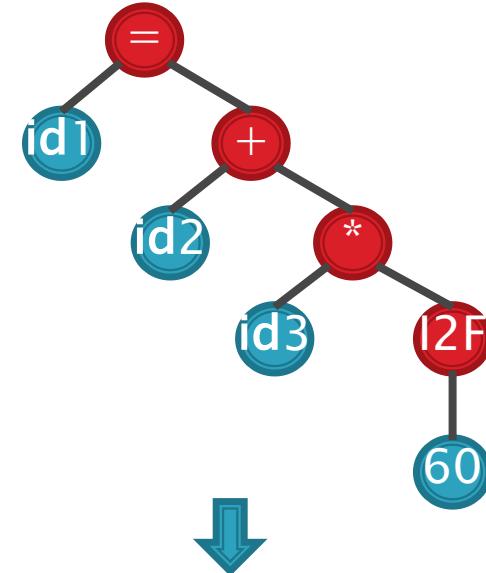
I2F = inttofloat

Генерация промежуточного кода

Генерация промежуточного представления (IR).

2 важных свойства IR:

- Должно легко генерироваться
- Должно легко транслироваться в целевой язык



```
t1 = inttofloat( 60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Оптимизация кода

Получение более качественного кода

```
t1 = inttofloat( 60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```



```
t1 = id3 * 60.0
id1 = id2 + t2
```

Генерация кода

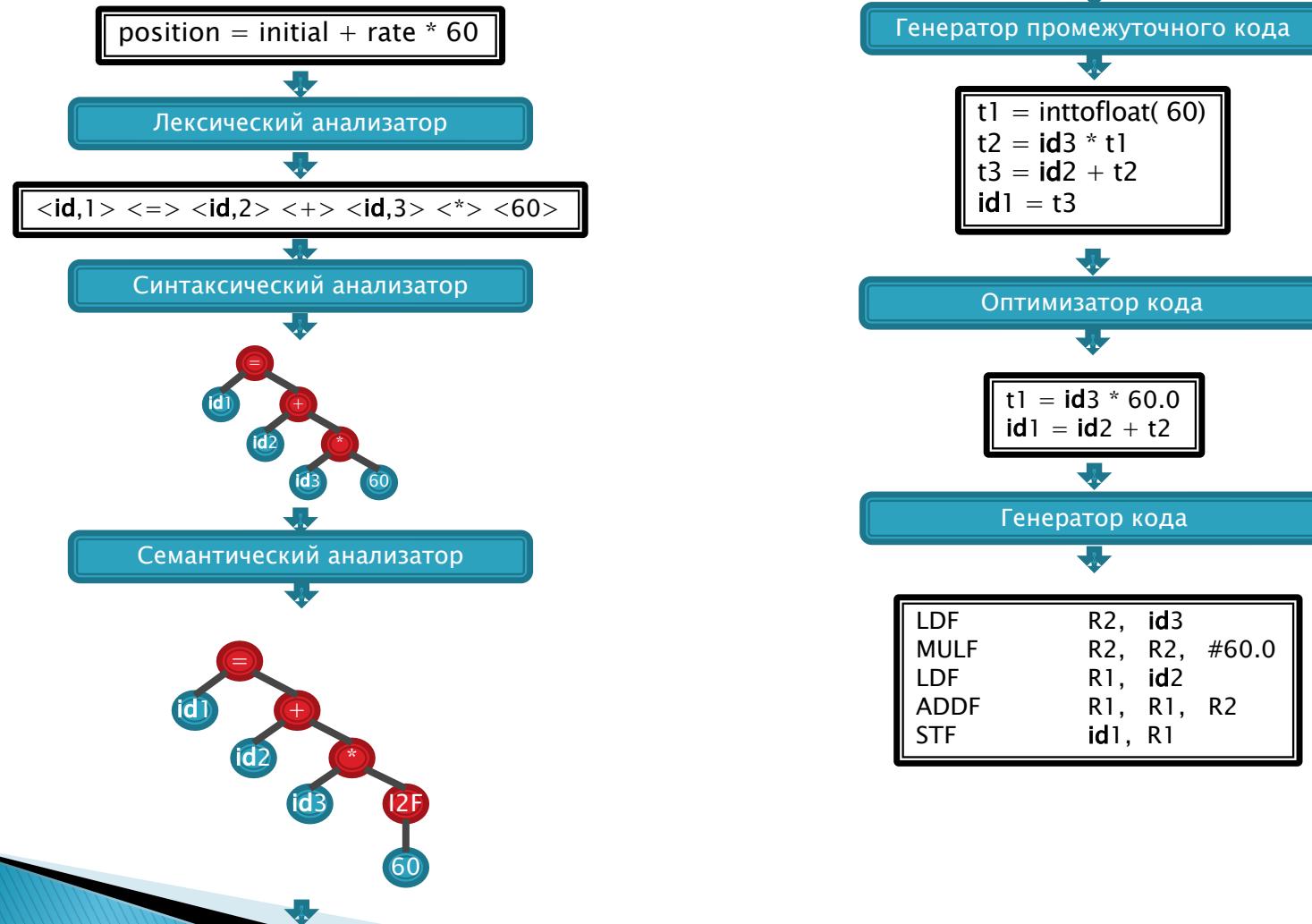
Отображение промежуточного представления в целевой язык

```
t1 = id3 * 60.0  
id1 = id2 + t2
```



```
LDF    R2,  id3  
MULF   R2,  R2,  #60.0  
LDF    R1,  id2  
ADDF   R1,  R1,  R2  
STF    id1, R1
```

Путь трансформации выражения



Изучение оптимизации кода

Оптимизация – *попытки* получить более эффективный код.

Требования к оптимизации:

1. Оптимизация должна быть корректной
2. Оптимизация должна повышать производительность многих программ
3. Время компиляции должно оставаться в разумных пределах
4. Требуемая инженерная работа должна быть осуществима

Классы оптимизаций

- ▶ Глобальные (межпроцедурные)
- ▶ Оптимизации потока данных
- ▶ Оптимизации потока управления
- ▶ Машинно-зависимые оптимизации

Пример межпроцедурной оптимизации (inline)

```
int factorial( int a)
{
    int r = 1;
    for ( ; a > 1; a--)
    {
        r *= a;
    }
    return r;
}

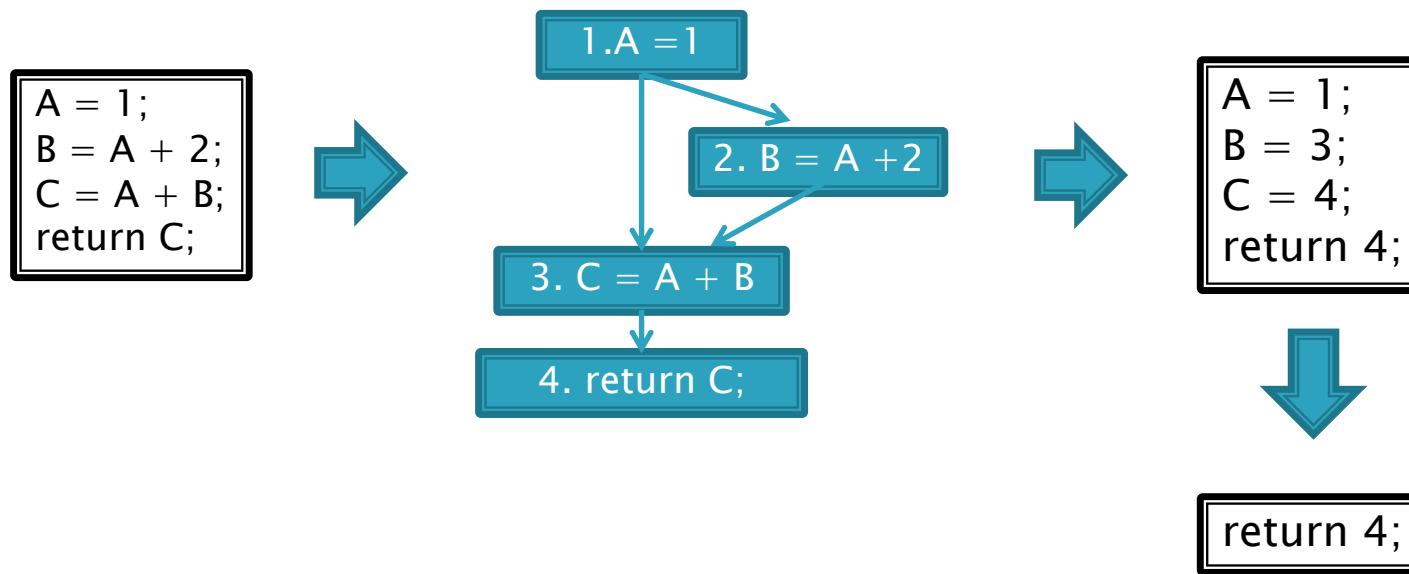
int func1( int i) { return factorial( i);}
int func2( void) { return func1( 3);}
```

```
int func2( void)
{
    int r = 1; int a = 3;
    for ( ; a > 1; a--)
    {
        r *= a;
    }
    return r;
}
```

```
int func2( void)
{
    int r = 1;
    r *= 3;
    r *= 2;
    return r;
}
```

```
int func2( void){return 6;}
```

Примеры оптимизаций потока данных



Пример оптимизации потока управления

```
x = 1;
for ( i=0;i<N;i++)
{
    x += 4;
}
```

```
x = 1;
for ( i=0;i<(N-2);i+=3)
{
    x += 4;
    x += 4;
}
if ( i < N)
{
    i++; x+=4;
    if ( i < N)
    {
        i++;x+=4;
    }
}
```

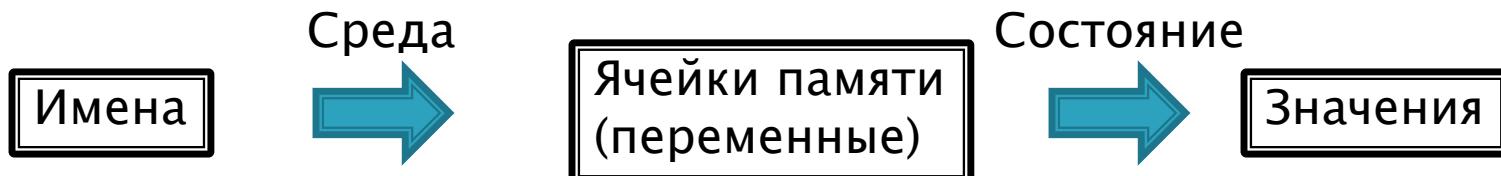
```
x = 1;
for ( i=0; i<(N-2);i+=3)
{
    x += 12;
}
if ( i < N)
{
    i++; x+=4;
    if ( i < N)
    {
        i++;x+=4;
    }
}
```

Понятия статического и динамического

Область видимости (scope) объявления x – область программы, в которой используется x из этого объявления.

Язык использует *статическую область видимости*, если можно определить область видимости объявления при рассмотрения исходной программы. В противном случае язык использует *динамическую область видимости*.

Среды и состояния



1. Среда (environment) — это отображение имен на ячейки памяти.
2. Состояние (state) – это отображение местоположений в памяти на их значения

```
...
int i; /* global i */
const int p = 5;
...
void f (...) {
    int i; /* local i */
    ...
    i = 3; /* use of local i */
    ...
}
...
x = i + 1; /* use of global i*/
```

Литература

- ▶ Ахо, Альфред В., Лам, Моника С., Сети, Рави, Ульман, Джейфри Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд. : Пер. с англ. – М. : ООО "И.Д. Вильямс", 2008.